• РАДИО И СВЯЗЬ •

СПРАВОЧНИК

ПРОГРАММИРОВАНИЕ МИКРОЭВМ НА ЯЗЫКЕ БЕЙСИК

СПРАВОЧНИК

Е.С.БАШМАКОВА
И.М.ВИТЕНБЕРГ
А.Б.ЛИБЕРОВ
А.Л.ПАШКОВ

ПРОГРАММИРОВАНИЕ МИКРОЭВМ НА ЯЗЫКЕ БЕЙСИК

ПОД РЕДАКЦИЕЙ
ПРОФЕССОРА
И.М.ВИТЕНБЕРГА

МОСКВА „РАДИО И СВЯЗЬ"
1991

ББК 32.973-01
П78
УДК 681.3.06: 519.682.2: 681.322-181.48(03)

Рецензенты: кандидаты физ.-мат. наук Г. В. Сенин, Г. Г. Гнездилова

Редакция литературы по информатике и вычислительной технике

П78 Программирование микроЭВМ на языке Бейсик: Справочник /Е. С. Башмакова, И. М. Витенберг, А. Б. Либеров, А. Л. Пашков; Под ред. И. М. Витенберга. – М.: Радио и связь, 1991. – 240 с.: ил.

ISBN 5-256-00762-9.

Описаны основные конструкции различных версий языка Бейсик, реализованных на ПЭВМ, а также основные средства языка: системные, графические, звуковые и ввода-вывода. Приведены программы, иллюстрирующие возможности языка для решения разнообразных задач. Описаны способы отладки программ.

Для инженеров, программистов и преподавателей учебных заведений.

П $\frac{2404010000-079}{046\ (01)-91}$ 151-91

ББК 32.973 – 01

Предисловие

Настоящий справочник ставит своей целью ознакомить читателя с основными правилами программирования персональных ЭВМ (ПЭВМ) на языке Бейсик. В книге представлены понятия языка, построение программ и приемы работы с программами.

Главы справочника несут различную нагрузку. В гл. 1, которая является вводной, содержатся сведения о языке программирования Бейсик для начинающих программистов, а также обзорная информация, полезная опытным программистам.

Материал, приведенный в гл. 2 и 3, предназначен для программистов, впервые сталкивающихся с языком Бейсик. В гл. 2 описаны синтаксические элементы программы, представление чисел и системы счисления, выполнение основных операций. В гл. 3 рассмотрены возможности языка для ввода данных в программу, вывода данных из программы, при работе с оперативной памятью и портами ввода-вывода, организации циклов, выполнении условных и безусловных переходов, использовании встроенных функций, а также при работе со строковыми переменными и организации подпрограмм.

Глава 4 ориентирована на пользователей, машины которых оснащены периферийными устройствами. В ней подробно описываются команды и операторы работы с файлами, печатающим устройством, некоторые специфические команды и операторы для работы с клавиатурой и экраном дисплея. Ряд инструкций применяется как в минимальных комплектах ПЭВМ, так и при широком использовании внешних устройств; эти инструкции рассматриваются в нескольких главах. Например, операторы ввода информации с клавиатуры (INPUT, LINE INPUT, INKEY$, INPUT$, GET и т. п.) и операторы вывода информации на печать (PRINT, PRINT USING) рассматриваются в гл. 3, а команды вывода информации на экран дисплея LIST и на печатающее устройство LLIST — в гл. 5.

В гл. 5, рассчитанной на программистов среднего уровня, содержится также информация о возможности тестирования и отладки программ, исправления ошибок. При работе с программами можно вводить программу заново, изменять ее, вызывать уже созданные.

Глава 6 предназначена для программистов, использующих в своей работе графику и звук.

Глава 7 дает некоторое представление о возможности построения достаточно простых программ, которые можно воспроизвести при изучении программирования.

Справочник построен по алфавитно-смысловому принципу. В начале каждой из основных глав приводится рисунок, играющий роль краткого обобщения содержания главы. Далее в смысловом порядке описаны основные понятия и определения, а затем в алфавитном порядке — инструкции языка, представленные ключевыми словами. В гл. 4 и 6 для лучшего усвоения материала описание инструкций разнесено по параграфам.

Авторы разделили информацию о ключевых словах по главам и частично по параграфам, что, по их мнению, должно облегчить изучение языка программистом, впервые приступившим к написанию программ. Для более подготовленного программиста в приложении 1 дан

перечень ключевых слов с указанием, в каких главах подробно рассмотрено назначение инструкции, ее формат и особенности.

В справочник включена информация о десяти версиях языка Бейсик; перечень этих версий приведен в первой главе. В приложении 1 и в каждой главе содержатся сведения о составе ключевых слов в каждой из версий.

В приложении 2 описывается внутреннее представление программ.

В каждой главе даны примеры, иллюстрирующие основные положения; в примерах использованы версии языков MBASIC и Бейсик-ПК8020, если дополнительно не оговорено.

В справочнике применяются термины "строковая переменная", "строковая операция" и т. п., так как понятие "строковый" существует во всех описанных версиях языка Бейсик.

При описании форматов инструкций в книге используются следующие специальные символы:

[...] – необязательный параметр;

(...) – обязательный параметр;

{...} – возможность выбора из нескольких вариантов;

<...>– клавиша;

/ – признак "или".

В тексте встречаются два символа: ¤ ("солнышко") и $ ("знак денежной единицы") – они синонимы, т. е. несут одинаковую смысловую нагрузку.

При подготовке справочника авторы пользовались в основном документацией фирм- и заводов-изготовителей машин; в списке литературы приведены издания, которые могут помочь более углубленному изучению языка.

Глава 1

Язык Бейсик и его место в системах программирования

1.1. Основные характеристики и версии языка Бейсик

Язык Бейсик — это распространенный язык программирования высокого уровня для микро- и персональных ЭВМ (ПЭВМ). Основные преимущества языка Бейсик:

простота создания и отладки программ;

диалоговый характер системы программирования на языке Бейсик, что позволяет проводить все операции по разработке, отладке и выполнению программ с помощью видеотерминального устройства;

широта распространения языка, что обеспечивает возможность использования ранее разработанных программ, а также перенос программ на другие ЭВМ;

возможность включения в программу фрагментов на машинном языке.

Основной класс пользователей языка Бейсик — непрофессиональные программисты, пользователи ЭВМ. Именно для облегчения работы таких пользователей и был разработан в 1965 г. в США, в Дартмутском колледже язык программирования, близкий по структуре и по названиям операторов к языку Фортран и названный BASIC (Beginner's All-purpose Symbolic Instruction Code, т. е. многоцелевой мнемокод для начинающих). Язык был создан по заказу фирмы General Electric в целях обеспечения легкого доступа к компьютеру в системе разделения времени для начинающих программистов. Поскольку авторы Бейсика создавали язык для программистов-новичков, то основным требованием они считали простоту языка. Другим требованием было упрощение общения человека с машиной. Нужна была система, которую легко освоить и которой удобно пользоваться.

Для языка Бейсик характерны: простой, легко запоминающийся синтаксис языка; малое число конструкций, которые необходимо освоить; однозначная семантика каждой конструкции.

Основные конструкции языка — предложения. В Бейсик включены лишь самые необходимые. Синтаксис языка простой — программа имеет строчную организацию, одна строка программы — это одно предложение. Вид всех предложений одинаковый — начинается номером, за которым находится ключевое слово и другая необходимая информация. Выражения составляются по обычным математическим правилам. В Бейсике можно работать со скалярами и организовывать массивы, т. е. использовать лишь наиболее простые и часто употребляемые структуры данных. Создатели Бейсика понимали, что присваивание воспринимается не просто. Ключевое слово LET (Пусть) в качестве предложения присваивания облегчает понимание. Чтобы не вводить новую синтаксическую единицу — метку, создатели языка решили при переходах использовать номера строк, которые прежде всего нужны для редактирования программы.

Интерактивный (диалоговый) язык Бейсик обеспечивает возможность работы программистов в режиме непосредственного взаимодействия с компьютером — в диалоговом режиме. Пользователь общается с машиной с помощью видеотерминального устройства, с которого он

вводит текст своей программы, редактирует программы, находящиеся в памяти, запускает программы на выполнение. Диалоговый режим замечателен тем, что компьютер немедленно реагирует на все действия пользователя, может подсказать, если человек, не зная как поступить в данной ситуации, обращается за помощью. В случае ошибки в программе пользователь может исправить ее сразу после того, как получил сообщение об ошибке,и попытаться продолжить или начать сначала выполнение программы. Диалоговый режим особенно удобен для разработки, тестирования и отладки программ, т. е. в процессе программирования.

Интерпретация обеспечивает простую диагностику ошибок: сообщения об ошибках выдаются в терминах языка, на котором написана программа, а не в терминах прерываний и адресов памяти машины. Платой за это удобство является замедление выполнения программы в 10 — 20 раз по сравнению с выполнением скомпилированного кода.

Предложения Бейсик-программы можно вводить в любой последовательности, а выполняться они будут в порядке возрастания их номеров. Такой порядок могут нарушать лишь предложения перехода. Чаще всего предложения нумеруются не подряд, а с шагом 10. Это облегчает вставку новых строк в программу.

Таким образом, Бейсик — не только язык программирования, но и единая система автоматизации программирования, включающая в себя интерпретатор, редактор и средства отладки. При работе с Бейсик-системой пользователь связан только с программой на языке Бейсик. Он набирает программу, вводя строки с номерами, и просит систему выполнить некоторые действия. Его не интересует, как организовано выполнение его команд в компьютере.

Язык Бейсик позволяет широко использовать внешние устройства в качестве устройств ввода-вывода и хранения информации. К таким устройствам относятся накопители на гибких магнитных дисках (НГМД), накопители на кассетной магнитной ленте (НКМЛ), печатающее устройство (ПУ), устройство отображения информации (дисплей), периферийные устройства ввода-вывода информации, расширяющие возможности ЭВМ (манипуляторы типа "мышь", джойстик, световое перо и т. п.); к внешним устройствам можно отнести также локальные вычислительные сети.

Язык Бейсик поддерживает работу двух основных классов графических устройств: устройств ввода графической информации и устройств вывода. К первому классу относятся манипуляторы "мышь", джойстик, графические планшеты разного типа. Сюда же относятся такие менее распространенные в персональных ЭВМ устройства, как световое перо, трекболл, сенсорный экран.

Разные версии языка Бейсик поддерживают работу с указанными в этом наборе устройствами графического ввода-вывода (УГВВ) двумя способами:

созданием специальных инструкций для определенного класса УГВВ;

созданием универсальных инструкций для различных классов УГВВ.

Основным устройством вывода графической информации является графический дисплей — графическое видеоконтрольное устройство (ГВКУ). К другим устройствам вывода относятся печатающее устройство и графопостроитель (плоттер), предназначенные для получения "твердой копии" изображения на экране графического дисплея, т. е. для переноса изображения на бумагу или другой носитель.

Устройства ввода графической информации также в основном используются только при совместной работе с графическим дисплеем.

Основными элементами при работе с ГКВУ являются графический экран и дисплейный процессор.

До 1975 г. язык Бейсик был реализован практически на всех типах ЭВМ, существовавших тогда в мире, и был наиболее распространенным диалоговым языком в системах с разделени-

ем времени. Рост популярности языка происходил одновременно с начавшимся в 1976 – 1978 гг. массовым распространением микроЭВМ и базирующихся на их основе персональных ЭВМ (ПЭВМ). Те и другие были ориентированы на однопользовательский диалоговый режим, и язык Бейсик стал основным языком программирования.

В 1978 г. Международная организация по стандартизации ISO приняла стандарт на так называемый "минимальный Бейсик" (X3.60 – 1978). Этот стандарт, определивший минимально необходимые языковые конструкции, стал основой, ядром многочисленных подмножеств языка, разработанных различными фирмами. Можно выделить два основных направления развития версий языка Бейсик и использующих его Бейсик-систем:

1. Бейсик-системы, встроенные в микроЭВМ, использующиеся в лабораторных, технологических и технических исследованиях. Особенно широко использовала язык Бейсик для таких систем известная фирма Hewlett Packard (США). Бейсик-системы такого рода включают в себя поддержку аппаратных особенностей используемого оборудования.

2. Универсальные Бейсик-системы диалогового типа, обеспечивающие разработку программного обеспечения ПЭВМ и предоставляющие пользователю доступ к аппаратным средствам.

Поскольку в первые годы развития ПЭВМ базовый комплект поставки не включал, как правило, накопителей на НГМД (это положение сохраняется и сейчас в бытовых ПЭВМ), Бейсик-системы располагались в постоянном запоминающем устройстве (ПЗУ) или программируемом ПЗУ (ППЗУ) и для них были характерны две особенности: малая емкость памяти собственно системы (от 5 до 16 Кбайт) и включение в их состав всего необходимого программного обеспечения для управления аппаратными средствами ПЭВМ. Это приводило как бы к сращиванию в одном ПЗУ самой Бейсик-системы и управляющих программ (драйверов) устройств ввода-вывода.

Таким образом, в зависимости от набора аппаратных средств ПЭВМ и емкости памяти ПЗУ число инструкций Бейсик-системы и их назначение могли сильно изменяться. Так возникли основные версии языка Бейсик, реализованные в ПЭВМ различных фирм:

на базе МП 6502 ПЭВМ фирмы Commodore, например PET, и фирмы Apple, например Apple-II;

на базе МП TMS9900 ПЭВМ фирмы Texas Instruments, например TI-99;

на базе МП Z80 ПЭВМ фирмы Radio Shack, например TRS-80 и фирмы Sinclair, например ZX-80;

на базе МП i8080 ПЭВМ фирмы Intel, например MDS-800.

Многие версии языка Бейсик созданы фирмой Microsoft, специализирующейся с 1975 г. на разработке программных реализаций Бейсик-систем. Версия языка для Бейсик-системы, известная под именем MBASIC, стала в конце 70-х – начале 80-х годов промышленным стандартом и ее разновидности были разработаны фирмой Microsoft для многих 8-разрядных микроЭВМ.

В 1981 г. для ПЭВМ фирмы IBM фирмой Microsoft была разработана новая версия языка Бейсик, получившая название BASICA. Эта версия и ее дальнейшие разновидности (MSX-BASIC и др.) стали промышленным стандартом для целого ряда 16-разрядных ПЭВМ.

Новые версии языка Бейсик включают в себя целый набор модулей, не определенных в "минимальном Бейсике": модули работы с дисковыми файлами, модули выполнения графических и звуковых инструкций. В последнее время широко обсуждался вопрос о новом стандарте ISO на язык Бейсик, имеются проекты нового стандарта: проект стандарта ANSI от 1984 г. и стандарт ECMA-116. Однако в этих документах предлагается создавать новые версии, сильно отличающиеся от сложившихся, фактических стандартов на язык Бейсик, которым удовлетворяют миллионы действующих на разных ПЭВМ программ. Поэтому в настоящем издании ссылки на эти стандарты приведены не будут, но при создании новых версий языка необходимо их изучение и использование.

1.2. Отечественные Бейсик-системы

Первой реализацией языка Бейсик в СССР стала система программного обеспечения для ЭВМ типа М-220, разработанная в 1970 – 1971 гг. группой под руководством Ю. Л. Кеткова. В дальнейшем были разработаны Бейсик-системы также и для БЭСМ-6. На всех отечественных микро- и персональных ЭВМ существуют Бейсик-системы, аналогичные тем или иным зарубежным системам. Так, в микроЭВМ Искра-226 основой системного программного обеспечения является Бейсик-2, аналогичный Бейсик-системе в ЭВМ WANG-220B. В ПЭВМ "АГАТ" имеется интерпретатор Бейсик-АГАТ, аналогичный системе BASIC Apple-II.

В микроЭВМ семейства "Электроника" может работать Бейсик-система, аналогичная системе BASIC PCKS для микроЭВМ фирмы DEC. В микроЭВМ на базе МП К580ИК80 (СМ1800, ЕС7970 и т. п.), как правило, используется стандартная система MBASIC.

В 1987 г. начался выпуск комплекса учебной вычислительной техники (КУВТ) "КОРВЕТ". Основным языком программирования является Бейсик. Бейсик-система КУВТ "КОРВЕТ" позволяет работать с дисковыми файлами, имеет расширенный набор устройств ввода-вывода (УВВ), может работать в локальной сети, имеет накопители на кассетной магнитной ленте. Поскольку ПЭВМ, входящие в КУВТ "КОРВЕТ" (ПК8020 и ПК8010), базируются на МП типа К580ИК80А, а Бейсик-системы с указанными выше возможностями на МП такого типа не существует, была разработана Бейсик-система с ядром MBASIC и модулями собственной разработки.

Семейство отечественных профессиональных ПЭВМ типа ЕС1840, ЕС1841 и последующие имеет Бейсик-систему, аналогичную системе BASICA.

Существует также отечественная реализация оригинальной версии языка BASIC/F, разработанная А. Е. Корчаком на Рижском производственном объединении VEF им. В. И. Ленина.

1.3. Основные отличия версий языка Бейсик

Ниже будут рассмотрены только основные отличия каждой из версий языка Бейсик; более конкретные отличия, касающиеся инструкций (команд, операторов, функций) языка, будут приведены в соответствующих главах при описании инструкций.

Рассмотрим версии языка, наиболее характерные для периода 1976 – 1986 гг. и часто используемые в качестве основы для построения Бейсик-систем:

1. Бейсик ГОСТ 27787–88;
2. **XYBASIC*** (фирма Mark Williams);
3. Бейсик SINCLAIR ZX SPECTRUM+2 (фирма AMSTRAD CONSUMER ELECTRONICS – **Бейсик-Спектрум+2);**
4. Бейсик Apple-II (**Бейсик-АГАТ**);
5. **Бейсик-TRS–80** (фирма Microsoft);
6. **MBASIC v. 5.0** (фирма Microsoft);
7. Бейсик-КОРВЕТ (ПК8010) – Бейсик-ПК8010;
8. Бейсик-КОРВЕТ (ПК8020) – Бейсик-ПК8020;
9. **MSX-BASIC** (фирма Microsoft);
10. **BASICA** (фирма Microsoft) для PC IBM (аналогично ЕС1840).

Основные характеристики версий языка Бейсик, используемые в конкретных реализациях машин и систем, представлены в табл. 1.1.

---

* Полужирным шрифтом выделены названия версий языка Бейсик, используемые в дальнейшем тексте.

Таблица 1.1

Характеристика	Номер версии								
	2	3	4	5	6	7	8	9	10
Число арифметических операций	9	6	6	6	8	8	8	8	8
Число логических операций	4	4	3	3	6	6	6	6	6
Число строковых функций	11	5	2	9	15	15	15	15	15
Число арифметических функций	12	13	4	15	15	15	15	15	15
Число битовых операций	7	-	-	-	-	-	-	-	-
Использование последовательных файлов	+	-	-	+	+	+	-	+	+
Использование файлов произвольного доступа	-	-	-	+	+	-	-	+	+
Число операторов работы с памятью	11	7	4	7	8	9	9	10	10
Число графических операторов	-	3	8	4	-	10	10	10	10
Число звуковых операторов	-	2	-	-	-	3(*)	1	3	3
Редактор	Ком.	Вс.	Пер.	Ком.	Ком.	Ком.	Ком.	Вс.	Вс.

Окончание табл. 1.1

Характеристика	Номер версии 2	3	4	5	6	7	8	9	10
Наличие операторов работы с локальной сетью	-	-	-	-	-	+	+	+	+
Наличие функциональной клавиатуры	-	-	-	-	-	+	+	+	+

Примечание. Ком. — командный редактор; Вс. — встроенный редактор; Пер. — перебор строки; * — только в одной из модификаций

Объем и содержание языка в различных версиях, в том числе разработанных в одной фирме, сильно отличаются. Это отличие затрагивает не только бо́льшую часть отдельных инструкций языка, но связано с включением в более поздние версии целых классов новых инструкций (графика, звук и т. п.). Поэтому естественным является стремление к стандартизации языка с обеспечением максимальной переносимости разработанного программного обеспечения.

Работы по стандартизации языка Бейсик за рубежом наиболее активно ведутся в США силами американской и международной организации по стандартизации ANSI и ISO. Однако, на наш взгляд, эта работа не обеспечит переносимости большого числа (десятки миллионов экземпляров) созданных и разрабатываемых промышленных программ для распространенных ПЭВМ типа IBM PC, MSX и др. Кроме того, проекты нового стандарта пока находятся в стадии предварительного обсуждения.

Используя результаты проведенного анализа, можно сформулировать основные требования к стандарту языка Бейсик:

реализация всех операторов стандартной версии языка на отечественных ПЭВМ;

возможность развития языка;

учет зарубежных промышленных стандартов.

1.4. Основные концепции Государственного стандарта языка Бейсик

Переносимых программ на языке Бейсик, удовлетворяющих определенным и достаточно жестким требованиям, сравнительно немного. Вместе с тем существует потребность в создании переносимого программного обеспечения на языке Бейсик для достаточно сложных задач. Кроме того, стандартизация языка Бейсик должна обеспечить поддержку наиболее удачно разработанных версий языка для разных по архитектуре и аппаратному обеспечению ПЭВМ.

С 1 июля 1989 г. введен в действие ГОСТ 27787–88 на язык программирования Бейсик.

Исходя из вышеназванных требований, стандарт языка Бейсик построен по схеме "Ядро плюс модули". Ядро содержит описание операторов и функций, обязательных к реализации на всех ЭВМ, имеющих в составе программного обеспечения язык Бейсик, независимо от их

архитектуры и комплектации. Каждый модуль содержит описание операторов и функций, реализация которых зависит от архитектуры и от комплектации ЭВМ, имеющих в составе программного обеспечения язык Бейсик. Ядро составляет "минимальный Бейсик", вторично определенный стандартом Международной организации по стандартизации в 1984 г.

В стандарте рассматриваются следующие модули:

расширения основных средств;

графических средств;

работы с накопителями на магнитных дисках;

работы с накопителями на магнитных лентах;

командный.

По мере развития аппаратных и программных средств предусматривается пополнение как состава каждого модуля, так и набора модулей в целом. Совокупность ядра и модулей обеспечивает гибкость выбора фиксированного подмножества стандарта, в максимальной степени удовлетворяющего конкретным потребностям пользователей и комплектации оборудования. Кроме того, ГОСТ на язык Бейсик согласован с готовящимся сейчас новым стандартом ISO языка Бейсик тоже за счет выбора подходящего подмножества стандарта.

По аналогии со стандартом "минимального Бейсика" в ядре описаны синтаксис и семантика языковых конструкций, приведен перечень обнаруживаемых отклонений от стандарта, обязательный для каждой соответствующей стандарту реализации, а также рекомендации по истолкованию трудных для понимания фрагментов описания ядра. Таким образом, стандарт языка Бейсик сделан закрытым, т. е. в него включены требования к поведению языкового процессора при обработке отклонений от стандарта. Подобное требование позволит обеспечить унификацию диагностики.

В ядре зафиксированы также свойства языка, которые не определяются стандартом или засисят от реализации. Ясно, что использование таких свойств языка в программах снижает их переносимость. Поэтому при описании стандарта в него включены рекомендации по обеспечению переносимости программ для включенных в стандарт языковых конструкций.

В целом в стандарте описаны принципы формирования номенклатуры и состава модулей, выработана адекватная русская терминология, учтены требования Государственной системы стандартизации. Одним из критериев формирования состава модулей явилось обеспечение преемственности и переносимости программного обеспечения, создаваемого на языке Бейсик. В первую очередь это относится к реализации языка Бейсик для ПЭВМ, так как этот тип ЭВМ является наиболее массовым и наиболее подверженным изменениям (по составу и характеристикам). Установив заранее синтаксис и семантику операторов и функций, можно ожидать, что программы, разработанные в настоящее время на некоторых ЭВМ, в будущем смогут работать на всех ЭВМ.

При выборе модулей и определении их состава учитывалась также зависимость от состава и характеристик периферийных устройств и характеристик самой ЭВМ.

1.5. Загрузка интерпретаторов и выход в операционную систему

Интерпретаторы рассматриваемых версий языка Бейсик занимают разную память, могут работать в операционных системах CP/M, ISIS-II, МикроДОС, MSDOS и CP/M-86, а также могут быть встроенными в ППЗУ.

1.5.1. XYBASIC

Полный формат команды загрузки XYBASIC:

```
XYBASIC <возврат каретки>
```

Когда интерпретатор загружен, на экране появляется сообщение

```
XYBASIC {version} REV n.m
COPYRIGHT 1978,1979,1980  BY MARK WILLIAMS COMPANY
CHICAGO
WIDTH?
END OF MEMORY?
xxxxx BYTES FREE
OK
```

xxxxx – количество свободных байтов.

Первая строка указывает на то, что вы используете версию n.m. XYBASICa. Сообщение {version} может быть CP/M или ISIS-II, кроме того, если версия включает команды редактирования текста, в сообщении указывается EDIT, а если версия включает дисковые команды – DISK.

Вторая строка информирует о фирме-разработчике данной версии интерпретатора и о лицензионной чистоте данного программного продукта.

Третья строка запрашивает длину строки терминала. Эта информация необходима для организации форматированного вывода. Ответ должен быть десятичным числом (до 255), за которым следует символ <возврат каретки>. Если сразу вводится <возврат каретки>, то по умолчанию принимается длина строки, равная 80 символам.

Четвертая строка запрашивает память для программы пользователя. Ответ должен быть десятичным числом, которое является наибольшим адресом оперативной памяти в системе компьютера, за которым следует <возврат каретки>. Если необходимо зарезервировать память для подпрограмм на машинном языке, нужно ввести максимальный адрес, который может использовать интерпретатор XYBASIC. Если будет введен только <возврат каретки>, то XYBASIC найдет и будет использовать наибольший допустимый адрес автоматически. В CP/M и ISIS-II эта информация поступает из операционной системы.

Пятая строка сообщает, сколько байтов памяти остались свободными для программ и переменных.

Для выхода из интерпретатора в операционную систему необходимо нажать клавиши Ctrl–B.

1.5.2. Бейсик-Спектрум+2

После того как интерпретатор полностью загружен, появляется тестовый сигнал: на экране телевизора присутствуют цветные полосы, а из динамика слышится постоянный тоновый сигнал.

Для выключения тестового сигнала следует нажать клавишу RESET (Повторный запуск). Тестовый сигнал исчезнет, а на экране появится начальное меню: 1–загрузчик магнитной ленты; 2–Бейсик-128; 3–калькулятор; 4–Бейсик-48.

Начальное меню появляется на экране при включении компьютера либо после нажатия клавиши RESET и предлагает выбор одной из четырех функций. Функции меню появляются выделенными на экране с помощью полосы подсветки. Используя клавиши управления курсором, выбирается функция "Бейсик-128". После этого выбор функции подтверждается нажатием клавиши ENTER (Ввод).

Теперь компьютер переключился в режим "Бейсик-128". Вы увидите заголовок в нижней части экрана и мерцающий курсор в левом верхнем углу.

Для возврата в начальное меню необходимо использовать меню редактирования, которое вызывается с помощью нажатия клавиши EDIT (Редактирование). Вновь с помощью клавиш управления курсором и клавиши ENTER выбирается функция EXIT (Выход) для возврата в начальное меню.

1.5.3. Бейсик-АГАТ

Для обращения к интерпретатору, находящемуся в ПЗУ, из программы "Системный монитор" используются следующие команды:

***E0C0G или УПР-РЕГ**

для обращения к интерпретатору без сохранения программы и состояния интерпретатора;

***E003G или УПР-РЕГ-S**

для обращения к интерпретатору с сохранением программы и состояния интерпретатора.

При наличии в ПЭВМ "АГАТ" дисковой операционной системы для обращения к интерпретатору можно нажать клавишу СБРОС. При этом программа и состояние интерпретатора сохраняются.

После загрузки интерпретатора на экране дисплея появляется приглашение к работе в виде символа "]" и мигающий курсор за ним.

1.5.4. Бейсик-TRS-80

После включения компьютера в сеть на экране дисплея появляется сообщение MEMORY SIZE? (Размер памяти?). Существует два варианта ответа на указанный запрос. Если предполагается работа с программами на машинном языке, загружаемыми в специально отведенный сегмент памяти, то ответ должен быть командой SYSTEM. Если предполагается работа с программами на языке Бейсик, следует нажать только клавишу ENTER (Ввод) без нажатия каких-либо других клавиш. Это позволит при написании Бейсик-программы использовать всю память (для 4К версии – 3284 байт; для 16К версии – 15572 байт).

После нажатия клавиши ENTER (Ввод) на экране появляется сообщение:

```
RADIO SHACK LEVEL II BASIC
READY
>_
```

В Бейсик-TRS–80 существуют 4 режима работы:

командный (Command) – в 4К версии это режим калькулятора;

программный (Execute) – выполнение командой RUN Бейсик-программ;

(В 4К версии используются только эти режимы.)

редактирование (Edit) – только в 16К версии; позволяет редактировать строки программы;

машинный (Monitor) – позволяет загружать объектные файлы в память.

1.5.5. MBASIC v.5.0.

Для вызова интерпретатора необходимо ввести команду MBASIC. Полный формат команды:

MBASIC [спецификация файла] [/F:число файлов] [/S:размер записи] [/M:рабочая область] <возврат каретки>

спецификация файла – спецификация программного файла, который будет загружаться и выполняться. Это строковая константа, не заключенная в кавычки;

/F:число файлов – максимальное число файлов, которые могут быть открыты в любое время в течение работы интерпретатора. Каждый блок файла данных требует 166 байт памяти. Если этот параметр опущен, то по умолчанию число файлов равно 3. Максимальное значение параметра /F: – 15;

/S:размер записи – максимальный размер записи для файлов произвольного доступа. По умолчанию размер записи равен 128 байтам.

/M:рабочая область – максимальная память, которая может быть использована интерпретатором как рабочее пространство. Максимальное значение – 64 Кбайта. Если этот параметр не указан, то используется вся доступная память.

Все числа в данных параметрах могут быть заданы в десятичном, восьмеричном и шестнадцатеричном представлении.

Когда интерпретатор загружен, на экране дисплея появляется сообщение:

```
BASIC-80 REV. 5.0.
[CP/M Version]
Copyright 1977, 78, 79, 80 (C) by Microsoft
Created:14-Jul-80
xxxxx Bytes free
Ok
```

xxxxx – количество свободных байтов.

Для выхода из интерпретатора в операционную систему используется команда SYSTEM.

1.5.6. Бейсик-КОРВЕТ

В КУВТ "КОРВЕТ" имеются два интерпретатора языка Бейсик: один размещен в постоянной памяти (Бейсик-ПК8010), другой на диске (Бейсик-ПК8020).

Загрузка Бейсик-ПК8010. После включения питания подается короткий звуковой сигнал и выдается сообщение:

```
ОПТС n.m
```

После окончания ОПТС (оперативной проверки технических средств) выдается сообщение:

```
Бейсик КОРВЕТ в.n.m
Москва 19xx г.

Ok
```

n.m. – номер версии программного обеспечения;

xx – год.

Загрузка Бейсик-ПК8020. Для вызова интерпретатора языка Бейсик необходимо ввести команду BASIC. Полный формат команды BASIC:

```
BASIC[спецификация файла][/F:числофайлов][/M:размер]

<возврат каретки>
```

спецификация файла – имя программы, которая будет загружаться и выполняться. Это строковая константа, не заключенная в кавычки;

/F:число файлов – максимальное число файлов, которые могут быть открыты во время работы интерпретатора. Каждый блок файла данных требует 166 байт памяти. Если этот параметр не указан, то по умолчанию число файлов равно 3. Максимальное значение /F: – 15.

/M:размер – максимальная память, которая может быть использована интерпретатором как рабочее пространство. Если этот параметр не указан, то используется максимальная доступная память 48 Кбайт.

После загрузки интерпретатора на экране дисплея появляется сообщение:

```
Бейсик КОРВЕТ в.п.м.
Москва
МикроДОС 19xx

Ok
```

п.м. – версия программного обеспечения;

xx – год.

Для возврата в операционную систему используется команда SYSTEM.

1.5.7. MSX-BASIC

Программа MSX-BASIC расположена в постоянной памяти. Возможны два способа вызова интерпретатора. Певый – если операционная система не загружена, и второй – если операционная система загружена.

Дисковый вариант MSX-BASIC. После загрузки операционной системы и появления подсказки для входа в Бейсик-систему необходимо набрать команду BASIC, полный формат которой выглядит так:

BASIC [спецификация файла] <возврат каретки>

спецификация файла – имя программного файла, который будет загружаться и выполняться. Это строковая константа, не заключенная в кавычки.

После загрузки интерпретатора на экране дисплея появляется сообщение:

```
MSX BASIC version 1.0
Copyright 1983 by Microsoft
xxxxx Bytes free
Disk BASIC version 1.0
Ok
```

xxxxx – количество свободных байтов.

Для возврата в операционную систему необходимо набрать оператор CALL SYSTEM или команду _SYSTEM.

MSX-BASIC в ПЗУ. После включения питания при отключенном дисководе и незагруженной операционной системе на экране дисплея появляется сообщение:

```
MSX SYSTEM
version  1.0
Copyright 1983 by Microsoft
```

Затем после некоторого времени появляется запрос:

```
Enter date (M-D-Y):
```

который требует ввести дату начала работы с интерпретатором.

После ответа на экране появляется сообщение, аналогичное дисковой версии, и интерпретатор готов к работе.

1.5.8. Расширенный Бейсик IBM PC (ЕС 1840) – BASICA

Для вызова интерпретатора необходимо ввести команду BASICA. Полный формат команды BASICA:

BASICA [спецификация файла][/F:число файлов][/S:размер записи][/C:буфер][/M:рабочая область]<возврат каретки>

спецификация файла – имя программного файла, который будет загружаться и выполняться; это строковая константа, не заключенная в кавычки;

/F:число файлов – максимальное число файлов, которые могут быть открыты в любое время в течение работы интерпретатора; каждый файл требует 188 байт памяти для блока управления файлом плюс размер, заданный в параметре /S:; если параметр /F: не указан, то по умолчанию число файлов равно 3; максимальное значение /F: равно 15;

/S:размер записи – размер записи для файлов произвольного доступа; параметр длины записи в операторе OPEN не может превышать это значение; по умолчанию размер записи равен 128 байт, максимальное значение, которое можно ввести – 32767; рекомендуется использовать /S:512 при использовании файлов произвольного доступа;

/C:буфер – размер каждого буфера связи, максимальное значение – 32767; если этот параметр не указан, то резервируется 256 байт для буфера приема и 128 – для буфера передачи; при наличии линии высокой скорости передачи рекомендуется /C:1024;

/M:рабочая область – максимальная память, которая может быть использована интерпретатором как рабочая область, максимальное значение – 64 Кбайт; если этот параметр не указан, то используется вся доступная память.

Все числа в данных параметрах могут быть заданы в десятичном, восьмеричном или шестнадцатеричном виде.

Когда интерпретатор загружен, на экране появляется сообщение:

```
The IBM Personal Computer Basic
Version A2.00 Copyright IBM Corp. 1981, 1982, 1983
xxxxx Bytes free

Ok
```

xxxxx – количество свободных байтов.

Для возврата в операционную систему используется команда SYSTEM.

1.6. Трансляторы языка Бейсик

Существуют два основных типа трансляторов языка Бейсик: интерпретирующего и компилирующего типов – интерпретаторы и компиляторы.

Интерпретатор непосредственно выполняет (интерпретирует) исходный текст программы, вводимый в ОЗУ либо с клавиатуры, либо с внешнего запоминающего устройства.

Компилятор языка Бейсик по функциям ничем не отличается от транслирующих систем подобного типа с других языков.

Существуют также трансляторы промежуточного типа, называемые интерпретаторами компилирующего типа.

В основном ПЭВМ оснащаются интерпретаторами, позволяющими реализовать все возможности диалоговой системы программирования. Как было сказано выше, интерпретаторы замедляют выполнение программы в 10 – 20 раз. Если в программе требуется работа с реаль-

ными объектами или необходимо многократно использовать разработанную программу, то обычно создание программы проводят в два этапа: сначала разрабатывают и отлаживают программу с помощью интерпретатора, а затем пропускают ее через компилятор, получая, таким образом, перемещаемый объектный модуль.

Транслятор с языка Бейсик фирмы Microsoft, который обычно называется BASCOM, представляет собой набор дополнительных транзитных программ, который может быть включен в стандартный набор транзитных программ операционных систем CP/M, МикроДОС или MSDOS и может использоваться совместно с интерпретатором BASIC фирмы Microsoft (MBASIC, BASICA и т. п.).

Набор программ BASCOM предоставляет средство, с помощью которого файлы с исходным текстом программы на языке Бейсик могут быть преобразованы (оттранслированы) в файлы, содержащие перемещаемый объектный код. Эти файлы с перемещаемым объектным кодом могут быть далее объединены с другими файлами в процессе окончательной сборки и создания файла с исполняемым машинным кодом.

В набор программ BASCOM входит файл BASCOM.COM, который содержит собственно компилятор с языка Бейсик, а также библиотечные файлы. Для выполнения компиляции файла вводится команда:

BASCOM имя файла.BAS

По завершении работы компилятора создается файл с расширением .REL, который содержит перемещаемый объектный код исходной программы. Исполняемый машинный код получается после работы программы LINK-80.

Глава 2

Основные элементы языка Бейсик

2.1. Режимы работы

Когда интерпретатор готов к работе, на экран выводится подсказка. Это состояние называется уровнем команд.

В это время интерпретатор может быть использован в одном из двух режимов: в прямом (командном) или в программном. Как правило, в рассматриваемых версиях признаком готовности интерпретатора является сообщение "Ok", только в Бейсик TRS-80 — это сообщение "READY", в XYBASIC — "OK", а в Бейсик-АГАТ — символ "]".

Прямой (командный) режим — это режим непосредственных вычислений, он удобен для отладки. Интерпретатор выполняет задание сразу после его введения. В этом режиме оператор предварительно не определяется номером строки. Можно вывести на экран результаты работы арифметических или логических операций и запомнить их для дальнейшего использования, но сами задания не сохраняются после выполнения. Например:

```
PRINT 20+5
 25
Ok
```

В программном режиме вводят и выполняют программы. Вводимая строка, которая является частью программы, должна начинаться с номера. Затем строка запоминается как часть программы в памяти. Программа может быть выполнена при введении команды RUN. Например:

```
10 PRINT 20+5
RUN
 25
Ok
```

Любая вводимая директива должна заканчиваться нажатием клавиши возврата каретки.

2.2. Формат программной строки

Программа задается в виде последовательности программных строк, которые описывают алгоритм решения задачи.

В языке Бейсик программные строки имеют следующий формат:

nnnnn оператор [:оператор...]<возврат каретки>

nnnnn – номер строки.

В табл. 2.1. приведены минимальный и максимальный номера строк, которые можно использовать при написании программ.

Операторы делятся на исполнительные и неисполнительные. Каждый оператор – это ключевое (резервированное) слово, определяющее характер действия компьютера. Вслед за ним располагается информация, необходимая для указанного действия. Ключевые слова должны отделяться пробелами.

Таблица 2.1

Версия	Номер строки	
	минимальный	максимальный
Стандарт	>0	--
XYBASIC	1	65535
Бейсик-Спектрум+2	1	9999
Бейсик-АГАТ	0 (1)	3999(32767)
Бейсик-TRS-80	0	65529
MBASIC	0	65529
Бейсик-ИК8010	0	65529
Бейсик-ИК8020	0	65529
MSX-BASIC	0	65529
BASICA	0	65529

Исполнительный оператор – это инструкция программы, которая сообщает интерпретатору, что делать при выполнении программы (например, GOTO, PRINT).

Неисполнительный оператор – это оператор, который не вызывает у интерпретатора никаких программных действий (например, REM, DATA).

В одной строке может быть несколько операторов, они должны разделяться двоеточием (:), общее число символов в строке не должно превышать 255.

Как правило, любая система программирования накладывает ограничение на число строк в программе. Объединение нескольких операторов в одной программной строке позволяет разрабатывать программы, в которых число операторов может быть увеличено в несколько раз.

Вводимые в память компьютера программные строки упорядочиваются по возрастанию номеров строк, что позволяет добавлять новые строки между введенными ранее или стирать ненужные. По номеру строки осуществляется безусловный переход или вызов подпрограммы. Сообщение об ошибке выводится с указанием номера строки, где данная ошибка произошла.

При составлении программ рекомендуется нумеровать строки с интервалом в пять, десять или сто, что позволит, при необходимости, вставить в программу дополнительные строки.

В рассматриваемых версиях языка, кроме Бейсик-Спектрум+2, имеются средства для автоматической генерации номеров строк с заданным интервалом (команда AUTO) и средства для перенумерации строк с определенного номера и с заданным интервалом (команда RENUM). Версия Бейсик-Спектрум+2 осуществляет перенумерацию строк с помощью меню встроенного редактора.

2.3. Синтаксические элементы программы

Основными элементами языка Бейсик являются выражения, операции и данные (рис. 2.1.).

Программа на языке Бейсик записывается с использованием следующих символов:

прописные и строчные буквы латинского алфавита от A до Z и, если реализовано, буквы русского алфавита от А до Я (буквы русского алфавита могут использоваться только в строковых константах и комментариях);

арабские цифры от 0 до 9;

специальные символы:

Рис. 2.1

= Знак равенства или символ присваивания
+ Знак сложения
– Знак вычитания
* Знак умножения
/ Знак деления
^ Знак экспоненты (степени)
(Левая круглая скобка
) Правая круглая скобка
% Знак процента
# Знак номера (или фунт)
$ Знак денежной единицы (или доллар)
! Восклицательный знак
, Запятая
. Точка
' Апостроф
; Точка с запятой
: Двоеточие
& Амперсанд
@ Коммерческое "AND"
? Знак вопроса
< Знак "меньше чем"
> Знак "больше чем"
\ Знак целочисленного деления
_ Знак подчеркивания
" Двойные кавычки

символы пропусков: пробел и горизонтальная табуляция.

Синтаксическими элементами программы (рис. 2.2.) являются имена переменных, ключевые (резервированные) слова, константы и ограничители (простые и составные).

В большинстве рассматриваемых версий при вводе программы с клавиатуры ключевые словаможно записывать как прописными, так и строчными буквами. Интерпретатор переводит их в строчные, если они не являются частью строки, заключенной в кавычки, комментарием или данными. Интерпретатор Бейсик-Спектрум+2 переводит в строчные буквы только ключевые слова.

Имя переменной – набор символов, определяющий переменную. Имя переменной должно обязательно начинаться с буквы. Число символов, входящих в имя, зависит от версии языка или интерпретатора машины. Состав символов, входящих в имя переменной, для разных версий представлен в табл. 2.2.

Имя переменной определяет также и тип переменной с помощью специального символа, замыкающего имя. Имена переменных, отличающиеся этим специальным символом, не идентичны между собой, например A%, A$, A! и A# присваиваются разным переменным. Специальные символы после имени переменной означают:

% – переменная целочисленная;
# – переменная вещественная, двойной точности;
! – переменная вещественная, одинарной точности;
$ – переменная строковая.

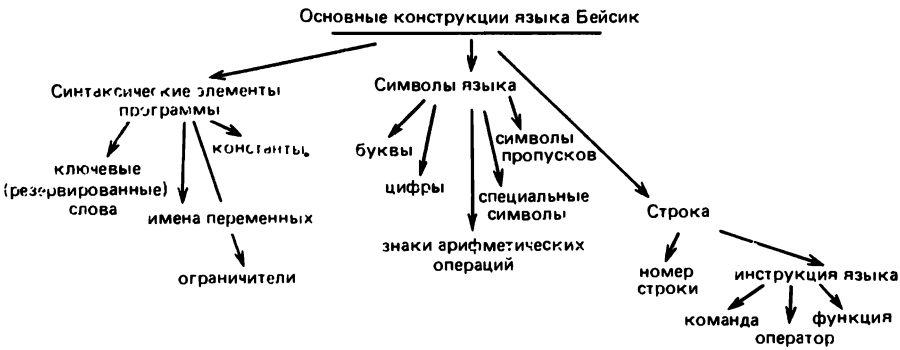


Рис. 2.2

Таблица 2.2

Версия	Число символов в имени переменной (без специального символа)	Примечание
Стандарт	не менее двух	Первый символ должен быть буквой, остальные буквами или цифрами
XYBASIC	Любой длины, запоминаются первые восемь символов	Первый символ должен быть буквой, остальные - буквами или цифрами. Ключевое слово не может быть именем или частью имени переменной.
Бейсик-Спектрум+2	Любой длины	Первый символ должен быть буквой, другие буквами или цифрами. Пробелы игнорируются и все буквы внутри переводятся в строчные. Ключевое слово может быть частью имени, если оно не обрамлено пробелами. Имя строки состоит из одной буквы, за которой следует символ $. Переменные управления в FOR...NEXT циклах имеют имена длиной в одну букву. Числовые массивы имеют имена длиной в одну букву, которая может совпадать с именем простой переменной. Строковые массивы имеют имена длиной в одну букву, за которой следует символ $, и не могут совпадать с именем переменной.
Бейсик-АГАТ	Два символа	Первый символ должен быть буквой, второй буквой или цифрой. Ключевое слово не может быть именем или частью имени переменной.
Бейсик-TRS-80	До 255 символов, но запоминаются первые два символа	Первый символ должен быть буквой, остальные буквами или цифрами. Ключевое слово не может быть именем или частью имени переменной

Окончание табл. 2.2

MBASIC	40 символов	Первый символ должен быть буквой, остальные буквами, цифрами или точкой. Ключевое слово не может быть именем или частью имени переменной.
Бейсик-ПК8010	Любой длины, но запоминаются первые два символа	Первый символ должен быть буквой, остальные буквами или цифрами. Ключевое слово не может быть именем или частью имени переменной
Бейсик-ПК8020	Любой длины, но запоминаются первые два символа	Первый символ должен быть буквой, остальные буквами или цифрами. Ключевое слово не может быть именем или частью имени переменной
MSX-BASIC	Любой длины, но запоминаются первые два символа	Первый символ должен быть буквой, остальные буквами или цифрами. Ключевое слово не может быть именем или частью имени переменной
BASICA	Любой длины, но запоминаются первые сорок символов	Первый символ должен быть буквой или цифрой, остальные буквами или цифрами. Ключевое слово может быть частью имени или именем переменной.

В табл. 2.3. представлена информация о наличии разных типов переменных в рассматриваемых версиях.

Таблица 2.3

Версия	Специальные символы				Тип переменной по умолчанию
	$	%	!	#	
Стандарт	+	+	+	+	Числовая переменная одинарной точности (!)
XYBASIC	+	+	+		Числовая переменная с плавающей точкой (!)
Бейсик-Спектрум+2	+				Числовая переменная

Окончание табл. 2.3

Версия	Специальные символы				Тип переменной по умолчанию
	$	%	!	#	
Бейсик-АГАТ	+	+			Числовая переменная с плавающей точкой
Бейсик-TRS-80	+	+	+	+	Числовая переменная одинарной точности (!)
MBASIC	+	+	+	+	То же
Бейсик-ПК8010	+	+	+	+	—"—
Бейсик-ПК8020	+	+	+	+	—"—
MSX-BASIC	+	+	+	+	Числовая переменная двойной точности (#)
BASICA	+	+	+	+	Числовая переменная одинарной точности (!)

Для задания типа переменной могут быть использованы операторы:

DEFINT – для целочисленных переменных;

DEFSNG – для переменных с плавающей точкой одинарной точности;

DEFDBL – для переменных с плавающей точкой двойной точности;

DEFSTR – для строковых переменных.

В табл. 2.4. указано наличие операторов описания типа переменной в рассматриваемых версиях языка.

Таблица 2.4

Версия	DEFINT	DEFSNG	DEFDBL	DEFSTR
Стандарт	+	+	+	+
XYBASIC	+	+		+
Бейсик-TRS-80	+	+	+	+
MBASIC	+	+	+	+
Бейсик-ПК8010	+	+	+	+
Бейсик-ПК8020	+	+	+	+
MSX-BASIC	+	+	+	+
BASICA	+	+	+	+

Окончание табл. 2.4

Примечание. В версии XYBASIC операторы записываются с пробелом: DEF INT, DEF SNG, DEF STR.

Ключевое слово – это слово с определенной смысловой нагрузкой. Оно является инструкцией языка Бейсик и по этой инструкции интерпретатор выполняет определенные действия.

Программа на языке Бейсик – это упорядоченная последовательность строк, включающих в себя номер строки и набор инструкций, состоящих из ключевых слов, за которыми следуют параметры. Инструкции описывают операции, которые необходимо выполнить. Операция определяется ключевым словом.

Инструкции в языке Бейсик делятся на команды, операторы и функции. **Команды** определяют способ управления программой. Они управляют, исполняют или воздействуют на программу. **Операторы** разрешают доступ, ввод-вывод или обработку данных и определяют конечный результат. **Функции** устанавливают значения математических операций, операций обработки строки и операций, которые определяет сам пользователь. Функции являются частью оператора. **Ключевые слова** должны отделяться от других синтаксических единиц ограничителями или пробелами. Набор ключевых (резервированных) слов различен для разных версий языка Бейсик и приведен в приложении 2.

Простые ограничители – это любые специальные символы, кроме символов $ % ! и #.

Составные ограничители – образуются соединением двух специальных символов:

<> не равно; <= меньше или равно (не больше); >= больше или равно (не меньше)

Константы – используются для записи величин, которые не изменяются во время выполнения программы.

Комментарий – это набор символов, следующий после оператора REM или '. Может включаться в текст программы, предназначен для записи пояснений к алгоритму задачи, не влияет на выполнение программы.

2.4. Данные и их описание

Данные в языке Бейсик (рис. 2.3) подразделяются на два вида: **константы** и **переменные.** С каждым видом данных связано понятие типа.

Тип данных определяет внутреннее представление данных в памяти. Бейсик оперирует с данными следующих типов: строковыми и числовыми – шестнадцатеричными, восьмеричными, двоичными и десятичными с фиксированной и плавающей точкой одинарной и двойной точности.

Строковые данные – набор данных в виде записей, образующих строки для обработки текстов и вывода на печать. При работе со строковыми данными язык Бейсик позволяет:

присваивать строковое значение строковой переменной;

создавать строковые формы, которые образуют новые строки.

2.4.1. Константы

Константы – вещественные значения, не изменяемые во время выполнения программы. Имеется два типа констант: строковые (символьные) и числовые.

Для каждого типа констант имеются свои правила записи. Последовательность символов, образующих константу, определяет как значение, так и тип этой константы.

В табл. 2.5 показано наличие типов констант в разных версиях языка Бейсик.

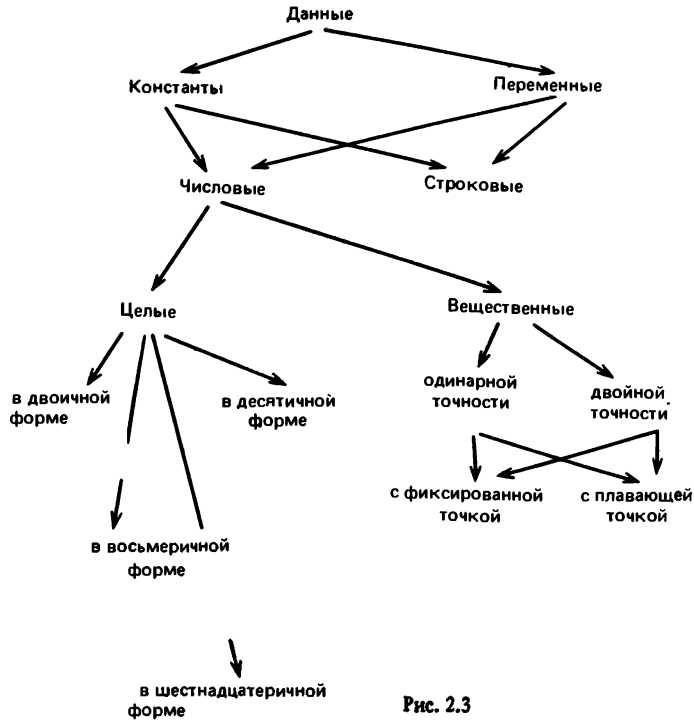


Рис. 2.3

Таблица 2.5

Тип переменных	Номер версии									
	1	2	3	4	5	6	7	8	9	10
Строковые	+	+	+	+	+	+	+	+	+	+
Десятичные целые	+	+	+	+	+	+	+	+	+	+
Десятичные с фиксированной точкой	+	+	+	+	+	+	+	+	+	+
Десятичные с плавающей точкой одинарной точности	+	+	+	+	+	+	+	+	+	+
Десятичные с плавающей точкой двойной точности					+	+	+	+	+	+

Окончание табл. 2.5

Тип переменных	Номер версии									
	1	2	3	4	5	6	7	8	9	10
Шестнадцатеричные	+	+	+	+	+	+	+	+	+	+
Восьмеричные					+	+	+	+	+	+
Двоичные		+	+				+	+	+	+

Строковая константа — это последовательность символов (до 255), заключенная в кавычки.

Кавычки не могут использоваться внутри строковой константы. В качестве строковых констант могут применяться любые символы кода ASCII (КОИ-8), имеющие графическое представление. В частности, можно использовать специальные символы ("+", "=", "?", "/" и др.) цифры, прописные и строчные буквы латинского и русского алфавитов. Например:

```
PRINT "ПРОГРАММА PROGRAMM программа programm"
```

Чтобы включать в строковые данные кавычки или символы, не имеющие графического представления, используется функция CHR$.

Числовая константа — положительное или отрицательное число. Числовые константы могут быть целыми, с фиксированной или плавающей точкой. Целые числовые константы могут быть записаны в десятичном, двоичном, восьмеричном и шестнадцатеричном форматах. Целые константы запоминаются в двух байтах памяти. Целое десятичное число — число от −32768 до +32767 включительно.

Например **−12 или 27**

Двоичное число — цифры в двоичной системе счисления. Двоичные константы имеют префикс (табл. 2.6), за которым следует шестнадцать цифр (0 и 1).

Восьмеричное число — цифры в восьмеричной системе счисления. Восьмеричные константы имеют префикс, за которым следует шесть восьмеричных цифр (0 — 7).

Шестнадцатеричное число — цифры в шестнадцатеричной системе счисления. Шестнадцатеричные константы имеют префикс (в Бейсик-Спектрум+2 — постфикс), за (перед) которым следуют четыре шестнадцатеричные цифры (0 — 9 и буквы A — F).

Константы в двоичном, восьмеричном и шестнадцатеричном форматах запоминаются в двух байтах и интерпретируются как целые числа. Константы имеют разное представление в рассматриваемых версиях языка Бейсик (табл. 2.6).

Таблица 2.6

Тип константы	Номер версии									
	1	2	3	4	5	6	7	8	9	10
Шестнадцатеричная	&H	#	h постф.	$	&H(*)	&H	&H	&H	&H	&H
Восьмеричная					&O(*)	&O	&O	&O или &	&O	&O
Двоичная		&	BIN				&B	&B	&B	&B

Примечание. Данное представление только для дисковых версий языка Бейсик.

Число с фиксированной точкой – положительное или отрицательное вещественное число, т. е. число, содержащее точку на постоянном месте. Например:

–31.94, + 78.02 или **541.77**

Число с плавающей точкой – положительное или отрицательное число, представленное в экспоненциальной форме. Константа с плавающей точкой состоит из знаковой фиксированной целой части (мантиссы), за которой следует буква E и (необязательно) знаковое целое число (экспонента). Константы с плавающей точкой в некоторых версиях языка Бейсик могут быть одинарной E и двойной D точности.

Константы одинарной точности могут быть записаны:

семью или меньше цифрами;

в экспоненциальной форме, используя букву E;

с завершающим знаком !.

Константы двойной точности могут быть записаны:

восемью или больше цифрами;

в экспоненциальной форме, используя букву D;

с завершающим знаком #.

Например:

1E13 или **3.5D–6**

Диапазоны изменения чисел с плавающей точкой для разных версий представлены в табл. 2.7.

Таблица 2.7

Версия	Числа одинарной точности	Числа двойной точности
XYBASIC	от -1.7E38 до 1.7E38	--
Бейсик-Спектрум+2	от 4E-39 до 1E38	--
Бейсик-АГАТ	от 1E-38 до 1E+38	--
Бейсик-TRS-80	от 1.7E-38 до 1.7E+38	от 1.7D-38 до 1.7D+38
MBASIC	от 0.9E-38 до 1.7E+38	от 0.2D-38 до 1.7D+38
Бейсик ПК8010	от 1.7E-38 до 1.7E+38	от 1.7D-38 до 1.7D+38
Бейсик ПК8020	от 1.7E-38 до 1.7E+38	от 1.7D-38 до 1.7D+38
MSX-BASIC	от 10E-64 до 10E+64	от 10D-64 до 10D+64
BASICA	от 1.7E-38 до 1.7E+38	от 1.7D-38 до 1.7D+38

2.4.2. Переменные

Переменная – это величина, к которой обращаются по имени и которая может принимать различные значения.

Имена переменных – это ячейки памяти, содержащие значения этих переменных. Имена могут определять значение переменной как константу или могут быть результатом вычислений в программе. Перед тем как переменной присвоено какое-либо значение, значение переменной равно нулю.

Тип переменной определяет ее внутреннее представление. В языке Бейсик существует два типа переменных: строковые и числовые.

Строковая переменная может занимать до 255 байт. Строковые значения присваиваются переменным так же, как и числовые. Специальный символ "$" идентифицирует строковую переменную. Если строковой переменной не было присвоено значение, то считается, что ей присвоена пустая строка. Новое значение строковой переменной может быть присвоено оператором LET.

Например:

```
10 A$="STRING"
20 PRINT A$
30 A$="строка"
40 PRINT A$
RUN
STRING
строка
```

В этом примере значения, присваиваемые переменной A$, заключаются в кавычки. Строковым переменным могут присваиваться и более сложные выражения. Если строковой переменной будет присвоено числовое значение или числовой переменной – строковое, то появится сообщение об ошибке.

Примеры строковых переменных:

```
STROKA$="Next line - следующая строка"
CHISLO$="40*1.723E+5"
NAME$="Александр Сергеевич Пушкин"
```

Для присвоения новых значений строковым переменным можно использовать также операторы READ, DATA, INPUT и т. п.

Числовая переменная может быть целочисленной и вещественной. Целочисленные переменные (2 байта) могут представляться в десятичном, двоичном, восьмеричном и шестнадцатеричном форматах. Вещественные переменные одинарной точности занимают 4 байта памяти, двойной точности – 8 байт.

В языке Бейсик имеются переменные, а также наборы переменных, образующие массивы.

Массив – это упорядоченная последовательность величин, обозначенных одним именем (или набор переменных), отличающихся индексом.

Отдельные величины, образующие массив, называются элементами массива.

Элементы массива определяются именем массива и индексом, заключенным в скобки, и образуют переменные с индексом (индексированные переменные). Индекс указывает на положение элемента в массиве. Элемент массива имеет столько индексов, какова размерность массива. При обращении к массиву индекс может задаваться любым выражением. Отрицательное значение индекса вызывает ошибку.

Язык Бейсик позволяет использовать массивы строковых переменных таким же образом, как и массивы числовых переменных. Например, DIM NAME$(50) определяет массив, размещающий 51 строковую переменную, которые имеют имена NAME$(0), NAME$(1), ..., NAME$(50).

В версии языка Бейсик-Спектрум+2 имеются следующие особенности:

массивы могут иметь несколько размерностей произвольной длины;

имя массива строковых переменных не может совпадать с именем строковой переменной.

Все строки в массиве имеют одинаковую фиксированную длину, которая задана как дополнительная окончательная размерность в операторе DIM. Индексы начинаются с 1.

В остальных рассматриваемых версиях простые переменные и массивы могут иметь одинаковые имена.

Во всех версиях, кроме Бейсик-Спектрум+2, минимальное значение индексов равно 0, а не 1. В некоторых версиях имеется оператор OPTION BASE, позволяющий изменять минимальное значение индексов.

Как под переменные, так и под массивы должна быть резервирована память, поэтому необходимо сообщить, какие массивы будут использованы в программе, каков тип переменных и размер каждого массива. Описание массива осуществляется оператором DIM и должно появляться в программе до первого обращения к элементу массива.

В версиях MBASIC, Бейсик-ПК8010, Бейсик-ПК8020, MSX-BASIC, BASICA, если описание массива отсутствует, то по умолчанию максимальное значение каждого индекса равно 10.

Если во время выполнения программы оказывается мало памяти, то в некоторых версиях используется оператор ERASE, удаляющий массивы из программы.

Размерность массивов и число элементов определяется количеством байтов свободной памяти. В версии языка Бейсик-АГАТ используется не более трех размерностей.

2.5. Перевод чисел одной точности в числа другой точности

Если числовое значение одной точности присваивается числовой переменной другой точности, то число будет запоминаться с точностью, описанной именем переменной. Например:

```
10 A%=23.42
20 PRINT A%
RUN
 23
```

При этом, если любое значение более высокой точности присваивается переменной меньшей точности, то осуществляется необходимое округление. Например:

```
10 C%=55.88
20 PRINT C%
RUN
 56
```

Если число с меньшей точностью переводится в число с большей точностью, то полученное в результате число не может быть более точным, чем число с меньшей точностью. Так, при переводе числа A одинарной точности в B# двойной точности только первые шесть цифр B# будут точными. Например:

```
10 A=2.04
20 B#=A
30 PRINT A;B#
RUN
 2.04   2.03999991853027
```

При вычислении выражения все операнды в арифметических операциях или операциях отношения переводятся в ту же точность, что и операнд с наибольшей точностью. Так, в нижеприведенном примере арифметическая операция выполняется с двойной точностью, и результат D# получается как число двойной точности. Например:

```
10 D#=6#/7
20 PRINT D#
RUN
 .8571428571428571
```

Если в результате выполнения арифметической операции, выполняемой с двойной точностью, должно быть получено число одинарной точности, то полученный результат округляется до числа одинарной точности. Например:

```
10 D=6#/7
20 PRINT D
RUN
 .857143
```

Логические операции переводят свои операнды в целые числа и образуют целочисленный результат. Операнды должны быть в диапазоне целых чисел, иначе появляется сообщение об ошибке Underflow (Антипереполнение).

2.6. Выражения, типы операций

Выражения представляют собой компактную запись, указывающую, какие операции надо производить над данными, чтобы получить требуемое значение. Порядок вычисления выражения определяется приоритетом используемых операций.

В языке Бейсик выражение состоит из операндов, соединенных знаком арифметических, логических операций или операций отношения и круглыми скобками. Операндами выражений являются любые переменные, числовые и строковые константы. Также в выражении могут присутствовать встроенные функции.

В языке Бейсик используются следующие типы операций:

выполнение встроенных функций;

арифметические;

отношения;

логические.

Последовательность выполнения операций в порядке их приоритета связана с особенностями различных Бейсик-систем. Можно выделить четыре основных правила приоритета для рассматриваемых версий:

1.(Бейсик-АГАТ, Бейсик-TRS—80)

Самые внутренние круглые скобки
Встроенные функции
Возведение в степень
Унарный плюс или минус
Умножение и деление
Сложение и вычитание
Операции отношения
Логические операции NOT, AND, OR

2.(MBASIC, Бейсик-ПК8010, Бейсик-ПК8020, MSX-BASIC, BASICA)

Самые внутренние круглые скобки
Арифметические функции
Относительные функции
Логические функции
Возведение в степень
Унарный плюс или минус
Умножение и деление
Целочисленное деление

	Образование арифметического модуля MOD
	Сложение и вычитание
	Операции отношения
	Логические операции NOT, AND, OR, XOR, IMP, EQV
3.(XYBASIC)	4.(Бейсик-Спектрум+2)
Самые внутренние круглые скобки	Индексирование и расслоение
Встроенные функции	Встроенные функции
Унарный плюс или минус	Внутренние скобки
Операция JOIN	Унарный плюс или минус
Возведение в степень	Возведение в степень
Умножение, деление, целочисленное деление и образование арифметического модуля MOD	Умножение и деление
Сложение и вычитание	Сложение и вычитание
Операции отношения	Операторы отношения в числовых выражениях
Логические операции NOT, AND, OR, XOR	Логические операции NOT, AND, OR

2.6.1. Арифметические операции

Таблица 2,8

Знак операции	Операция	Запись	Пример
-	Отрицание	-X	-3
^	Возведение в степень	X^Y	2^3=8
*	Умножение	X*Y	5.4*1.25=6.75
/	Деление с плавающей точкой	X/Y	4/0.2=20
\	Целочисленное деление	X\Y	10\4=2 25.686\.99=26
MOD	Образование арифметического модуля	X MOD Y	13 MOD 3.8=1 25.68 MOD 6.99=5
+	Сложение	X+Y	4+3.87=7.87
-	Вычитание	X-Y	4-3.87=0.13
JOIN	Конкатенация двух восьмибитовых чисел	X JOIN Y	1 JOIN 2 = 258 #FF JOIN #FF=-1

При выполнении операции целочисленного деления операнды округляются до целых чисел, которые должны лежать в диапазоне от –32768 до +32767. Частное от деления округляется до целого отбрасыванием дробной части.

Операция арифметического модуля вычисляет целое значение остатка целочисленного деления.

Оператор JOIN осуществляет конкатенацию двух 8-битовых чисел в 16-битовое число. Например, в результате конкатенации чисел 1 (двоичное 0000 0001) и 2 (двоичное 0000 0010) получится число 258 (двоичное 0000 0001 0000 0010).

Разные версии языка Бейсик могут отличаться набором выполняемых арифметических операций (табл. 2.9).

Таблица 2.9

Версия	Арифметическая операция								
	Унарный минус	^	*	/	\	MOD	+	-	JOIN
Стандарт	+	+	+	+	+	+	+	+	
XYBASIC	+	+	+	+	+	+	+	+	+
Бейсик-Спектрум+2	+	+	+	+			+	+	
Бейсик-АГАТ	+	+	+	+			+	+	
Бейсик-TRS-80	+	+	+	+			+	+	
MBASIC	+	+	+	+	+	+	+	+	
Бейсик-ПК8010	+	+	+	+	+	+	+	+	
Бейсик-ПК8020	+	+	+	+	+	+	+	+	
MSX-BASIC	+	+	+	+	+	+	+	+	
BASICA	+	+	+	+	+	+	+	+	

2.6.2. Операции отношения

Операции отношения используются для сравнения двух переменных. Результат сравнения ИСТИНА (–1) или ЛОЖЬ (0). В версии Бейсик-АГАТ ИСТИНА (1). В табл. 2.10 представлены операции отношения.

Таблица 2.10

Знак операции	Операция	Пример
=	Равенство	X=Y
<>	Неравенство	X<>Y
<	Меньше	X<Y
>	Больше	X>Y
<=	Меньше или равно (не больше)	X<=Y
>=	Больше или равно (не меньше)	X>=Y

2.6.3. Логические операции

Логические операции имеют дело с комбинациями значений ИСТИНА–ЛОЖЬ. В результате работы операций также получается или ИСТИНА, или ЛОЖЬ. Считается, что операнд логического оператора или результат операции есть ИСТИНА, если он не равен нулю.

Рассмотрим логические операции в порядке приоритета: NOT – логическое отрицание; AND – конъюнкция; OR – дизъюнкция; XOR – Исключающее ИЛИ; IMP – импликация; EQV – эквивалентность.

NOT – операция логического отрицания аргумента. Изменяет значение аргумента на противоположное.

X	NOT X
И	Л
Л	И

AND – операция логического умножения двух аргументов (конъюнкция, И). В результате выполнения операции получается значение ИСТИНА, если оба аргумента истинны, и ЛОЖЬ – во всех остальных случаях.

X	Y	X AND Y
И	И	И
И	Л	Л
Л	И	Л
Л	Л	Л

OR – операция логического сложения двух аргументов (дизъюнкция, ИЛИ). В результате выполнения операции получится значение ЛОЖЬ, если оба аргумента ложны; в остальных случаях результат принимает значение ИСТИНА.

X	Y	X OR Y
И	И	И
И	Л	И
Л	И	И
Л	Л	Л

XOR – операция логического Исключающего ИЛИ двух аргументов. Результатом выполнения операции является ЛОЖЬ, если оба аргумента имеют одинаковые значения, в противном случае результатом является значение ИСТИНА.

X	Y	X XOR Y
И	И	Л
И	Л	И
Л	И	И
Л	Л	Л

IMP – операция логической импликации двух аргументов. Результатом будет значение ЛОЖЬ, если первый операнд имеет значение ИСТИНА, а второй – ЛОЖЬ; во всех остальных случаях результатом будет значение ИСТИНА.

X	Y	X IMP Y
И	И	И
И	Л	Л
Л	И	И
Л	Л	И

EQV – операция логической эквивалентности двух аргументов. Результатом выполнения этой функции будет ИСТИНА, если оба аргумента истинны или ложны. В противном случае результатом будет ЛОЖЬ.

X	Y	X EQV Y
И	И	И
И	Л	Л
Л	И	Л
Л	Л	И

В табл. 2.11 представлены сведения о наличии логических операций в разных версиях языка Бейсик.

Таблица 2.11

Версия	NOT	AND	OR	XOR	IMP	EQV
Стандарт	+	+	+	+	+	+
XYBASIC	+	+	+	+		

Окончание табл. 2.11

Версия	NOT	AND	OR	XOR	IMP	EQV
Бейсик-Спектрум+2	+	+	+	+		
Бейсик-АГАТ	+	+	+			
Бейсик-TRS 80	+	+	+			
MBASIC	+	+	+	+	+	+
Бейсик-ПК8010	+	+	+	+	+	+
Бейсик-ПК8020	+	+	+	+	+	+
MSX-BASIC	+	+	+	+	+	+
BASIC	+	+	+	+	+	+

Логические операции выполняются путем преобразования их операндов в 16-битовые знаковые целые числа в дополнительном коде в допустимом для каждой версии диапазоне. Если операнд не лежит в этом диапазоне, то результат будет ошибочным. Логические операции выполняются с целыми числами поразрядно (побитно), т.е. каждый бит результата зависит от значения соответствующих битов двух операндов.

Логические операции могут применяться для программирования на уровне битов в пределах байта. Например, оператор AND может быть использован для "маскирования" всех битов, кроме одного, байта состояния машинного порта ввода-вывода. Оператор OR может быть использован для "соединения" двух байтов при создании двоичного значения. Следующие примеры демонстрируют, как работают логические операции.

```
63 AND 16 = 16
```

Так как числа 63 и 16 в двоичном коде представляются 111111 и 10000 соответственно, то после выполнения логической операции AND будет получено двоичное число 10000, что соответствует числу 16, т.е. 111111 AND 10000 = 10000.

```
15 AND 14 = 14
-1 AND 8 = 8
4 OR 2 = 6
10 OR 10 = 10
-1 OR -2 = -1
NOT X = -(X+1)
```

Часто логические операции используются в условном операторе IF. Например:

```
IF (A=1) OR (B=5) THEN GOTO 350
IF A=(B OR C) THEN GOSUB 800
IF ERR=13 AND ERL=400 THEN PRINT : GOTO 5000
```

2.7. Строковые операции

Строковые операции могут быть следующими: конкатенации; деления строки на части; сравнения; выполнение встроенных функций.

Операция конкатенации (сцепления) обозначается символом плюс (+). В результате этой операции образуется строка, левая часть которой равна первому операнду, а правая часть — второму. Например:

```
10 CO$="Пример "
20 TP$="задачи "
30 ST$=TP$+"на языке BASIC"
40 PRINT CO$+ST$
RUN
Пример задачи на языке BASIC
Ok
```

Операция деления строки на части присутствует только в версии языка Бейсик-Спектрум+2. В остальных версиях языка Бейсик этой цели служат встроенные функции, описанные в гл. 3.

В версии Бейсик-Спектрум+2 для заданной строки ее подстрока состоит из нескольких символов строки, следующих подряд. Следовательно, "string" — это подстрока строки "bigger string", а строки "b string" и "big ger" не являются подстроками этой строки.

Для описания подстрок имеется понятие, называемое делением на части, и оно может быть применено к произвольным строковым выражениям.

Общий формат:

строковое выражение (начало TO конец)

начало и конец — целочисленные выражения.

По умолчанию начало равно 1, а конец — длине строки. Например:

```
"abcdef"(2 TO 5) равно "bcde"
"abcdef"(TO 5) равно "abcde"
"abcdef"(2 TO) равно "bcdef"
"abcdef"(TO) равно "abcdef"
```

Последнее выражение можно также записать в виде "abcdef"().

В другом формате деления на части опускается ключевое слово TO и содержится только одно число.

```
"abcdef"(3) равно "abcdef"(3 TO 3) и равно "c".
```

Если значение начала больше значения конца, то результатом является пустая строка. Таким образом, "abcdef"(5 TO 7) приводит к ошибке индекса, поскольку строка содержит только 6 символов, однако выражения "abcdef"(8 TO 7) и "abcdef"(1 TO 0) оба равны пустой строке и поэтому допустимы.

Начало и конец не должны быть отрицательными, в противном случае появится сообщение об ошибке "B integer out of range" (В число вне диапазона).

Можно не только извлекать из строковых переменных подстроки, но также присваивать значения подстрокам. Например:

```
10 LET a$="I love my Sinclair"
20 LET a$(11 TO 18)="Amstrad*****"
30 PRINT a$
RUN
I love my Amstrad*
```

Заметим, что поскольку подстрока a$(11 TO 18) имеет длину, равную только восьми символам, то только первые восемь символов (Amstrad) используются в качестве ее значения; остальные четыре символа (****) отбрасываются.*

Подстрока должна иметь такую же длину после присваивания значения, какую она имела до присваивания, т.е., если присваиваемое значение больше длины строки, которой присваивается значение, то лишние символы справа отсекаются, а если присваиваемое значение по длине короче длины строки, которой присваивается значение, то строка дополняется пробелами.

Сложные строковые выражения должны быть заключены в скобки перед делением на части. Например:

```
"abc"+"def"(1 TO 2)    равно "abcde"
("abc"+"def")(1 TO 2) равно "ab"
```

Строки можно сравнивать, используя операции отношения так же, как и числа. Строковое сравнение выполняется посимвольным сравнением строк по кодам ASCII (КОИ-8). Если все коды двух строк совпадают, то строки равны. Если коды различны, то строка с меньшим кодом меньше. Если во время сравнения конец одной строки встретился раньше, то эта строка меньше. Например:

```
"AA"<"AB"
"Filename"="Filename"
"X&">"X#"
"C ">"C"
"kg">"KG"
"B$<"9/10/89"   где B$="8/10/89"
```

Все строковые константы, используемые в выражениях, должны быть заключены в кавычки.

2.8. Описание инструкций

Таблица 2.12

Инструкция	Номер версии									
	1	2	3	4	5	6	7	8	9	10
DEFDBL	+				+	+	+	+	+	+
DEFINT	+				+	+	+	+	+	+

Окончание табл. 2.12

Инструкция	Номер версии									
	1	2	3	4	5	6	7	8	9	10
DEFSNG	+				+	+	+	+	+	+
DEFSTR	+				+	+	+	+	+	+
DIM	+	+	+	+	+	+	+	+	+	+
ERASE			+			+	+	+	+	+
OPTION BASE	+					+				+
REM	+	+	+	+	+	+	+	+	+	+

DEFINT, DEFSNG, DEFDBL, DEFSTR – операторы, описывающие тип переменных: целочисленные (INT), одинарной точности (SNG), двойной точности (DBL), строковые (STR)

```
DEFтип буква[-буква],[,буква[-буква]]...
```

тип – INT, SNG, DBL, STR

Оператор DEF тип определяет, что переменные, имена которых начинаются с указанных букв, будут переменными заданного типа. Данные операторы, если они используются в программе, задаются в начале программы, перед использованием переменной. Например:

```
10 DEFINT X,D-H : DEFDBL L-P : DEFSTR A
```

В XYBASIC существуют только операторы DEF INT, DEF SNG, DEF STR, причем пробел между ключевым словом DEF и типом обязателен.

DIM – определяет максимальную размерность массива, резервирует память для него

Формат 1 (кроме Бейсик-Спектрум+2).

```
DIM имя массива(список индексов) [,имя массива(список
индексов)]...
```

Если имя элемента массива используется без предварительного объявления оператором DIM, то максимальное значение его индекса равно 10. Если используется индекс, который больше заданного максимального значения, то появляется сообщение об ошибке Subscript out of range (Индекс вне диапазона).

Минимальное значение индекса по умолчанию равно 0. При необходимости оно может быть установлено равным 1. Для этого используется оператор OPTION BASE. Например:

```
DIM X(2,3,4),Y$(15)
```

Приведенный в примере оператор DIM определяет X как трехмерный массив с тремя элементами по первому измерению, четырьмя элементами по второму и пятью элементами по третьему; а также определяет Y$ как строковый массив из 16 элементов.

Формат 2 (только для Бейсик-Спектрум+2).

```
DIM имя массива(индексы)
```

Оператор DIM устанавливает все элементы числового массива равными нулю, а все элементы строкового массива – пустыми.

ERASE – оператор очистки области памяти, отведенной под массивы

```
ERASE имя массива[,имя массива]...
```

Исключает массивы из программы. Если во время выполнения программы оказывается мало памяти, можно использовать оператор ERASE. После того, как массивы удалены, пространство памяти, зарезервированное под массивы, можно использовать для других целей. Оператор ERASE может быть использован также для переопределения размерности массива, так как при попытке заново определить уже определенный массив возникает сообщение об ошибке Duplicate definition of array (Двойное определение массива).

OPTION BASE – оператор, устанавливающий минимальное значение для индексов массива

OPTION BASE n

n – 0 или 1.

Если n=1, то минимальное значение индекса массива равно 1, по умолчанию n=0. Оператор должен выполняться перед описанием или перед первым обращением к какому-либо массиву.

REM – служит для внесения в текст программы комментариев

REM комментарий

Оператор REM является неисполнительным оператором. После ключевого слова (remark – замечание) может следовать любой латинский или русский текст, а также символы псевдографики. В некоторых версиях языка, например MSX-BASIC, BASICA, Бейсик-ПК8020, Бейсик-ПК8010, MBASIC, ключевое слово REM может быть заменено апострофом (символ ').

Некоторые версии интерпретаторов автоматически заносят на место апострофа REM с двоеточием (:) впереди, например в версиях Бейсик-ПК8020 и Бейсик-ПК8010. В версии MSX-BASIC нет такой замены и в тексте программы стоят апострофы.

Глава 3

Основы программирования на языке Бейсик

3.1. Общие рекомендации

При программировании на языке Бейсик необходимо учитывать три основные фактора, характерные, впрочем, для всех языков программирования: понимание программы, емкость используемой памяти, время выполнения программы.

В общем случае нельзя написать программу, в которой все эти три фактора оптимизированы, так как понимание программы находится в противоречии с двумя другими факторами. Емкость используемой памяти и время выполнения программы также находятся в противоречии друг с другом. Поэтому основные рекомендации при написании программ на языке Бейсик следующие: при отладке программы широко использовать комментарии, каждый оператор писать на отдельной строке, разделять пробелами ключевые слова, знаки операций и операнды. При отладке программы желательно использовать операторы трассировки и отладочную печать.

Когда программа отлажена, необходимо провести ее анализ и установить, какой из факторов необходимо оптимизировать: память или время выполнения. Возможно и последовательное улучшение обоих факторов, что приводит к итерационному процессу оптимизации программы. Если программа многоразового использования, то проведение итерационного

процесса обязательно. При написании программы для решения одноразовой задачи оптимизация не обязательна, если время выполнения программы удовлетворяет программиста и программа помещается в памяти ПЭВМ.

В любом случае желательно сохранять последний отладочный вариант программы. Он служит для дальнейшей работы с программой, являясь исходным текстом программы.

Рекомендации по повышению скорости работы и одновременно уменьшению объема программы:

убирать все ненужные операторы REM, оставляя лишь самые необходимые;

не использовать лишние пробелы между операторами и операциями;

использовать, если это возможно, несколько операторов в каждой программной строке, так как каждый раз, когда вы вводите новый номер строки и заканчиваете ее нажатием клавиши возврата каретки, объем программы увеличивается на 5 байт (2 байта – адрес программной строки, 2 байта – ее номер и 1 байт – возврат каретки);

при присваивании опускать ключевое слово LET;

использовать целочисленные переменные, например в операторах FOR, ON, DIM, TAB, так как целые числа занимают 2 байта, а числа одинарной точности – 4 байта, кроме этого затрачивается время на округление чисел;

если подпрограмма всегда вызывается из одного места в программе, то следует использовать оператор безусловного перехода GOTO, а не вызов подпрограммы, так как активный оператор GOSUB занимает 6 байт памяти, а оператор GOTO памяти не требует;

при программировании выражений стараться меньше использовать скобки, так как операции внутри скобок выполняются первыми и результат необходимо запоминать, а это занимает память;

экономить размер массива, так как при определении массива резервируется память для всех элементов массива, даже если массив весь не заполнен.

Рекомендации по уменьшению объема программы:

задавать сначала наиболее используемые переменные, когда переменная определена, она размещается в верхней части таблицы переменных; следующая переменная будет расположена ниже. При обращении к таблице переменных сокращается время поиска часто используемой переменной;

использовать подпрограммы на внешних носителях, организуя их вызов в память на место уже отработавшей подпрограммы, например с помощью оператора CHAIN.

Рекомендации по повышению быстродействия программы:

при работе с переменными, например строковыми или целочисленными, использовать операторы DEFSTR или DEFINT, так как они ускоряют обработку переменных и нет необходимости использовать специальные символы описания переменных;

если возможно, использовать переменные, а не константы, так как доступ к переменным занимает меньше времени, чем перевод констант в число одинарной точности;

использовать подпрограммы на машинном языке, которые особенно эффективны в наиболее часто исполняемых частях программы;

выносить из циклов, особенно внутренних, все вычисления и инвариантные к условиям повторения фрагменты программы;

использовать эффективные вычислительные алгоритмы.

3.2. Ввод-вывод данных

Ввод данных в программу на языке Бейсик осуществляется несколькими способами (рис. 3.1.):

1. Данные могут быть заданы в самой программе или вычислены в ней с помощью оператора присваивания LET.

2. Данные могут быть считаны с помощью оператора READ из таблицы данных, созданной оператором DATA.

3. Данные могут быть введены в программу пользователя с клавиатуры во время выполнения программы с помощью операторов UNPUT и LINE INPUT.

4. Символ может быть введен в программу пользователя с клавиатуры во время выполнения программы с помощью функций INKEY¤, INPUT¤.

5. Данные могут быть введены в программу из файла данных с помощью операторов INPUT#, LINE INPUT# и функции INPUT¤.

Первый способ. С помощью оператора LET присваивается значение выражения переменной, заданной в левой части. Так как слово LET необязательно, то для лучшего понимания текста программы слово LET не указывают. Если задается

```
LET X=14 или X=14 ,
```

то значение переменной X будет равным 14. Переменные, заданные слева от знака равенства в операторе присваивания, могут принимать любые значения. Выражение, стоящее справа от знака равенства в операторе присваивания, может быть любым числом, любой переменной или формулой (выражением), состоящей из чисел, переменных и знаков операций. Рассмотрим несколько примеров присваивания различных значений переменным.

```
100 I%=4
110 I%=I%+1
120 D$="ПРИМЕР"
130 D$=D$+D$
140 X=3
150 Y=5
160 Z=(X+Y)*X+2*(Y-X)
```

В программной строке 100 число 4 является значением переменной I%.

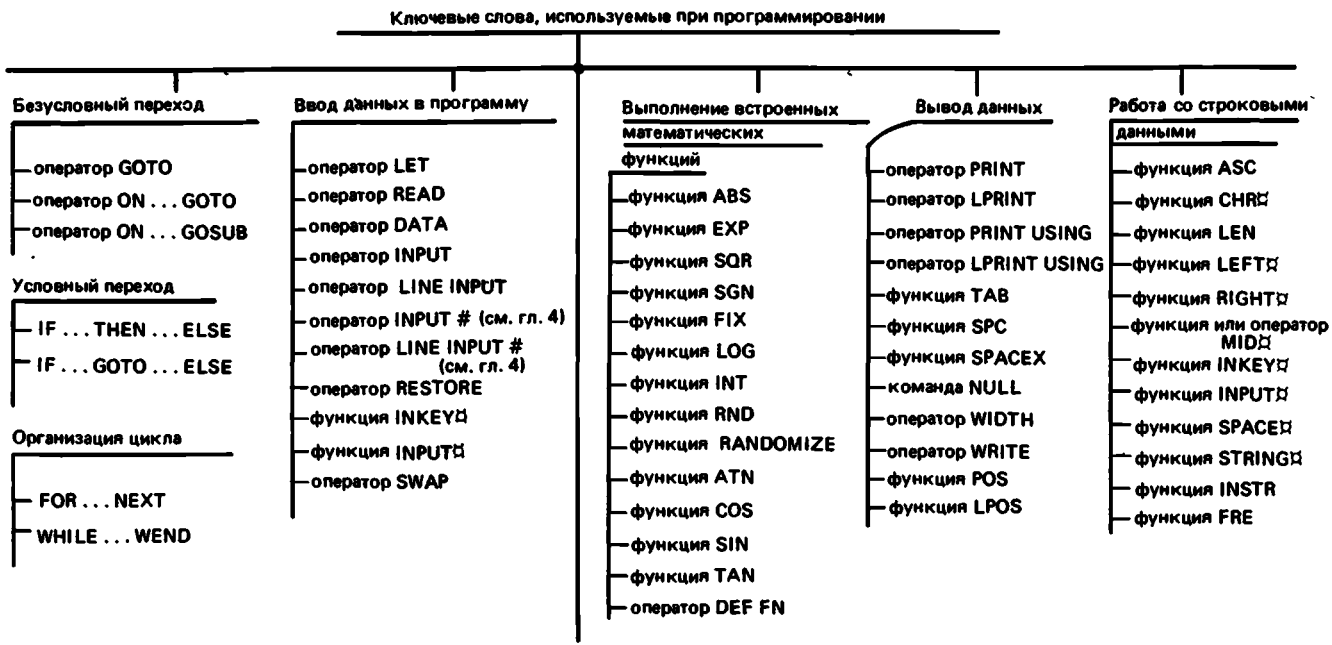
В строке 110 знак + является знаком операции сложения в обычном математическом понимании. В этом примере к значению переменной I%, равному 4, прибавляется 1 и полученное значение, равное 5, присваивается переменной I%.

В строке 120 строка "ПРИМЕР" является значением строковой переменной D$.

В строке 130 знак + является знаком операции конкатенации (сцепления) строк, которая создает строку, левая часть которой — первый операнд, а правая часть — второй операнд. Строка "ПРИМЕР" будет сцеплена со строкой "ПРИМЕР", и в результате будет получена новая строка "ПРИМЕРПРИМЕР". Таким образом, после выполнения программной строки 130 новым значением строки D$ будет "ПРИМЕРПРИМЕР".

В строках 140 и 150 переменной X присваивается значение, равное 3, а переменной Y — равное 5. В строке 160 справа от знака равенства стоит формула. Значения X=3 и Y=5 подставляются в эту формулу, производятся вычисления с учетом скобок и приоритета операций. Полученный результат, равный 28, присваивается переменной Z. Таким образом, после выполнения строки 160 число 28 является значением переменной Z.

Второй способ. Данные в программу вводятся с помощью операторов READ и DATA. Оператор DATA позволяет вводить таблицы данных в программу, доступ к которым осуществляется оператором READ.


Ключевые слова, используемые при программировании
Безусловный переход
оператор GOTO
оператор ON . . . GOTO
оператор ON . . . GOSUB
Условный переход
IF . . . THEN . . . ELSE
IF . . . GOTO . . . ELSE
Организация цикла
FOR . . . NEXT
WHILE . . . WEND
Ввод данных в программу
оператор LET
оператор READ
оператор DATA
оператор INPUT
оператор LINE INPUT
оператор INPUT # (см. гл. 4)
оператор LINE INPUT # (см. гл. 4)
оператор RESTORE
функция INKEY¤
функция INPUT¤
оператор SWAP
Выполнение встроенных математических функций
функция ABS
функция EXP
функция SQR
функция SGN
функция FIX
функция LOG
функция INT
функция RND
функция RANDOMIZE
функция ATN
функция COS
функция SIN
функция TAN
оператор DEF FN
Вывод данных
оператор PRINT
оператор LPRINT
оператор PRINT USING
оператор LPRINT USING
функция TAB
функция SPC
функция SPACEX
команда NULL
оператор WIDTH
оператор WRITE
функция POS
функция LPOS
Работа со строковыми данными
функция ASC
функция CHR¤
функция LEN
функция LEFT¤
функция RIGHT¤
функция или оператор MID¤
функция INKEY¤
функция INPUT¤
функция SPACE¤
функция STRING¤
функция INSTR
функция FRE

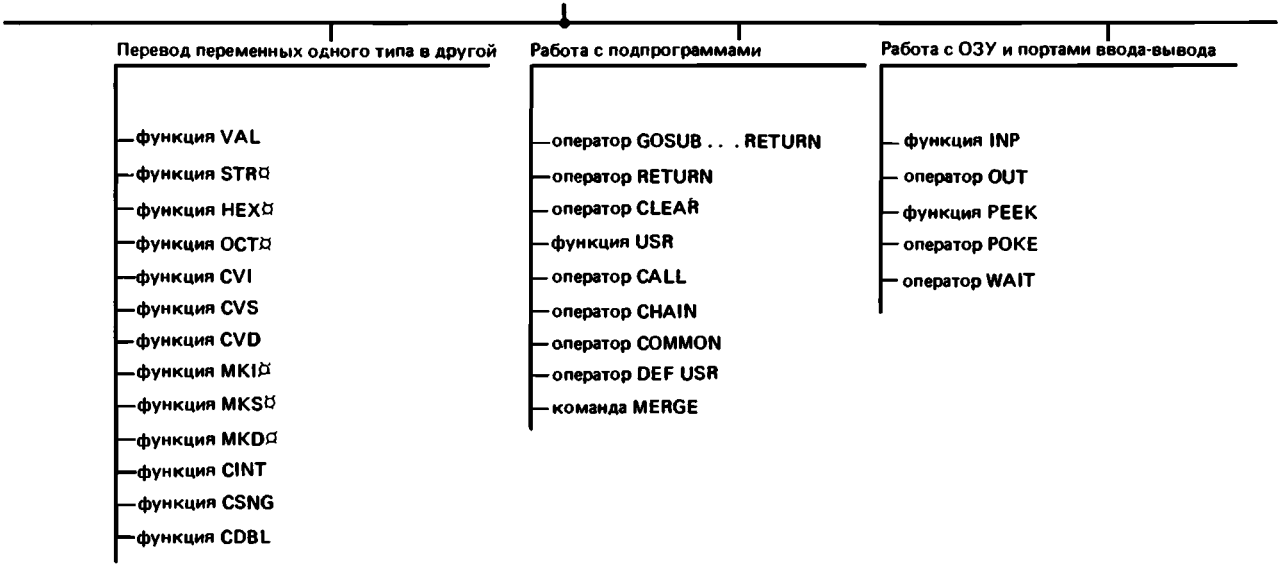

Перевод переменных одного типа в другой
функция VAL
функция STR¤
функция HEX¤
функция OCT¤
функция CVI
функция CVS
функция CVD
функция MKI¤
функция MKS¤
функция MKD¤
функция CINT
функция CSNG
функция CDBL
Работа с подпрограммами
оператор GOSUB . . . RETURN
оператор RETURN
оператор CLEAR
функция USR
оператор CALL
оператор CHAIN
оператор COMMON
оператор DEF USR
команда MERGE
Работа с ОЗУ и портами ввода-вывода
функция INP
оператор OUT
функция PEEK
оператор POKE
оператор WAIT

Рис. 3.1

При выполнении оператора READ считывается значение из оператора DATA и присваивается переменной, заданной в операторе READ. Рассмотрим небольшую программу, данные в которую вводятся с помощью операторов READ и DATA:

```
10 READ X
20 PRINT X;
30 GOTO 10
40 DATA 1,2,3,4
```

Выполнение этой программы происходит следующим образом:

выполняется программная строка 10. Указатель оператора READ стоит на первом элементе таблицы данных, созданной оператором DATA. Считывается первый элемент, который присваивается переменной X. Указатель оператора READ пересылается на следующий (второй) элемент таблицы данных. Таким образом, после выполнения строки 10 значение переменной X равно 1;

выполняется программная строка 20. Значение переменной X, равное 1, выводится на экран. Точка с запятой после X указывает на то, что следующее число будет выведено на экран в ту же строку в следующую позицию с учетом того, что перед и после числа выводится пробел;

выполняется программная строка 30. Осуществляется переход на строку 10;

выполняется программная строка 10. Из таблицы данных считывается второй элемент, который присваивается переменной X. Указатель оператора READ пересылается на третий элемент таблицы. Таким образом, после выполнения строки 10, значение переменной X равно 2.

Далее последовательно выполняются строки 20, 30, 10, 20, 30, 10, 20, 30, а на экране будут появляться числа 1 2 3 4.

При следующем выполнении строки 10 выдается сообщение об ошибке Out of data in 10 (Вне данных в 10), так как из таблицы данных считаны все элементы.

Чтобы избежать этой ошибки, необходимо либо программно проследить за соответствием количества элементов в таблице данных числу обращений к оператору READ, либо в программу вставить оператор RESTORE, который переопределяет указатель оператора READ на первый элемент таблицы данных, созданной оператором DATA. А если в операторе RESTORE указан номер строки, то указатель оператора READ устанавливается на первый элемент оператора DATA, стоящего в этой строке.

Чтобы считать данные из таблицы элементов, созданной оператором DATA, необходимо сделать следующее дополнение к программе:

```
25 IF X=4 THEN RESTORE
RUN
 1  2  3  4  1  2  3  4  1  2  3  4  1  2  3  4
<CTRL-C>
ВЫХОД В 25
Ok
```

Для остановки выполнения программы нажимаются клавиши CTRL—C. Если программу дополнить программными строками

```
25 IF X=8 THEN RESTORE 50
50 DATA 5,6,7,8
```

то при выполнении программы на экран дисплея будут выводиться следующие числа:

```
1  2  3  4  5  6  7  8  5  6  7  8  5  6  7  8  5
6 <CTRL-C>
```

Удобный способ определения последнего элемента в операторе DATA заключается в следующем: поместить в оператор последним числом фиктивное число, намного превосходящее числа, которые обрабатываются в программе. Фиктивное число должно выбираться очень тщательно, чтобы его нельзя было спутать с обрабатываемыми данными. Рассмотрим следующую программу:

```
10 DATA 8,4,5,7,-3,-8,4,1000
20 S=0
30 K=0
40 READ X
50 IF X=1000 THEN GOTO 90
60 K=K+1
70 S=S+X
80 GOTO 40
90 PRINT S,S/K,K
100 END
```

В этой программе необходимо найти сумму и среднее арифметическое значение чисел 8, 4, 5, 7, −3, −8 и 4. Число 1000 — фиктивное. Когда оно будет прочитано, программа напечатает значение S (сумма), S/K (среднее арифметическое значение) и K (количество чисел), где K — счетчик чисел, которые должны быть просуммированы. Значение счетчика меняется только после того, как будет определено, что прочитано нефиктивное число, т. е. меньше 1000. Фиктивное число может быть расположено в отдельном операторе DATA, чтобы можно было изменять только обрабатываемые значения.

Оператор READ может одновременно читать несколько чисел. Из оператора DATA числа читаются в том же порядке, в каком выполняются в программе операторы READ.

С помощью оператора READ из таблицы данных, созданной оператором DATA, можно считывать как числовые, так и строковые данные, при этом необходимо учитывать, что тип переменной, заданной в операторе READ, должен соответствовать типу константы в операторе DATA, иначе появится сообщение об ошибке Type mismatch (Неверный тип).

Оператор DATA можно располагать в любой части программы. Его можно разделить на части. Рекомендуется не разбрасывать операторы DATA по программе. Лучше всего поместить их все вместе в начале или недалеко от конца программы, но до выполнения оператора END.

Третий способ. Часто бывает необходимо вводить элементы данных в программу во время выполнения. Для этой цели используются операторы ввода INPUT и LINE INPUT. Когда в программе встречается оператор INPUT или LINE INPUT, выполнение программы приостанавливается, на экране появляется знак вопроса и программа ожидает ввода данных. Оператор INPUT позволяет выводить на экран подсказку для пользователя. Вводимые данные присваиваются переменным, заданным в операторе INPUT, и могут быть строковыми и числовыми. При несоответствии типа вводимых данных и переменных в операторе INPUT на экран выводится сообщение ?Redo from start (?Повторите ввод), и программа ожидает ввода данных.

Может быть так, что какие-либо данные в программе будут меняться, например размерность массива, начальное или конечное значение счетчика, в этих случаях очень удобно применять оператор INPUT.

Следующая программа иллюстрирует этот случай.

```
10 INPUT "Размерность массива = ";N
20 DIM NAME$(N),MAS(N)
30 FOR I=1 TO N
```

```
40 PRINT I,
50 INPUT "Фамилия ";NAME$(I)
60 INPUT "Оклад ";MAS(I)
70 NEXT
80 PRINT
90 FOR I=1 TO N
100 IF MAS(I) > 100 THEN PRINT I,NAME$(I),MAS(I)
110 NEXT
RUN
Размерность массива =? 5
 1 Фамилия ? Иванов
Оклад ? 150
 2 Фамилия ? Петров
Оклад ? 80
 3 Фамилия ? Антонов
Оклад ? 175
 4 Фамилия ? Смирнов
Оклад ? 105
 5 Фамилия ? Кузнецов
Оклад ? 95

 1 Иванов 150
 2 Антонов 175
 3 Смирнов 105
Ok
```

Если в программе необходимо вводить строки, то рекомендуется использовать оператор LINE INPUT. Этот оператор присваивает строковой переменной строку текста любого содержания длиной до 255 символов. В отличие от INPUT оператор LINE INPUT не печатает знак вопроса, если он не является частью строки подсказки.

Четвертый способ. Функция INPUT$ позволяет вводить определенное пользователем количество символов с клавиатуры или из файла данных. Если вводится больше символов, чем было определено при задании функции, то лишние данные игнорируются, а если меньше, то недостающими символами являются пробелы.

Функция INKEY$ также позволяет влиять на выполнение программы с клавиатуры. Эта функция вводит либо символ, считанный с клавиатуры, либо пустой символ, если не была нажата никакая клавиша. Введенный символ не отображается на экране дисплея. Перед использованием функции INKEY$ необходимо присвоить ей значение строковой переменной. Следующие программные строки демонстрируют работу функции INKEY$:

```
:
:
100 PRINT "Нажмите клавишу <возврат каретки>"
110 A$=INKEY$
120 IF A$=CHR$(&H0D) THEN GOTO 140
130 GOTO 110
:
:
```

В данном примере функция INKEY$ позволяет продолжить, например, вывод на экран информации после нажатия клавиши возврата каретки.

Удобно пользоваться функцией INKEY$ при чтении с экрана каких-либо директив, например, в обучающих прикладных программах, в программных меню. Следующий пример демонстрирует эту возможность.

```
.
:
100 PRINT 1;"СОЗДАТЬ ФАЙЛ"
110 PRINT 2;"СТЕРЕТЬ ФАЙЛ"
120 PRINT 3;"ПЕРЕИМЕНОВАТЬ ФАЙЛ"
130 PRINT 4;"ОТРЕДАКТИРОВАТЬ ФАЙЛ"
140 PRINT 5;"КОНЕЦ РАБОТЫ"
150 SM$=INKEY$
160 IF SM$="" THEN GOTO 150
170 SM=VAL(SM$)
180 ON SM GOSUB 220,300,400,500,600
190 PRINT "Вы ввели неправильный номер задания"
200 PRINT "Попробуйте еще раз"
210 GOTO 100
220 REM "1 Создать файл"
:
300 REM "2 Стереть файл"
:
400 REM "3 Переименовать файл"
:
500 REM "4 Отредактировать файл"
:
600 REM "5 Конец работы"
610 PRINT "Конец работы"
620 END
```

Пятый способ. В программу данные могут вводиться из файла данных, с помощью операторов INPUT#, LINE INPUT# и функции INPUT$. Предположим, что ранее был создан файл с последовательным доступом FILE.DAT. Следующая программа считывает данные из этого файла, присваивает переменной X и сравнивает значение переменной с нулем. Если значение X меньше нуля, то оно выводится на печатающее устройство.

```
10 OPEN "I",#1,"FILE.DAT"
20 IF EOF(1) THEN END
30 INPUT #1,X
40 IF X<0 THEN LPRINT X ELSE GOTO 20
```

Если из файла данных требуется считывать длинный текст, то рекомендуется использовать оператор LINE INPUT#.

Вывод данных – в языке Бейсик можно выводить на экран дисплея или на печатающее устройство числа и текст.

Для вывода на экран используется оператор PRINT, а для вывода на печатающее устройство – оператор LPRINT. В дальнейшем будем рассматривать только оператор PRINT, так как оператор LPRINT обладает аналогичными возможностями. Оператор PRINT можно использовать для:

вывода на экран текущего значения переменных и констант

```
PRINT X,Y,Z
PRINT 1;-2,3.6
PRINT X,0,F$
```

выполнения вычислений и вывода на экран их результатов

```
PRINT I;EXP(I),SQR(I),I^2+I^4
PRINT 1/SIN(X+Y*Z)
```

вывода на экран текстовых сообщений

```
PRINT "ПРОГРАММА РЕШЕНИЯ КВАДРАТНОГО УРАВНЕНИЯ"
PRINT "КВАДРАТ ЧИСЛА";Y;"РАВЕН";Y*Y
```

Оператор PRINT без параметров выводит на экран пустую строку.

Оператор PRINT, обладая большой гибкостью, позволяет получить различные форматы вывода на экран. Ниже описывается использование оператора PRINT в версии MBASIC; использование операторов PRINT в других версиях представлено в 3.5.

Позиция каждого выведенного на экран элемента определяется знаком препинания в списке выражений оператора PRINT. Экранная строка условно делится на зоны вывода по 14 символов каждая. Управление использованием зон производится с помощью запятых в списке выражений. Запятая указывает на то, что следующее значение будет выводиться с начала следующей зоны, а если все зоны вывода уже заполнены, то с начала первой зоны следующей строки.

Точка с запятой в списке выражений указывает на то, что следующее значение будет выводиться со следующей позиции с учетом пробелов при выводе чисел. Так, за числом всегда следует пробел. Положительному числу предшествует пробел, а отрицательному числу — знак минус. Строки, заключенные в кавычки, печатаются без изменений. Поэтому следующие операторы дадут различные результаты. Например:

```
10 PRINT 1;2
20 PRINT "1";"2"
30 PRINT 10,20
40 PRINT "10","20"
RUN
 1  2
12
 10            20
10           20
```

Если запятая или точка с запятой заканчивают список выражений, то следующий оператор PRINT (или INPUT) продолжит вывод в той же строке через соответствующее число пробелов, в противном случае вывод продолжается со следующей строки. Например:

```
10 PRINT "A"
20 GOTO 10
RUN
A
A
A
A
:
:
```

При изменении программной строки 10 на строку

```
10 PRINT "A",
```

получается

```
RUN
A         A         A         A         A         A
A         A         A         A         A         A
A         A         A         A         A         A
:
:
```

А если изменить программную строку 10 на строку

```
10 PRINT "A";
```

получается

```
RUN
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...
```

Оператор PRINT без параметров можно использовать для включения пустой строки, если необходимо разделить полученные результаты, или для перехода на новую строку. Например:

```
10 FOR I=1 TO N
20 FOR J=1 TO N
30 PRINT X(I,J);
40 NEXT J
50 PRINT
60 NEXT I
```

Программная строка 50 необходима для того, чтобы после печати значения X(1,1) следующее значение X(2,1) начало печататься в следующей строке и т. д.

Для форматирования выходной строки в языке Бейсик существуют следующие средства:

функция TAB позволяет выводить информацию с заданной позиции строки дисплея;

функция SPC выводит заданное число пробелов в строке;

функция SPACE$ образует строку, состоящую из заданного числа пробелов;

команда NULL определяет число пустых символов, выводимых после возврата каретки, т. е. задает левую границу экрана;

оператор WIDTH устанавливает ширину строки дисплея, т. е. задает правую границу экрана;

оператор PRINT USING позволяет выводить строковые и числовые данные, используя формат, заданный в операторе. Оператор PRINT USING определяет правую границу выравнивания, печатает знак + или − в начале или конце числа, печатает числа в экспоненциальной форме и т. п.

Иногда в программе может потребоваться очистить экран перед выводом определенной информации, для этой цели используется либо команда CLS, которая будет рассмотрена в гл. 6, либо оператор PRINT, заданный следующим образом:

```
PRINT CHR$(12)
```

где 12 (или &H0C) — код очистки экрана. В разных ПЭВМ этот код может изменяться.

Для вывода данных на экран дисплея можно использовать и оператор WRITE. Различие между операторами PRINT и WRITE в том, что оператор WRITE вставляет запятые между элементами, заключает строки в кавычки и положительным числам не предшествует пробел.

3.3. Изменение последовательности выполнения программы

Безусловный переход – принудительное, без проверки каких-либо условий, изменение последовательности выполнения программы.

Выполнение программы начинается с программной строки с наименьшим номером и следует далее в порядке увеличения номеров строк. Часто бывает необходимо прервать последовательное выполнение. Для этой цели используется оператор безусловного перехода GOTO, передающий управление программной строке, номер которой указан за ним. Например, если задан GOTO 100, то следующей будет выполняться программная строка с номером 100.

Оператор вычисляемого перехода ON ... GOTO позволяет использовать значение выражения для выбора номера строки программы из списка значений. Например, оператор ON VAR GOTO 100, 200, 300, 400 передает управление программной строке 100, если значение VAR равно 1, строке 200, если VAR равно 2 и т. д. Если значение переменной VAR – не целое число, то оно округляется. Если значение переменной равно 0 или больше количества номеров строк в списке, то будет выполняться следующая программная строка. Если значение переменной отрицательное или больше 255, то выдается сообщение об ошибке. Программная строка 10 ON VAR GOTO 100, 200, 300, 400 эквивалентна следующим программным строкам:

```
10 IF VAR=1 THEN GOTO 100
20 IF VAR=2 THEN GOTO 200
30 IF VAR=3 THEN GOTO 300
40 IF VAR=4 THEN GOTO 400
```

Подпрограмма – часть программы, допускающая многократное обращение из разных точек программы.

Подпрограммы, используемые в программах, написанных на языке Бейсик, могут быть написаны на языке Бейсик или на языке ассемблера (машинном языке).

В языке Бейсик можно организовать программу так, что различные программные сегменты будут выступать в роли подпрограмм. Для обращения к таким подпрограммам используется оператор вызова подпрограммы GOSUB. Оператор возврата из подпрограммы RETURN возвращает управление из подпрограммы обратно в вызывающую программу, после чего программа продолжает выполняться с оператора, следующего за оператором GOSUB. Подпрограмма может содержать несколько операторов RETURN для возврата из разных точек подпрограммы. В программе может быть любое количество подпрограмм. Подпрограмма может обратиться к другой подпрограмме, но всегда должна возвращать управление вызывающей программе. Разрешается любая глубина вызова подпрограмм, однако необходимо следить за тем, чтобы возврат из подпрограммы осуществлялся в той же последовательности, что и вызов. Подпрограммы могут находиться в любом месте программы, но часто используемые рекомендуется размещать в начале программы. Рассмотрим пример использования оператора GOSUB на программе получения простых чисел.

```
10 REM "Программа получения простых чисел"
20 INPUT "СКОЛЬКО ВЫ ХОТИТЕ ПОЛУЧИТЬ ПРОСТЫХ ЧИСЕЛ
        ==> ";N
30 DIM X(N)
40 X(1)=1
50 X(2)=2
60 X(3)=3
70 X(4)=5
80 Y=5
```

```
90 I=3
100 GOSUB 500
110 Y=Y+2
120 GOSUB 500
130 Y=Y+4
140 IF I<N THEN GOTO 100
150 GOTO 600
500 REM "Подпрограмма"
510 FOR J=4 TO I
520 Z=X(J)
530 IF Z*Z>Y THEN GOTO 570
540 R=Y-INT(Y/Z)*Z
550 IF R=0 THEN RETURN
560 NEXT J
570 I=I+1
580 X(I)=Y
590 IF I<=N THEN RETURN
600 REM  "Вывод полученного  массива простых чисел
        на экран"
610 PRINT "Следующий список содержит ";N;" простых
          чисел"
620 PRINT
630 K=0
640 FOR I=1 TO N
650 PRINT TAB(7*K);X(I);
660 K=K+1
670 IF K<9 THEN GOTO 700
680 K=0
690 PRINT
700 NEXT I
710 END
```

Если в программе обращаются к подпрограмме только один раз, то рекомендуется использовать не оператор GOSUB, а оперзтор GOTO.

С помощью оператора вызова подпрограммы ON ... GOSUB можно обратиться к одной из нескольких заданных подпрограмм. Переход осуществляется в зависимости от значения выражения, стоящего после ключевого слова ON. Это значение определяет, какая подпрограмма будет вызываться. В списке после ключевого слова GOSUB может быть любое количество номеров строк. Некоторые из них могут повторяться. Если значение выражения не является целым числом, то используется целая часть этого значения для определения, куда должно быть передано управление.

В языке Бейсик отдельные программы, сохраненные на НГМД, можно использовать как подпрограммы и вызывать из основной (корневой) программы. Для этой цели команды RUN, LOAD, MERGE и другие используются в программном режиме. (Подробно эти команды описаны в гл. 4.) Эти возможности реализуются при создании программ меню.

Например:

```
90 PRINT "Вызов игры по номеру"
95 PRINT
100 PRINT "       ИГРЫ"
105 PRINT
110 PRINT " 1. ОТЕЛЛО (реверси) (человек-
           компьютер)"
```

```
120 PRINT " 2. ОТЕЛЛО (человек-человек)"
130 PRINT " 3.   МОРСКОЙ БОЙ"
140 PRINT " 4.   СКАЧКИ"
150 PRINT " 5.   НАРДЫ"
160 PRINT " 6.   ТРЕХМЕРНЫЕ КРЕСТИКИ-НОЛИКИ"
170 PRINT " 7.   КРОСС"
180 PRINT " 0.   КОНЕЦ ПРОГРАММЫ"
190 KK$=INKEY$ : IF KK$="" THEN GOTO 190
200 IF KK$=" " THEN GOTO 190
210 IF KK$="0" THEN END
220 IF KK$="1" THEN NAME$="OTHELLO.BAS"
230 IF KK$="2" THEN NAME$="OTHELLO1.BAS"
240 IF KK$="3" THEN NAME$="BOJ.BAS"
250 IF KK$="4" THEN NAME$="HORSE.BAS"
260 IF KK$="5" THEN NAME$="NARD.BAS"
270 IF KK$="6" THEN NAME$="QUBIC.BAS"
280 IF  KK$="7" THEN  NAME$="GROSS.BAS" ELSE  GOTO
    300
290 RUN NAME$
300 PRINT "Вы набрали неправильный номер задания"
305 PRINT "Введите еще раз"
310 GOTO 190
```

Эта программа позволяет перейти к выполнению любой из перечисленных игровых программ. При нажатии клавиши с соответствующим номером игры происходит загрузка и выполнение нужной программы с помощью команды RUN, заданной в программной строке 290. В вызываемых программах рекомендуется создавать возможность возврата в программу МЕНЮ. В следующем примере для присоединения к основной программе отдельных подпрограмм используется команда MERGE также в программном режиме. Например:

```
100 K$=INKEY$ : IF K$="" THEN GOTO 100
105 IF K$=" " THEN GOTO 1000
110 IF K$="A" OR K$="a" THEN MERGE "PROGA.A"
120 IF K$="B" OR K$="L" THEN MERGE "PROGB.A"
130 IF K$="C" OR K$="c" THEN MERGE "PROGC.A"
140 IF K$ "D" OR K$="d" THEN MERGE "PROGD.A"
150 IF K$="E" OR K$="e" THEN MERGE "PROGE.A"
160 GOTO 10
```

Необходимо помнить, что программы PROGA.A, PROGB.A, PROGC.A, PROGD.A и PROGE.A должны быть сохранены в коде ASCII (КОИ-8), т. е. при записи этих программ на диск в команде SAVE надо указать параметр A (см. гл. 4).

Формирование программы из отдельных модулей очень удобно для отладки. Из таких программ можно создать свою библиотеку и формировать из этих модулей любые программы. При задании имен программ можно указывать дисковод, на котором хранятся отдельные программы.

Для работы с подпрограммами на языке ассемблера необходимо зарезервировать требуемую память. Бейсик использует всю память, доступную от его начального адреса вверх (максимально 64 Кбайт), поэтому только верхние адреса в памяти могут быть отведены для подпрограмм. Чтобы зарезервировать память для подпрограмм, можно использовать один из способов:

при вызове интерпретатора указать максимальное число байтов, которые могут быть использованы как рабочее пространство, например:

```
BASIC /M:&H8000
```

с помощью оператора CLEAR установить максимальное число байтов, которые могут быть использованы интерпретатором, например:

```
CLEAR &H8000
```

Когда вызывается подпрограмма на языке ассемблера, интерпретатор выделяет 16 бит стековой памяти для работы этой подпрограммы. Если необходимо больше стекового пространства, то стек интерпретатора Бейсика должен быть сохранен и дополнительно должен быть определен собственный стек подпрограммы на языке ассемблера. При возврате в Бейсик-программу, из которой осуществляется вызов, следует восстановить указатель стека интерпретатора.

Подпрограмма на языке ассемблера может быть загружена в память с помощью системного монитора, оператора POKE, а также с помощью программы LINK после трансляции.

Так, относительно короткие программы могут быть закодированы на машинном языке прямо в интерпретаторе, для этого необходимо выполнить следующие действия:

ввести в операторы DATA шестнадцатеричные значения кодов каждого байта в формате, определенном в каждой конкретной версии; если шестнадцатеричного представления в данной версии нет, то можно задавать десятичные значения;

выполнить цикл, который считывает байты данных, и затем занести их в область, зарезервированную ранее при вызове интерпретатора или оператором CLEAR.

Например:

```
:
:
1000 REM "N% - начальный адрес подпрограммы;
          M% - конечный адрес подпрограммы на
               машинном языке"
1010 FOR I%=N% TO M%
1020 READ J%
1030 POKE I%,J%
1040 NEXT
1050 SUBRT%=N%
1060 A%=2 : B%=3 : C%=0
1070 CALL SUBRT%(A%,B%,C%)
1080 PRINT C% :REM "C" - результат"
:
:
1900 DATA &H55,....
```

Для вызова подпрограмм, написанных на машинном языке, используются функциональный вызов USR и оператор CALL.

Условный оператор — оператор, осуществляющий проверку истинности выполнения определенного условия и задающий альтернативные последовательности выполнения команд.

В языке Бейсик это позволяет делать оператор IF.

```
100 IF A<>5 THEN GOTO 360 ELSE PRINT 5
```

Оператор IF имеет три части: часть IF (в примере IF A<>5), часть THEN (в примере THEN GOTO 360) и часть ELSE (в примере ELSE PRINT 5). Часть IF содержит логическую формулу, а части THEN и ELSE – один или несколько операторов, разделенных двоеточием. Часть ELSE может быть совсем опущена. При выполнении оператора IF сначала вычисляется логическая формула. Если это значение истинно, то выполняется часть THEN, а если это значение ложно, то часть ELSE, если она существует, или следующая программная строка. В данном примере будет осуществлен переход на программную строку 360, если значение переменной A не равно 5. Если значение переменной A равно 5, то будет печататься 5 и выполняться следующая программная строка.

```
10 INPUT "Введите число (0-19)";N
20 IF N<5 THEN PRINT "Должно быть больше"
30 IF N>5 THEN PRINT "Должно быть меньше"
40 IF N<>5 THEN GOTO 10
50 PRINT "Вы угадали число"
RUN
Введите число? 7
Должно быть меньше
Введите число? 4
Должно быть больше
Введите число? 5
Вы угадали число
Ok
```

В программной строке 10 осуществляется ввод числа в программу. В программной строке 20 проверяется условие: введенное число меньше 5? Если меньше, то печатается сообщение "Должно быть больше". В строке 30 проверяется условие: введенное число больше 5? Если больше, то печатается сообщение "Должно быть меньше". Программная строка 40 проверяет введенное число на неравенство пяти. Если число не равно пяти, то осуществляется переход на строку 10, и программа продолжает выполняться с начала. Если же введенное число равно пяти, то печатается сообщение "Вы угадали число", и выполнение программы заканчивается.

В логической формуле части IF могут быть использованы операции отношения: равно, не равно, меньше, больше, меньше или равно, больше или равно и логические операции: AND, OR, XOR, NOT, EQV и IMP.

Переменные в логической формуле оператора IF могут быть как числовыми, так и строковыми.

Цикл – программная реализация повторяющихся операций – многократно повторяемое выполнение одной и той же последовательности инструкций в программе, используемых при обработке регулярных массивов данных, вычислении рекурсивных соотношений и в других эффективных вычислительных процедурах.

Условиями выхода из цикла могут являться достижение предельного адреса, обнуление счетчика циклов, сходимость и другие условия.

В языке Бейсик существуют четыре способа организации циклов, которые будут рассмотрены для следующей задачи: создать таблицу натуральных чисел от единицы до ста, квадратов, кубов и корней этих чисел и вывести результаты вычислений на печатающее устройство.

Первый способ.

```
10 LPRINT "ЧИСЛО";TAB(10);"КОРЕНЬ",TAB(20);
20 LPRINT "КВАДРАТ",TAB(30);"КУБ"
21 LPRINT 1;TAB(10),SQR(1);TAB(20);1^2;TAB(30);1^3
```

```
22 LPRINT 2;TAB(10);SQR(2);TAB(20);2^2;TAB(30);2^3
 :
 :
120 LPRINT 100,TAB(10);SQR(100);TAB(20);100^2;TAB(
    30),100^3
```

В программных строках 10 и 20 выводится на печатающее устройство заголовок таблицы. В следующих строках вычисляются и выводятся на печатающее устройство само число, его квадратный корень, квадрат и куб. Каждая программная строка обрабатывает конкретную константу от 1 до 100.

Второй способ. При первом способе организации цикла каждое значение, которое надо вычислить и напечатать, задается в программной строке как константа. Зададим эти значения как переменные и будем присваивать этим переменным конкретные значения. Используя условный оператор IF, который проверяет заданное условие, и оператор безусловного перехода GOTO, создаем цикл.

```
10 LPRINT "ЧИСЛО";TAB(10);"КОРЕНЬ";TAB(20);
20 LPRINT "КВАДРАТ";TAB(30);"КУБ"
30 X=1
40 LPRINT X;TAB(10);SQR(X);TAB(20);X^2;TAB(30);X^3
50 X=X+1
60 IF X<=100 THEN GOTO 40
```

Таким способом для программирования этого же задания потребовалось 6 программных строк. Программные строки 10 и 20 печатают заголовок таблицы. Программная строка 30 присваивает переменной X начальное значение, равное 1. Программная строка 40 вычисляет и выводит на печатающее устройство число, его корень, квадрат и куб. Программная строка 50 увеличивает значение переменной X на 1. В программной строке 60 проверяется значение переменной X: если оно меньше или равно 100, то осуществляется переход на программную строку 40, и переменная X с новым значением продолжает обрабатываться. А если значение переменной X больше 100, то выполнение программы прекращается.

Следующие два способа иллюстрируют автоматическую организацию цикла.

Третий способ. Создадим автоматический цикл, используя операторы FOR...NEXT.

```
10 LPRINT "ЧИСЛО";TAB(10);"КОРЕНЬ";TAB(20);
20 LPRINT "КВАДРАТ";TAB(30);"КУБ"
30 FOR X=1 TO 100
40 LPRINT X;TAB(10);SQR(X);TAB(20);X^2;TAB(30);X^3
50 NEXT
```

Таким способом для программирования этого же задания потребовалось 5 программных строк. Программные строки 10 и 20 выводят на печатающее устройство заголовок таблицы. Программная строка 30 является заголовком цикла, в котором указывается начальное, конечное значения счетчика цикла и шаг цикла. В этом примере шаг цикла, равный 1, задан по умолчанию. Шаг цикла – это число, которое прибавляется к значению счетчика. Программные строки, следующие за оператором FOR, выполняются до тех пор, пока не встретится оператор NEXT. Затем к содержимому счетчика прибавляется значение шага. Выполняется проверка счетчика на конечное значение. И в зависимости от этого цикл повторяется или заканчивается, и выполняются операторы, следующие за оператором NEXT. Таким образом, программная строка 40, выполняющая вычисления квадрата, куба и корня и выводящая эти значения на печать, выполнилась 100 раз. Переменная счетчика X увеличивается на 1 автома-

тически, и каждый раз программная строка 40 обрабатывает новое значение переменной X, которая используется в этой строке для вычисления корня, квадрата и куба. Использование переменной счетчика X в программной строке 40 разрешено, так как в этой строке не изменяется само значение переменной X.

Четвертый способ. Создадим автоматический цикл, используя операторы WHILE...WEND.

```
10 LPRINT "ЧИСЛО";TAB(10);"КОРЕНЬ";TAB(20);
20 LPRINT "КВАДРАТ";TAB(30);"КУБ"
30 X=1
40 WHILE X<=100
50 LPRINT X;TAB(10);SQR(X);TAB(20);X^2;TAB(30);X^3
60 X=X+1
70 WEND
```

Для программирования этого же примера таким способом потребовалось 7 программных строк. Программные строки 10 и 20 выводят на печатающее устройство заголовок таблицы. В программной строке 30 переменной X присваивается начальное значение, равное 1. Программная строка 40 является заголовком цикла. Программная строка 50 выполняет вычисления корня, квадрата и куба числа и выводит их значение на печатающее устройство. В программной строке 60 значение переменной X увеличивается на 1. Программные строки или операторы, расположенные между операторами WHILE и WEND, выполняется до тех пор, пока заданное условие (а именно, X <= 100) истинно. Оператор WEND является концом цикла.

3.4. Встроенные функции

Встроенные функции – подпрограммы, обращение к которым в языке Бейсик стандартизовано. При записи встроенных функций аргумент функции должен быть заключен в скобки и записан после имени функции.

В качестве аргумента может быть использовано число, переменная или выражение. Выражение может содержать обращение к другим функциям. Например:

```
10 INPUT X
20 PRINT X,SIN(X),COS(X),TAN(X),1/TAN(X)
30 PRINT SIN(SQR(X)+1)
```

Эта программа выводит на экран число, его синус, косинус, тангенс, котангенс, а также синус квадратного корня числа X плюс 1. Число X вводится с клавиатуры.

Следующая программа, использующая встроенные функции, находит числа I, J и K, для которых I*I+J*J=K*K.

```
10 N=100
20 FOR I=1 TO N-1
30 FOR J=I+1 TO N
40 K=SQR(I*I+J*J)
50 IF K<>INT(K) THEN GOTO 70
60 PRINT I,J,K
70 NEXT J,I
80 END
```

Встроенные функции в основном используются для математических преобразований, для перевода переменных одного типа в другой и для преобразования строк (строковые функции).

Математические функции – к математическим функциям в языке Бейсик относятся тригонометрические функции, абсолютное значение числа (ABS), экспонента числа (EXP), извлечение квадратного корня (SQR), образование знака, целой части и натурального логарифма числа (SGN, FIX и LOG соответственно) и определение самого большого целого числа меньшего или равного аргументу (INT).

При решении некоторых задач, особенно статистических, игровых и обучающих, может потребоваться, чтобы числа задавались случайным образом. Для получения случайных чисел используется функция RND, которая генерирует псевдослучайные числа в диапазоне от 0 до 1. При каждом выполнении программы будет генерироваться одна и та же последовательность случайных чисел. В некоторых задачах требуется, чтобы последовательность генерировалась каждый раз новая. Для этого рекомендуется сделать переменным аргумент функции RND или вставить в программу оператор RANDOMIZE до первого обращения к функции RND, поэтому лучше его вставлять в начало программы. Случайные числа рекомендуется использовать в обучающих программах для задания начальных условий в различных вариантах, чтобы эти условия были непредсказуемы для обучающихся с помощью этой программы.

Тригонометрические функции – язык Бейсик предоставляет возможность пользоваться следующими тригонометрическими функциями: ATN – арктангенс числа; COS – косинус угла; SIN – синус угла; TAN – тангенс угла.

Аргументы в тригонометрических функциях задаются в радианах. Для перевода градусов в радианы их нужно умножить на $\pi/180$. Важно отметить, что при переводе из радиан в градусы и обратно, чем точнее величина π, тем точнее будет выполнено преобразование. (Для справки: более точное значение числа π = 3.14159265358.)

В языке Бейсик нет встроенных функций секанса, косеканса, котангенса и т. д., но их можно вычислить, используя существующие в языке встроенные функции:

секанс – SEC(X) = 1/COS(X)
косеканс – CSC(X) = 1/SIN(X)
котангенс – COT(X)=1/TAN(X)
арксинус – ARCSIN(X)=ATN(X/SQR(1–X*X))
арккосинус – ARCCOS(X)=1.570796–ATN(X/SQR(1–X*X))
арксеканс – ARCSEC(X)=ATN(SQR(X*X–1))+(X<0)*3.141593
арккосеканс – ARCCSC(X)=ATN(1/SQR(X*X–1))+(X<0)*3.141593
арккотангенс – ARCCOT(X)=1.570796–ATN(X)
гиперболический синус – SINH(X)=(EXP(X)–EXP(–X))/2
гиперболический косинус – COSH(X)=(EXP(X)+EXP(–X))/2
гиперболический тангенс – TANH(X)=(EXP(X)–EXP(–X))/(EXP(X)+EXP(–X))
гиперболический секанс – SECH(X)=2/(EXP(X)+EXP(–X))
гиперболический косеканс – CSCH(X)=2/(EXP(X)–EXP(–X))
гиперболический котангенс – COTH(X)=(EXP(X)+EXP(–X))/(EXP(X)–EXP(–X))
арксинус гиперболический – ARCSINH(X)=LOG(X+SQR(X*X+1))
арккосинус гиперболический – ARCCOSH(X)=LOG(X+SQR(X*X–1))
арктангенс гиперболический – ARCTANH(X)=LOG((1+X)/(1–X))/2
арксеканс гиперболический – ARCSECH(X)=LOG((1+SQR(1+SQR(1–X*X))/X)
арккосеканс гиперболический – ARCCSCH(X)=LOG((1+SGN(X)*SQR(1+X*X))/X)
арккотангенс гиперболический – ARCCOTH(X)=LOG((X+1)/(X–1))/2

Чтобы использовать эти функции, можно применить оператор DEF FN.

Например, для арксинуса гиперболического

```
DEF FNAS(X)=LOG(X+SQR(X*X+1))
```

с последующим обращением к наименованию функции, когда необходимо,

```
Z=FNAS(Y).
```

Перевод переменных одного типа в другой — преобразование типа данных, т. е. преобразование последовательности символов в число или наоборот, изменение точности числовой переменной и другие преобразования подобного типа в языке Бейсик осуществляются с помощью встроенных функций:

CINT переводит числовой аргумент в целое число;

CSNG переводит числовой аргумент в вещественное число одинарной точности;

CDBL переводит числовой аргумент в вещественное число двойной точности;

CVI, CVS и CVD переводят строковые переменные в числа заданной точности;

MKI$, MKS$ и MKD$ переводят числа в строковые переменные.

Строковые функции. Для работы со строковыми данными используются следующие встроенные функции:

ASC определяет код ASCII (КОИ-8) первого символа строковой переменной;

```
PRINT ASC("S"),ASC("Ф")
 83        230
```

CHR$ определяет символ по заданному коду ASCII (КОИ-8);

```
CRLF$=CHR$(13)+CHR$(10)
PRINT CHR$(&H74)+CHR$(234)
tй
```

LEN вычисляет длину строки, т. е. число символов строковой переменной;

```
PRINT LEN("BASIC-программа")
 15
```

LEFT$ определяет самые левые символы строковой переменной;

```
PRINT LEFT$("BASIC-программа",5)
BASIC
```

RIGHT$ определяет самые правые символы строковой переменной;

```
PRINT RIGHT$ ("BASIC-программа", 9)
программа
```

MID$ определяет часть строки в строке, начиная с заданной позиции;

```
PRINT MID$("ПРОГРАММА",4,5)
ГРАММ
```

VAL определяет числовое значение строковой переменной;

```
PRINT VAL("35")+VAL("15")
 50
```

STR$ вычисляет строковое значение числовой переменной;

```
PRINT STR$(35)+STR$(15)
3515
```

HEX$ переводит число в шестнадцатеричную строку;

```
PRINT HEX$(42)
 2A
```

OCT$ переводит число в восьмеричную строку;

```
PRINT OCT$(42)
 60
```

SPACE$ воспроизводит строку пробелов;

```
PRINT SPACE$(5);"ПРОБЕЛ";SPACE$(5);"SPACE"
     ПРОБЕЛ      SPACE
```

STRING$ вычисляет символ, повторяемый заданное число раз;

```
PRINT STRING$(5,42);"BASIC",STRING(5,47)
*****BASIC/////
```

INSTR определяет позицию части строки в строке, начиная с заданной позиции;

```
PRINT INSTR(2,"ПРОГРАММА","Р")
5
```

Перед использованием строковых данных для чистки строковых переменных и увеличения строкового пространства рекомендуется использовать оператор CLEAR в начале программы. Функция FRE с любым строковым аргументом выполняет чистку неиспользованных областей памяти, которые были однажды использованы для строковых переменных, и определяет число свободных байтов.

3.5. Работа с оперативной памятью и портами ввода-вывода

Важнейшей особенностью языка Бейсик является включение в его состав инструкций **работы с оперативной памятью.**

Функция PEEK и оператор POKE используются соответственно для считывания и изменения содержимого ячеек оперативной памяти.

Функция PEEK позволяет считывать содержимое любой ячейки оперативной памяти в ПЭВМ.

```
10 INPUT " ->";N
20 PRINT N,PEEK(N),CHR$(PEEK(N))
30 N N+1
40 GOTO 20
```

Программная строка 10 устанавливает начальный адрес, заданный десятичным числом, начиная с которого будет печататься содержимое ячеек. Программная строка 20 выводит на печатающее устройство адрес (N), т. е. номер байта, содержимое которого необходимо проверить; содержимое этого байта (PEEK(N)), выраженное десятичным числом; графический символ (CHR$ (PEEK(N))), код которого находится по адресу N.

Изменяя значение N, можно выполнять эту программу для различных областей памяти.

Оператор POKE позволяет заносить содержимое байта по заданному адресу памяти. Запишем некоторую информацию в память, начиная с шестнадцатеричного адреса 6000.

```
10 N=&H6000
20 READ D
30 POKE N,D
40 N=N+1
50 IF N=&H600B THEN END
60 GOTO 20
70 DATA 80,69,69,75,45,65,45,66,79,79,33
```

Программа выполняется следующим образом. Программная строка 10 инициализирует начальный адрес. Программная строка 20 считывает число из оператора DATA. Программная строка 30 записывает значение D по адресу N. Программная строка 40 увеличивает адрес на 1. Программная строка 50 заканчивает выполнение после того, как все данные из оператора DATA будут записаны. Программная строка 60 возвращает управление для дальнейшего считывания значений из оператора DATA. Программная строка 70 содержит данные, которые будут записаны в память.

Следующая программа проверит, что записалось в память оператором POKE.

```
200 FOR N=&H6000 TO &H600A
210 PRINT HEX$(N),PEEK(N)
220 NEXT
RUN
6000          80
6001          69
6002          69
6003          75
6004          45
6005          65
6006          45
6007          66
6008          79
6009          79
600A          33
```

Оператор POKE может быть использован для установки курсора в требуемую позицию.

Порт ввода-вывода — это аппаратно-адресуемое устройство ввода-вывода, которому могут быть переданы (или от него получены) данные.

Язык Бейсик позволяет непосредственно считывать и изменять содержимое ячеек оперативной памяти, порта ввода-вывода, что позволяет писать программы управления на языке Бейсик, а не на языке ассемблера.

В языке Бейсик имеются инструкции INP и OUT, аналогичные по своему действию операторам IN и OUT языка ассемблера. Так, оператор OUT записывает числовые значения в порт ввода-вывода, а функция INP определяет значение байта заданного порта. Необходимо помнить, что номера портов в различных вычислительных машинах могут отличаться.

В языке Бейсик есть возможность приостанавливать выполнение программы, пока обрабатывается состояние входного порта машины, т. е. пока входной порт машины не выработает определенного сигнала. Для этой цели используется оператор WAIT. Но необходимо быть осторожным, так как с помощью оператора WAIT можно войти в бесконечный цикл, для останова которого необходимо перезагружать вычислительную машину.

3.6. Описание инструкций

Таблица 3.1

Ключевое слово	Номер версии									
	1	2	3	4	5	6	7	8	9	10
ABS	+	+	+	+	+	+	+	+	+	+
ACS			+							
ASC	+	+		+	+	+	+	+	+	+
ASN			+							
AT			+							
ATN	+	+	+	+	+	+	+	+	+	+
BCD		+								
BIN		+								
BIN$		+					+	+	+	
CALL		+		+		+			+	+
CDBL					+	+	+	+	+	+
CHR$	+	+	+	+	+	+	+	+	+	+
CINT					+	+	+	+	+	+
CLEAR		+	+	+	+	+	+	+	+	+
CLR				+						+
COMMON						+				
COS	+	+	+	+	+	+	+	+	+	+
CSNG					+	+	+	+	+	+
DATA	+	+	+	+	+	+	+	+	+	+
DEF FN	+	+	+	+	+	+	+	+	+	+
DEF SEG										+
DEF USR				+	+	+	+	+	+	+
END	+	+		+	+	+	+	+	+	+
EXP	+	+	+	+	+	+	+	+	+	+
FIRST		+								
FIX	+				+	+	+	+	+	+
FN	+	+	+	+	+	+	+	+	+	+
FOR	+	+	+	+	+	+	+	+	+	+
FRE	+	+		+	+	+	+	+	+	+
FRE$		+								
GET		+		+						
GET$		+								
GOSUB	+	+	+	+	+	+	+	+	+	+
GOTO	+	+	+	+	+	+	+	+	+	+
HEX$	+					+	+	+	+	+
HIMEM				+						
IF	+	+	+	+	+	+	+	+	+	+
IN		+	+							
INKEY$	+		+	+	+	+	+	+	+	+
INP					+	+			+	+
INPUT	+	+	+	+	+	+	+	+	+	+
INSTR	+	+			+	+	+	+	+	+
INT	+	+	+	+	+	+	+	+	+	+
JOIN		+								
LAST		+								

Продолжение табл. 3.1

Ключевое слово	Номер версии									
	1	2	3	4	5	6	7	8	9	10
LEFT$	+	+		+	+	+	+	+	+	+
LEN	+	+	+	+	+	+	+	+	+	+
LET	+	+	+	+	+	+	+	+	+	+
LINE INPUT	+				+	+	+	+	+	+
LINPUT		+								
LN			+							
LOG	+	+		+	+	+	+	+	+	+
LOMEM				+						
LSBYTE		+								
LSHIFT		+								
MEM					+					
MID$	+	+		+	+	+	+	+	+	+
MSBYTE		+								
NEXT	+	+	+	+	+	+	+	+	+	+
NULL		+				+				
OCT$		+				+	+	+	+	+
ON	+	+		+	+	+	+	+	+	+
OUT		+	+		+	+			+	+
PEEK	+	+	+	+	+	+	+	+	+	+
PI			+							
POKE	+	+	+	+	+	+	+	+	+	+
POP				+						
PRINT	+	+	+	+	+	+	+	+	+	+
RANDOM					+					
RANDOMIZE	+	+	+			+		+		+
READ	+	+	+	+	+	+	+	+	+	+
RESET		+			+	+				+
RESTORE	+	+	+	+	+	+	+	+	+	+
RETURN	+	+	+	+	+	+	+	+	+	+
RIGHT$	+	+		+	+	+	+	+	+	+
RND	+	+	+	+	+	+	+	+	+	+
ROTATE		+								
RSHIFT		+								
SCALL		+								
SENSE		+								
SET		+								
SGN	+	+	+	+	+	+	+	+	+	+
SPACE$	+					+	+	+	+	+
SPC		+		+		+	+	+	+	+
SQR	+	+	+	+	+	+	+	+	+	+
STR		+								
STR$	+	+	+	+	+	+	+	+	+	+
STRING$	+				+	+	+	+	+	+
SWAP						+	+	+	+	+
TAB	+	+	+	+	+	+	+	+	+	+
TAN	+	+	+	+	+	+	+	+	+	+
TEST		+								

Окончание табл. 3.1

Ключевое слово	Номер версии									
	1	2	3	4	5	6	7	8	9	10
UNS		+								
USING	+				+	+	+	+	+	+
USR			+	+	+	+	+	+	+	+
VAL	+	+	+	+	+	+	+	+	+	+
VAL$			+							
VARPTR					+	+	+	+	+	+
WAIT		+		+		+			+	+
WEND	+					+			+	+
WHILE	+					+			+	+
WIDTH						+	+	+	+	+
WRITE				+		+				+

ABS – функция определения абсолютного значения числа X

```
Y=ABS(X)
```

X – числовое выражение.

Абсолютное значение всегда положительно или равно нулю. Например:

```
PRINT ABS(7*(-5))
 35
Ok
```

ACS – функция вычисления арккосинуса числа X

```
Y=ACS(X)
```

X – числовое выражение.

Результат выполнения функции выдается в радианах. Если значение числа X не лежит в диапазоне от –1 до +1, то выдается сообщение об ошибке Invalid argument (Неправильный аргумент).

Например:

```
100 IF ACS(X)>0 THEN GOTO 300
```

ASC – функция определения кода ASCII (КОИ-8) первого символа строки X$

```
Y=ASC(X$)
```

Результатом работы функции является числовое значение. Например:

```
10 X$="TEST"
20 PRINT ASC(X$)
RUN
 84
Ok
```

ASN – функция вычисления арксинуса числа X

```
Y=ASN(X)
```

X – числовое выражение.

Результат выполнения функции выдается в радианах. Если значение числа X не лежит в диапазоне от −1 до +1, то выдается сообщение об ошибке Invalid argument (Неправильный аргумент).

Например:

```
10 FOR I=-1 TO 1 STEP 0.01
20 PRINT ASN(I)
30 NEXT I
```

AT – параметр перемещения текущей позиции вывода на экран

```
PRINT AT строка,столбец

INPUT AT строка,столбец
```

Перемещает текущую позицию вывода на экран, т. е. позицию, в которую должен выводиться следующий элемент, в заданные строку и столбец. Строки пронумерованы с 0 (начиная сверху) до 21; столбцы пронумерованы с 0 (начиная слева) до 31. Можно использовать параметр **AT**, чтобы определить ту позицию вывода, в которой уже что-то напечатано. Новый элемент будет выводиться на место старого. Используется только с операторами PRINT и INPUT. Например:

```
PRINT AT 11,16;"*"
```

Печатается символ * в центре экрана.

ATN – функция вычисления арктангенса числа X.

```
Y=ATN(X)
```

X – числовое выражение.

Результат выполнения функции вычисляется в радианах в диапазоне от $-\pi/2$ до $\pi/2$, где π = = 3.141593. Выражение X может быть задано в любом числовом формате, но вычисление функции ATN всегда выполняется с одинарной точностью. Для перевода радиан в градусы результат умножается на $180/\pi$. Например:

```
10 INPUT X
20 PRINT ATN(X)
RUN
?3
 1.249046
Ok
10 PI=3.141593
20 RADIANS=ATN(1)
30 DEGREES=RADIANS*180/PI
40 PRINT RADIANS,DEGREES
RUN
 .7853983         45
Ok
```

BCD – функция перевода целого двоичного числа в двоично-десятичное число

```
Y=BCD(выражение)
```

выражение – целочисленное выражение.

Двоично-десятичное представление – это представление, в котором каждые четыре бита представляют десятичное число между 0 и 9. Так как XYBASIC использует шестнадцатибитовые целые значения, то это значение может представлять четыре двоично-десятичные цифры или шестнадцать двоичных цифр. Например, значение #1234 представляет 4660, если рассматривать в двоичном представлении, и 1234, если это значение рассматривать в двоично-десятичном представлении.

Например:

```
10 A=17
20 PRINT BCD(A)
RUN
 0000000000010111
OK
```

BIN – функция перевода целого числа из двоично-десятичного представления в двоичное представление

```
Y=BIN(выражение)
```

выражение – целочисленное выражение.

Если выражение в функции BIN не является правомочным двоично-десятичным числом, то появляется сообщение об ошибке FC (Functional Call – Функциональный вызов).

Например:

```
10 A=17
20 PRINT BIN(A).
RUN
 0000000000001011
OK
```

BIN$ – функция, предназначенная для воспроизведения строки двоичных цифр, десятичный эквивалент которых задан в качестве аргумента

```
Y$=BIN$(n)
```

n – числовое выражение, значение которого изменяется от –32768 до 65535.

При отрицательном значении аргумента справедливо тождество: BIN$(–n) = BIN$(65535–n).

Например:

```
PRINT BIN$(14)
 1110
Ok
```

CALL – оператор вызова подпрограммы, написанной на машинном языке

Формат 1 (для версий XYBASIC, MBASIC и BASICA).

```
CALL числовая переменная [(переменная[,переменная]...)]
```

числовая переменная – имя числовой переменной. Значение переменной указывает начальный адрес памяти подпрограммы.

переменная – имя переменной, которая передается как аргумент в подпрограмму на машинном языке.

Формат 2 (для Бейсик-АГАТ).

```
CALL адрес
```

адрес – адрес подпрограммы.

Формат 1. Оператор CALL – один из способов связывания программ на машинном языке. Например:

```
110 OZ%=&H9000
120 CALL OZ%(A,B,C)
```

Для версии BASICA значение переменной указывает начальный адрес подпрограммы, вызываемой как смещение в текущем сегменте памяти (определенном последним оператором DEF SEG). Например:

```
10 DEF SEG=&H8000
20 PROG=0
30 CALL PROG(ST$,CH)
```

CDBL – функция перевода числового выражения X в число двойной точности

```
Y=CDBL(X)
```

Например:

```
10 A=454.67
20 PRINT A,CDBL(A)
RUN
 454.67           454.6699829101563
```

CHR$ – функция образования символьного эквивалента числового значения n, изменяющегося от 0 до 255

```
Y$=CHR$(n)
```

Функция CHR$ образует строку из одного символа n в коде ASCII (КОИ-8) и используется для получения специального символа на экране или на печатающем устройстве.

Например:

```
PRINT CHR$(65);CHR$(32);CHR$(70)
A F
Ok
```

CINT – функция перевода выражения X в целое число округлением дробной части

```
Y=CINT(X)
```

Если X не лежит в диапазоне целых чисел используемой версии языка Бейсик, то появляется сообщение об ошибке Overflow (Переполнение).

Например:

```
PRINT CINT(45.67)
 46
Ok
PRINT CINT(-2.89)
-3
Ok
```

CLEAR – оператор очистки данных

Формат 1 (для XYBASIC, Бейсик-TRS–80, Бейсик-ПК8010, Бейсик-ПК8020).

```
CLEAR [строковое пространство]
```

строковое пространство – размер памяти, разрешенный для строковых переменных. Для версии XYBASIC по умолчанию 256 байт.

Формат 2 (для Бейсик-Спектрум+2).

```
CLEAR [максимальный адрес]
```

максимальный адрес – максимально допустимый адрес.

Формат 3 (для Бейсик-АГАТ).

```
CLEAR
```

Формат 4 (для MBASIC, BASICA).

```
CLEAR[,[максимальный адрес][,стековое пространство]]
```

максимальный адрес – счетчик байтов, который устанавливает максимальное число байтов, доступных интерпретатору для запоминания программы и данных, т. е. максимально доступный адрес памяти;

стековое пространство – число байтов, устанавливаемое для стекового пространства. По умолчанию 1000 байт (512 байт для BASICA) или 1/8 часть доступной памяти, в зависимости от того, что меньше.

Формат 5 (для MSX-BASIC).

```
CLEAR [строковое пространство[,максимальный адрес]]
```

строковое пространство – размер памяти, разрешенный для строковых переменных. По умолчанию – 200 байт;

максимальный адрес – максимально допустимый адрес.

Оператор CLEAR устанавливает все числовые переменные в нуль и очищает строковые переменные, закрывает все открытые файлы, устанавливает конец памяти и количество строкового или стекового пространства. Оператор CLEAR освобождает всю память, используемую для данных, не стирая текущей программы. После работы команды CLEAR массивы не определены, числовые переменные принимают нулевые значения, строковые переменные очищаются, а любая информация, определенная оператором DEF, включая DEF FN, DEF SEG, DEF USR, DEFINT, DEFDBL, DEFSNG и DEFSTR, теряется.

Для формата 4 при необходимости зарезервировать память для программы на машинном языке, задается параметр максимальный адрес. При использовании большого количества операторов GOSUB или циклов FOR...NEXT рекомендуется задать параметр стековое пространство. Например:

```
CLEAR
CLEAR,32768,2000
```

CLR – команда очистки данных

```
CLR
```

См. оператор CLEAR.

COMMON – оператор передачи переменных в программы, вызванные оператором CHAIN

```
COMMON переменная [,переменная]...
```

Оператор COMMON может появляться в любом месте программы, хотя рекомендуется, чтобы он появлялся в начале. Одна и та же переменная не может появляться в двух и более операторах COMMON. Переменные массива определяются наличием скобок () в имени переменной. Если должны быть переданы все переменные, то в операторе CHAIN должен быть параметр ALL, а оператор COMMON не указывается.

Например:

```
100 COMMON A,B,C,D(),G$
110 CHAIN "A:PROG3",10
```

COS – функция определения косинуса числа X, заданного в радианах

```
Y=COS(X)
```

Чтобы перевести градусы в радианы, их надо умножить на π/180, где π = 3.141593. Вычисление косинуса выполняется с одинарной точностью. Например:

```
10 X=2*COS(.4)
20 PRINT X
RUN
 1.842122
Ok
10 PI=3.141593
20 PRINT COS(PI)
30 DEGREES=180
40 RADIANS=DEGREES*PI/180
50 PRINT COS(READIANS)
RUN
-1
-1
Ok
```

CSNG – функция перевода числового выражения X в число одинарной точности

```
Y=CSNG(X)
```

Подобна функциям CINT и CDBL для перевода чисел в целое и двойной точности. Например:

```
10 A#=975.3421222#
20 PRINT A#,CSNG(A#)
RUN
 975.342122          975.342
Ok
```

DATA – оператор, который запоминает числовые и строковые константы, подлежащие считыванию оператором READ

```
DATA список констант
```

Константы в списке разделяются запятыми. Оператор DATA – неисполнительный оператор, может быть расположен в любом месте программы и может содержать столько констант, сколько их поместится на строке экрана дисплея. В программе может быть использовано любое число операторов DATA.

Данные, которые содержатся в операторах DATA, могут считаться одним продолжающимся списком данных, невзирая на количество элементов в строке и на место, где расположены строки.

Операторы READ имеют доступ к операторам DATA в порядке номеров строк. В списке констант оператора DATA не допускаются выражения. Числовые константы могут быть заданы в любом формате. Нет необходимости в операторе DATA заключать в кавычки строковые константы, если они не содержат запятых, двоеточий, пробелов и символов русского алфавита. Тип переменной, заданной в операторе READ, должен соответствовать типу констант в операторе DATA, иначе появится сообщение об ошибке Sintax error (Синтаксическая ошибка).

Для того чтобы заново считать данные с начала списка операторов DATA, надо использовать оператор RESTORE.

DEF FN – оператор определения и именования функции, определенной пользователем

DEF FN имя[(аргумент[,аргумент]...)]=определение

имя – имя действительной переменной. Это имя, которому предшествует FN, становится именем функции (в Бейсик-Спектрум+2 имя может состоять только из одной буквы и символа '$');

аргумент – имя переменной в описании функции, которое будет заменено значением при вызове функции (в Бейсик-Спектрум+2 количество аргументов не более одного);

определение – выражение, выполняющее операцию функции.

Определение функции ограничивается одной строкой. Аргументы, которые появляются в выражении определения, служат только для определения функции, они не активизируют программные переменные, которые имеют то же имя. Имя переменной, используемое в описании функции, не должно появляться в списке аргументов. Если это сделано, то подставляется значение аргумента при вызове функции. Но может быть использовано текущее значение переменной. Аргументы в списке соответствуют один к одному значениям, которые передаются при вызове функции.

Тип функции определяет числовое или строковое значение, образуемое функцией. Если тип определения не соответствует типу функции, то появляется сообщение об ошибке Type mismatch (Ошибочный тип). Если функция числовая, то значение выражения определения переводится в число с точностью, описанной именем, перед возвратом в вызывающий оператор. Оператор DEF FN должен быть выполнен для определения функции перед ее вызовом, иначе появится сообщение об ошибке Undefined user function (Неопределенная функция пользователя).

Функция может быть определена более одного раза. Действительным является самое последнее выполненное определение. Допускается возможность определения рекурсивных функций, т. е. функций, которые вызывают сами себя. Однако если не обеспечивается способ остановки рекурсии, появляется сообщение об ошибке Out of memory (Вне памяти). Нельзя использовать оператор DEF FN в прямом режиме.

Например:

```
10 PI=3.141593
20 DEF FNarea(R)=PI*R^2
30 INPUT "РАДИУС ";RADIUS
40 PRINT "ПЛОЩАДЬ КРУГА ";FNarea(RADIUS)
RUN
РАДИУС ?2
ПЛОЩАДЬ КРУГА  12.56637
Ok
```

Строка 20 определяет функцию FNarea, которая вычисляет площадь круга с радиусом R. Функция вызывается в строке 40.

Пример функции с двумя аргументами:

```
10 DEF FNmlld(X,Y)=X-(INT(X/Y)*Y)
20 A=FNmlld(7.4,4)
30 PRINT A
RUN
 3.4
Ok
```

DEF SEG – оператор определения текущего сегмента памяти

```
DEF SEG=[адрес]
```

адрес – целочисленное выражение от 0 до 65535.

Инструкции BLOAD, BSAVE, CALL, PEEK, POKE, VARPTR и USR будут определять действительный физический адрес своей операции как смещение в заданном сегменте.

Если параметр адрес не указан, то значение устанавливается в сегмент данных (DS) Бейсика. Это является начальным значением по умолчанию. Сегмент данных – это начало рабочего пространства в памяти.

Если адрес задан, то он должен быть значением, выровненным на 16-байтовую границу. Значение сдвигается влево на 4 бита (умножается на 16), чтобы сформировать адрес сегмента для последующей операции. Интерпретатор не выполняет дополнительной проверки, чтобы убедиться, что значение сегмента действительно. Слова DEF и SEG должны быть разделены пробелом.

Если значение адреса лежит вне указанного диапазона, то появляется сообщение об ошибке Illegal function call (Неверный функциональный вызов) и будет сохраняться предыдущее значение.

Например:

```
10 DEF SEG 'восстановить сегмент DS в Бейсике
20 DEF SEG=&H8000 'установить сегмент буфера
                     графического экрана
```

DEF USR – оператор определения начального адреса подпрограммы на машинном языке, которая позднее вызывается функцией USR

Формат 1 (для Бейсик- АГАТ).

```
DEF USR=адрес
```

Формат 2 (для Бейсик-TRS–80, MBASIC, Бейсик-ПК8010, Бейсик-ПК8020, MSX-BASIC и BASICA).

```
DEF USR[цифра]=адрес
```

цифра – любая цифра от 0 до 9;

адрес – целочисленное выражение от 0 до 65535.

Цифра соответствует номеру подпрограммы USR, чей адрес определяется. Если цифра не указана, по умолчанию принимается DEF USR0. Значение адреса может задаваться в десятичном и шестнадцатеричном форматах и является действительным начальным адресом подпрограммы USR. Для переопределения начального адреса подпрограммы в программе может появляться любое число операторов DEF USR, так как разрешается доступ к стольким подпрограммам, сколько необходимо.

Например:

```
100 DEF USR0=&H9000
 :
500 X=USR0(X+2)
```

END – оператор завершения программы

```
END
```

Заканчивает выполнение программы, закрывает все файлы и возвращает управление на уровень команд. Оператор END может находиться в любом месте программы, но он не обязателен. Оператор END отличается от оператора STOP: не вызывает сообщения break или stop (выход или стоп); закрывает все файлы.

Например:

```
100 IF P>100 THEN END ELSE GOTO 570
```

EXP – функция вычисления экспоненциального значения числа X, т. е. e^x

```
Y=EXP(X)
```

Например:

```
10 X=2
20 PRINT EXP(X-1)
RUN
 2.718282
Ok
```

Аргумент X должен быть не больше определенного для каждой версии значения (например, для MBASIC – 87.3365), иначе возникнет переполнение.

FIRST и **LAST** – функции, которые определяют первый и последний адрес соответственно, используемые для запоминания текущей XYBASIC-программы

```
Y=FIRST
Y=LAST
```

Например:

```
PRINT FIRST,LAST,LAST-FIRST+1
```

В этом примере на экран выдается начальный адрес программы, конечный адрес программы и длина программы, расположенной в памяти.

FIX – функция определения целой части числового выражения X

```
Y=FIX(X)
```

При выполнении функции отбрасывается дробная часть числа справа от точки. Отличие функции FIX от INT заключается в том, что функция FIX одинаково работает с положительными и отрицательными числами. Например:

```
PRINT FIX(58.75)
 58
Ok
PRINT FIX(-58.75)
-58
Ok
```

FN – оператор, который вызывает функцию, определенную пользователем

```
Y=FNимя(аргумент)
```

Вызывает функцию, определенную пользователем в операторе DEF FN. Аргументы должны быть заключены в скобки; даже если аргументов нет, скобки должны присутствовать.

См. оператор DEF FN.

FOR...NEXT – оператор организации цикла

```
FOR переменная = выражение1 TO выражение2

[STEP выражение3]

:

NEXT [переменная]
```

переменная – целочисленная или одинарной точности переменная, которая будет использоваться как счетчик (в Бейсик-Спектрум+2 имя переменной должно быть из одной буквы);

выражение1 – начальное значение счетчика;

выражение2 – конечное значение счетчика;

выражение3 – шаг цикла.

Программные строки, следующие за оператором FOR, выполняются до тех пор, пока не встретится NEXT. Затем счетчик увеличивается на число, определенное выражением3 в STEP. Если шаг ие задан, то по умолчанию принимается равным единице. Затем выполняется проверка.

Если значение счетчика не больше конечного значения выражения2, то опять начинают выполняться программные строки, следующие за оператором FOR, и процесс повторяется. А если значение счетчика больше конечного значения выражения2, то начинают выполняться программные строки, следующие за оператором NEXT.

Если выражение3 отрицательно, то каждый раз в цикле значение счетчика уменьшается, и цикл выполняется, пока значение счетчика больше конечного значения.

Циклы FOR...NEXT могут быть вложены, т. е. один цикл FOR...NEXT может находиться (выполняться) внутри другого FOR...NEXT цикла.

Каждый вложенный цикл должен иметь свои имена переменных в качестве счетчика. Оператор NEXT для внутреннего цикла должен выполняться раньше оператора NEXT для внешнего цикла. Если вложенные циклы имеют одну точку конца, то в Бейсик-TRS–80, MBASIC, Бейсик-ПК8010, Бейсик-ПК8020, MSX-BASIC и BASICA может быть использован один оператор NEXT для всех переменных. Вид оператора NEXT

```
NEXT переменная1, переменная2, переменная3...
```

эквивалентен последовательности операторов:

```
NEXT переменная1

NEXT переменная2

NEXT переменная3

:
:
```

Переменная в операторе NEXT может быть не указана, в этом случае этот оператор NEXT будет заканчивать самый последний FOR...NEXT цикл. Например:

```
10 J=10 : K=30
20 FOR I=1 TO J STEP 2
30 PRINT I; : K=K+10 : PRINT K
40 NEXT
RUN
 1   40
 3   50
 5   60
 7   70
 9   80
Ok
```

FRE – функция определения количества свободных байтов после их предварительной очистки

```
Y=FRE(X)

Y=FRE(X$)
```

Определяет число байтов памяти, не использованных интерпретатором. Так как строки в языке Бейсик могут иметь переменную длину, то строковое пространство может стать фрагментированным. Функция FRE с любым строковым значением вызывает очистку перед указанием числа свободных байтов.

При очистке интерпретатор собирает все полезные данные и освобождает неиспользованные области памяти, которые были однажды использованы для строк. Интерпретатор будет автоматически делать очистку, когда она выполняется в неиспользованной рабочей области. Можно периодически использовать FRE("") для получения более короткой задержки при каждой очистке. Аргументы в функции FRE фиктивные. Например:

```
PRINT FRE(0)
 14542
Ok
```

FRE$ – функция определения неиспользованных байтов строкового пространства

```
Y=FRE$
```

Например:

```
10 CLEAR 256
20 PRINT FRE$
30 A$="DOG"
40 PRINT A$;FRE$
RUN
 256
DOG 253
```

GET – функция определения кода введенного символа

Формат 1 (для XYBASIC).

```
Y=GET
```

Формат 2 (для Бейсик-АГАТ).

```
GET X$
```

Формат 1. Определяет код ASCII введенного символа или 0, если символ не введен. Функция GET позволяет проверить, был ли введен символ во время выполнения программы.

Например:

```
10 IF GET=20 THEN GOSUB 100
20 I=I+1
30 GOTO 10
100 PRINT I;
110 RETURN
RUN
 46  100  126  169  183  201 ^C
BREAK AT LINE 20
OK
```

Так как XYBASIC автоматически отсекает бит четности в любом считанном символе, то нельзя с помощью GET получить код больше 127. И, конечно, не следует пытаться получить такие символы, как Ctrl–C, которые имеют специальное значение в XYBASIC.

Формат 2. Ввод одного символа с клавиатуры. Все символы равноправны. На экране введенный символ не отображается.

GET$ – определяет строковое значение введенного символа

```
Y=GET$
```

Функция GET$ позволяет проверить, был ли введен какой-либо символ во время выполнения программы. Если символ введен, то функция GET$ определяет строковое значение введенного символа. Если символ не введен, то определяется пустая строка. Например:

```
10 IF GET$="P" THEN GOSUB 100
20 I=I+1
30 GOTO 10
100 PRINT I;
110 RETURN
RUN
 57  113  181  193  369 ^C
BREAK AT LINE 20
OK
```

GOSUB....RETURN – операторы перехода к подпрограмме, написанной на языке Бейсик, и возврата из нее в текущую программу

```
GOSUB номер строки

        :

RETURN
```

номер строки – номер первой строки подпрограммы.

Подпрограмма может вызываться любое число раз из самой программы или из другой подпрограммы. Такие вложения подпрограмм ограничиваются только доступной памятью. Оператор RETURN заставляет Бейсик вернуться обратно из подпрограммы к оператору, следующему за вызвавшим эту подпрограмму оператором GOSUB. Подпрограмма может содержать больше одного оператора RETURN для возврата из различных точек подпрограммы.

Подпрограммы могут находиться в любом месте программы. Часто используемые подпрограммы следует размещать в начале программы, чтобы ускорить ее выполнение. Для перехода к различным подпрограммам в зависимости от значения выражения используется оператор ON...GOSUB.

Например:

```
10 GOSUB 40
20 PRINT "ВОЗВРАТ ИЗ ПОДПРОГРАММЫ" : END
40 PRINT "SUBROUTINE ";
50 PRINT "IN ";
60 PRINT "PROGRESS"
70 RETURN
RUN
SUBROUTINE IN PROGRESS
ВОЗВРАТ ИЗ ПОДПРОГРАММЫ
Ok
```

GOTO – оператор безусловного перехода

GOTO номер строки

Осуществляет безусловный переход из нормальной программной последовательности к определенному номеру строки. Если номер строки – номер исполнительного оператора, то выполняется этот оператор и следующие за ним, а если это неисполнительный оператор (такой, как REM или DATA), то выполнение продолжается с первого исполнительного оператора, встреченного после номера строки.

Оператор GOTO может быть использован в прямом режиме, чтобы заново ввести программу с описываемой точки, например при устранении ошибок.

```
5  DATA 5,7,12
10 READ R
20 PRINT "R="; R;
30 A=3.141*R^2
40 PRINT "AREA=";A
50 GOTO 5
RUN
R= 5       AREA= 78.5
R= 7       AREA= 153.86
R= 12      AREA= 452.16
BHE DATA B 10
Ok
```

Для перехода к различным строкам в зависимости от значения выражения используется оператор ON...GOTO.

HEX$ – функция, предназначенная для воспроизведения строки шестнадцатеричных цифр, десятичный эквивалент которых n задан в качестве аргумента

Y$=HEX$(n)

n – числовое выражение от –32768 до 65535.

Значение n округляется до целого числа перед выполнением функции HEX$. При отрицательном значении n справедливо тождество HEX$(–n)=HEX$(65535–n). Например:

```
10 INPUT X
20 A$=HEX$(X)
30 PRINT X,"DECIMAL IS ",A$,"HEXADECIMAL"
RUN
?32
 32 DECIMAL IS 20 HEXADECIMAL
Ok
RUN
?15
 15 DECIMAL IS F HEXADECIMAL
Ok
```

HIMEM: – оператор установки верхней границы памяти под переменные и строки, используемые программой

HIMEM:n

n – целочисленное выражение.

IF – оператор условного перехода

Формат 1 (для XYBASIC и Бейсик-АГАТ).

```
IF выражение THEN часть
```

часть – оператор или последовательность операторов, или номер строки для перехода.

Формат 2 (для Бейсик-Спектрум+2).

```
IF выражение THEN оператор[:оператор...]

IF выражение THEN GOTO номер строки
```

Формат 3 (для Бейсик-TRS–80, MBASIC, Бейсик-ПК8010, Бейсик-ПК8020, MSX-BASIC и BASICA)

```
IF выражение THEN часть [ELSE часть]

IF выражение GOTO номер строки [ELSE часть]
```

часть – оператор или последовательность операторов, или номер строки для перехода. Принимает решение относительно программного потока, основанного на значении выражения. Если значение выражения – ИСТИНА (не нуль), то выполняется часть THEN (или часть GOTO). Если значение выражения – ЛОЖЬ (нуль), то части THEN и GOTO игнорируются и выполняется часть ELSE, если она существует. Выполнение продолжается со следующего за оператором IF исполнительного оператора.

Операторы IF...THEN...ELSE могут быть вложены. Вложение ограничивается только длиной строки дисплея. Например:

```
IF X>Y  THEN PRINT "БОЛЬШЕ" ELSE IF Y>X THEN PRINT
"МЕНЬШЕ" ELSE PRINT "РАВНЫ"
```

Если оператор содержит неодинаковое количество частей THEN и ELSE, то каждая часть ELSE соответствует части THEN, закрывая ближайший оператор IF. Например:

```
IF A=B  THEN IF  B=C THEN  PRINT "A=C"  ELSE PRINT
"A<>C"
```

Не будет печататься A<>C, когда A<>B.

Если в прямом режиме за оператором IF следует номер строки, то это имеет тот же эффект, как если бы номеру предшествовал оператор GOTO.

IN – функция, определяющая байт, считанный из порта

Формат 1 (для Бейсик-Спектрум+2).

```
Y=IN n
```

n – адрес порта.

Формат 2 (для XYBASIC).

```
Y=IN (n)
```

n – номер порта.

Формат 1. Результат вводится на уровне процессора с порта n (0<=n<=FFFFh). Число n загружается в пару регистров BC и выполняется команда ассемблера IN A, (C). Например:

```
10 FOR n=0 TO 7
20 A=254+256*(255-2^n)
30 PRINT AT 0,0; IN A: GOTO 30
40 NEXT
```

Адреса 254+256*(255–2^n) при значениях n от 0 до 7 используются для опроса клавиатуры (8 полурядов по 5 клавиш).

Формат 2. Система XYBASIC выбирает значение из порта с номером n и присваивает его переменной Y. Номер порта может задаваться выражением. Например:

```
10 Y=IN (12) JOIN IN (13)
20 IF Y=#FFFF THEN 10
30 FOR I=0 TO 15
40 IF TEST (X,I)=0 THEN PRINT "^GBIT #";I]
50 NEXT I
```

INKEY$ – функция, предназначенная для "опроса" клавиатуры

```
Y$=INKEY$
```

Опрашивает клавиатуру на нажатие какой-либо клавиши и вводит символ, клавиша которого нажата. Если никакая клавиша не нажата, то вводится пустая строка.

При использовании INKEY$ символы не высвечиваются на экране, все символы передаются в программу за исключением кода CTRL–C.

Нажатие клавиши возврата каретки при определении ответа, используя INKEY$, будет передавать символ возврата каретки в программу. Необходимо присвоить значение переменной INKEY$ строковой переменной перед использованием символа любым оператором или функцией языка Бейсик. Например:

```
110 PRINT "Нажмите любую клавишу для продолжения"
120 A$=INKEY$ : IF A$="" THEN 120
210 KB$=INKEY$
220 IF LEN(KB$)=2 THEN KB$=RIGHT$(KB$,1)
```

INP – функция, определяющая байт, считанный из порта

```
Y=INP(n)
```

n – номер порта – выражение от 0 до 255.

Функция INP дополнительная к оператору OUT.

Например:

```
150 A=INP(255)
```

INPUT – оператор ввода с клавиатуры

Формат 1 (для XYBASIC).

```
INPUT["напоминание"]список переменных
```

список переменных –имена строковых и/или числовых переменных, разделенные запятыми.

Формат 2 (для Бейсик-Спектрум+2).

```
INPUTпоследовательность элементов
```

Формат 3 (для Бейсик-АГАТ, Бейсик-TRS–80, Бейсик-ПК8010, Бейсик-ПК8020 и MSX-BASIC).

```
INPUT["напоминание";]список переменных
```

список переменных – имена числовых и/или строковых переменных, разделенные запятыми.

Формат 4 (для MBASIC и BASICA).

```
INPUT [;]["напоминание";] список переменных
```

список переменных – имена числовых и/или строковых переменных, разделенные запятыми.

Осуществляет ввод с клавиатуры во время выполнения программы.

Когда выполняется оператор INPUT, программа делает остановку, на экране появляется знак вопроса, указывающий, что программа ожидает данные. Если задана строка напоминания, то она высвечивается перед знаком вопроса (?). Затем требуемые данные вводятся с клавиатуры.

Вводимые данные присваиваются переменным, заданным в списке. Число элементов данных должно соответствовать числу переменных в списке. Тип каждого вводимого элемента должен соответствовать типу, определенному именем переменной. Нет необходимости заключать в кавычки строковые данные.

Формат 1. Если на запрос оператора INPUT вводится больше элементов данных, чем указано в операторе, то появляется сообщение EXCESS IGNORED, лишние данные игнорируются и выполнение программы продолжается.

Если вводится меньше элементов данных или тип вводимых данных не соответствует типу определенных переменных, то появится сообщение REDO? и дается попытка повторить ввод. Так продолжается до тех пор, пока не будет задан правильный ответ.

Формат 2. Элементом оператора INPUT может быть любой из следующих:

элемент оператора PRINT, не начинающийся с буквы (пусто, числовое выражение, строковое выражение, функция управления AT, функция управления TAB);

имя переменной;

LINE (при этом обязательно имя строковой переменной).

Элементы оператора INPUT могут быть разделены запятыми, точками с запятой и апострофами аналогично оператору PRINT.

Все, что выводится с помощью оператора INPUT, помещается в нижней части экрана, которая функционирует в некотором смысле независимо от верхней части экрана.

В отличие от других версий, в версии Бейсик-Спектрум+2 значение переменной может быть выведено на экран в строке напоминания. Если элемент оператора INPUT начинается с буквы, то он должен быть переменной, значение которой необходимо ввести. Однако это ограничение можно обойти заключением соответствующей переменной в скобки. Любое выражение, которое начинается с буквы, заключается в скобки, если оно должно быть напечатано как часть строки напоминания.

Например:

```
LET my age=INT(RND*100):INPUT("I am";my age;".");
"How old are you?",your age
```

Здесь переменная 'my age' (см. гл. 2) заключена в скобки, поэтому ее значение выводится на экран. Переменная 'your age' не заключена в скобки, так что требуется ввести ее значение.

Элементами оператора INPUT могут быть элементы оператора PRINT. Например:

```
10 INPUT  "This is line 1",a$;AT 0,0;"This is line
0",a$
```

Если в операторе INPUT перед именем вводимой строковой переменной поставить ключевое слово LINE в следующем виде:

```
INPUT LINE a$
```

то строковые кавычки не будут выводиться (хотя обычно они выводятся для строковой переменной). Ключевое слово LINE нельзя использовать для числовых переменных, так как ввод воспринимается как строка литералов без кавычек. Механизм STOP (если первый символ при вводе есть STOP, то программа останавливается с ошибкой H) не будет работать. Для останова вместо этого надо нажать клавишу "курсор вниз".

Оператор INPUT LINE работает подобно оператору LINE INPUT в других версиях.

Формат 3. Если на запрос оператора INPUT вводится больше или меньше элементов данных или тип данных не соответствует типу переменной, то на экране дисплея появляется сообщение ?Redo from start (?Повторите ввод). Присваивание введенных значений не выполняется до тех пор, пока не будет задан правильный ответ.

Если на запрос оператора INPUT вводится больше элементов данных, то появится сообщение ?Extra ignored (?Лишние данные), лишние данные игнорируются и выполнение программы продолжается.

Если вводится меньше элементов данных, то до тех пор, пока не будет введено необходимое число данных, будут появляться два знака вопроса (??).

Формат 4. Если сразу после оператора INPUT следует точка с запятой, то во входных данных не появляется символ возврата каретки, определенный пользователем. Например:

```
10 INPUT X
20 PRINT X;" SQUARED IS";X^2_: END
RUN
?5
5 SQUARED IS 25
Ok
```

INSTR — функция, предназначенная для определения позиции строки Y$ в строке X$

```
Z=INSTR([n,]X$,Y$)
```

n — числовое выражение от 1 до 255;

X$, Y$ — строковые переменные, строковые выражения или строковые константы.

Определяет номер позиции начала строки Y$ в строке X$. Необязательное смещение n устанавливает номер позиции для начала просмотра. Функция INSTR определяет 0 в случаях

n>LEN(X$);

X$ — пустая строка;

Y$ — не может быть найдена.

Если Y$ пустая строка, то INSTR определяет n или 1. Если n вне диапазона, возвращается сообщение об ошибке.

Например:

```
10 A$="ABCDEB"
20 B$="B"
30 PRINT INSTR(A$,B$);INSTR(4,A$,B$)
RUN
 2          6
Ok
```

INT – функция определения целой части числового выражения X

```
Y=INT(X)
```

Определяет целое число, которое меньше или равно значению выражения X. Для положительных значений X функция INT работает так же, как функция FIX, т. е. отбрасывает дробную часть числа справа от точки. Для отрицательных значений X функция INT воспроизводит целое число, которое меньше или равно числу слева от точки. Например:

```
PRINT INT(99.88)
 99
Ok
PRINT INT(-12.11)
-13.
Ok
```

**LEFT$** – функция выделения самых левых символов строки X$

```
Y$=LEFT$(X$,n)
```

n – в диапазоне от 0 до 255.

Выделяет n самых левых символов строки X$. Если n больше длины строки, то повторяется вся строка X$. Если n=0, то образуется пустая строка. Например:

```
10 A$="BASIC program"
20 B$=LEFT$(A$,5)
30 PRINT B$
RUN
BASIC
Ok
```

LEN – функция определения длины строки

```
Y=LEN(X$)
```

Вычисляет количество символов в строковой переменной X$. Например:

```
X$="BOCA RATION, FL" : PRINT LEN(X$)
 15
Ok
```

LET – оператор присваивания

```
[LET] переменная = выражения
```

Слово LET необязательно. Например:

```
110 LET D=12          или          110 D=12
120 LET E=12^2                     120 E=12^2
130 LET F$="STROKA"                130 F$="STROKA"
140 LET SUM=(D+E)/5                140 SUM=(D+E)/5
```

Если числовые значения присваиваются строковым переменным или строковые значения числовым переменным, то появляется сообщение об ошибке Type mismath (Ошибочный тип).

LINE INPUT – оператор ввода строковой переменной с клавиатуры

Формат 1 (для Бейсик-TRS–80, Бейсик-ПК8010, Бейсик-ПК8020, MSX-BASIC).

```
LINE INPUT ["напоминание";] строковая переменная
```

напоминание — строковая константа, которая появляется на экране перед вводом данных.

Формат 2 (для MBASIC и BASICA).

```
LINE INPUT [;][“напоминание”;] строковая переменная
```

напоминание — строковая константа, которая появляется на экране перед вводом данных.

Формат 1. Присваивает строку данных строковой переменной без использования разграничителей. Знак вопроса не печатается, если он не является частью строки напоминания. Все введенные данные от конца напоминания до возврата каретки присваиваются строковой переменной. Действие оператора LINE INPUT можно остановить нажатием клавиш CTRL—C. Интерпретатор возвратится на уровень команд, появится подсказка Ok. Для возобновления выполнения программы вводится команда CONT.

Формат 2. Если сразу за LINE INPUT следует точка с запятой, то возврат каретки, введенный пользователем, не делает перевода строки и возврата каретки на экране.

LINPUT — оператор ввода строки символов с клавиатуры

```
LINPUT строковая переменная
```

См. формат 1 оператора LINE INPUT.

LN — функция, вычисляющая натуральный логарифм

```
Y=LN(X)
```

Для вычисления логарифма по любому другому основанию необходимо разделить натуральный логарифм числа на натуральный логарифм основания, т. е. $\log_a x = \ln x / \ln a$.

LOG — функция определения натурального логарифма числа X

```
Y=LOG(X)
```

X — число больше 0.

Например:

```
PRINT LOG(45/7)
 1.860752
Ok
```

Натуральные логарифмы используются во многих задачах, но иногда необходимо вычислить десятичный логарифм. Значение десятичного логарифма можно получить, умножив значение натурального логарифма на 0.43429448. Например:

```
100 LN=LOG(X)
110 LD=LN*0.43429448
```

В этом примере LN — натуральный логарифм числа X, LD — десятичный логарифм числа X.

LOMEM — устанавливает минимальный размер памяти для переменных и строк, используемых программой (LOMEM : n)

LSHIFT — см. ROTATE

MEM — функция определения количества использованных и неиспользованных байтов памяти

```
Y=MEM
```

Эта функция может использоваться в командном режиме для определения количества байтов, которое занимает загруженная программа, или может использоваться в программе для

предотвращения ошибки OM (Out of memory — Вне памяти) переопределением количества байтов, отведенных под строковое пространство, определением массивов меньшей размерности и т. п.

MID$ — функция или оператор выделения требуемой части заданной строки

как функция

```
Y$=MID$(X$,n[,m])
```

как оператор

```
MID$(X$,n[,m])=Y$
```

n — целочисленное выражение от 1 до 255;
m — целочисленное выражение от 0 до 255.

Когда MID$ используется как оператор, то замещается часть одной строки другой строкой. Символы в строке X$, начиная с n-го, замещаются символами строки Y$; m — число символов строки Y$, которые будут замещены. Если m не указано, то используется вся строка Y$.

Длина строки Y$ не изменяется независимо от того, задано число m или нет. Например, если Y$ длиной 4 символа, а Y$ длиной 5 символов, то после замещения X$ будет содержать первые 4 символа строки Y$. Если n или m находятся вне диапазона, то появляется сообщение об ошибке Illegal function call (Неверный функциональный вызов).

Функция воспроизводит строку длиной m символов из строки X$, начиная с n-го символа. Если m не указано или в строке меньше m символов справа от n-го символа, то образуется строка из всех самых правых символов, начиная с n-го. Если m=0 или n>LEN(X$), то MID$ образует пустую строку.

Например:

```
10 A$="GOTO"
20 B$="MORNING EVENING AFTERNOON"
30 PRINT A$,MID$(B$,9,7)
RUN
GOTO EVENING
Ok
```

MSBYTE, LSBYTE и JOIN — функции обработки битов

```
Y=LSBYTE(выражение)

Y=MSBYTE(выражение)

выражение1 JOIN выражение2
```

Функции LSBYTE и MSBYTE определяют самые младшие и самые старшие восемь битов 16-битового целочисленного выражения. Функция JOIN выполняет конкатенацию двух 8-битовых значений в 16-битовое значение.

NULL — команда, определяющая число пустых символов, которые должны быть напечатаны после символа "возврат каретки", т. е. задающая левую границу экрана

```
NULL n
```

n — целое число от 1 до 255.

Например:

```
NULL 3
Ok
```

OCT$ – функция, предназначенная для образования восьмеричного значения десятичного аргумента n

```
Y$=OCT$(n)
```

Перед выполнением значение n округляется до целого числа. Например:

```
PRINT OCT$(24)
30
Ok
```

ON...GOTO, ON...GOSUB – операторы перехода на один из заданных номеров строк в зависимости от вычисленного выражения n

```
ON n GOTO номер строки[,номер строки]...
```

Значение n определяет, какой номер строки в списке будет использован для перехода. Если необходимо, n округляется до целого числа. В операторе ON...GOSUB каждый номер строки в списке должен быть номером первой строки подпрограммы, поэтому необходимо иметь в подпрограмме оператор RETURN, чтобы вернуться в строку, следующую за оператором ON...GOSUB.

Если значение n равно 0 или больше количества строк перехода в программной строке (но меньше или равно 255), то выполнение программы продолжается со следующего исполнительного оператора. Если значение n отрицательно или больше 255, то появляется сообщение об ошибке Illegal fuction call (Неверный функциональный вызов).

Например:

```
100 ON L=1 GOTO 150,300,450,500
110 ON A GOSUB 1300,1400
:
1300 REM start of subroutine for A=1
:
1390 RETURN
```

OUT – оператор посылки данных m в выходной порт машины n

```
OUT n,m
```

n – номер порта от 0 до 255 (0 – 65535 для Бейсик-Спектрум+2);
m – байт данных от 0 до 255.

Оператор OUT является дополнительным к функции INP. Для Бейсик-Спектрум+2 загружает в пару регистров BC число n, в регистр A число m и выполняет команду ассемблера out (C),A.

Например:

```
10 OUT 132,100
```

PEEK – функция получения байта из указанного числом n адреса оперативной памяти

```
Y=PEEK(n)
```

n – целое число от 0 до 65535.

Считанное значение будет целым десятичным числом в диапазоне 0 ... 255. Функция PEEK дополнительная к оператору POKE.

Например:

```
10 IF (PEEK(&H410) AND &H30)=&H30 THEN MONO=1 ELSE
MONO=0
```

PI – число "пи"

```
Y=PI
```

Число "пи" является бесконечной непериодической десятичной дробью, его первые значащие цифры равны 3.1415927.

POKE – оператор записи байта данных m в оперативную память по адресу n

```
POKE n,m
```

n – число от 0 до 65535;

m – число от 0 до 255.

Дополнительной функцией к оператору POKE является функция PEEK; PEEK и POKE полезны для эффективного хранения данных, загрузки подпрограмм на машинном языке, передачи аргументов и результатов в и из подпрограмм на машинном языке. Интерпретатор не делает никакой проверки адресов. Например:

```
10 POKE 106,0
```

POP – оператор удаления из стека последнего адреса возврата

PRINT – оператор вывода информации на экран дисплея

```
PRINT [список выражений][; : ,]
```

Если список выражений не указан, то на экран выводится строка пробелов. Выражения в списке могут быть числовыми и/или строковыми. Строковые константы должны быть заключены в кавычки. Позиция каждого выведенного элемента определяется знаками препинания, используемыми для разделения элементов списка.

В версиях XYBASIC, Бейсик-TRS–80, Бейсик-ПК8010, Бейсик-ПК8020, MSX-BASIC и BASICA строка разделяется на зоны вывода по четырнадцать символов каждая. В версии Бейсик-Спектрум+2 запятая используется, чтобы вывод начинался либо с левой границы, либо с середины экрана, а апостроф (') используется для вывода с начала следующей строки. В версии Бейсик-АГАТ строка делится на три зоны вывода.

Запятая в списке выражений заставляет печататься следующее значение с начала следующей зоны. Точка с запятой заставляет печататься следующее значение со следующей позиции с учетом пробелов при печати числовых данных.

Если запятая или точка с запятой заканчивают список выражений, то следующий оператор PRINT начинает печать в той же строке через заданное число пробелов. Если длина печатаемой строки больше, чем определено оператором WIDTH, то печать продолжается на следующей физической строке.

За печатаемыми числами всегда следует пробел; положительным числам предшествует пробел, а отрицательным знак минус. В некоторых версиях языка числа одинарной точности могут быть представлены шестью или меньше цифрами в формате с фиксированной точкой, но с меньшей точностью, чем если бы они могли быть представлены в формате с плавающей точкой. Они выводятся, используя фиксированную точку или целочисленный формат. Например, 10^(–7) выводится как .0000001, а 10^(–8) выводится как 1E–8.

```
10 INPUT X
20 PRINT X;"SQUARED IS";X^2;"AND",
30 PRINT X;"CUBED IS";X^3
RUN
?3
 9 SQUARED IS 81 AND      9 CUBED IS 729
Ok
RUN
?21
 21 SQUARED IS 441 AND    21 CUBED IS 9261
Ok
10 FOR X=1 TO 5
20 J=J+5
30 K=K+10
40 ?J;K
50 NEXT X
RUN
 5   10   10   20   15   30   20   40   25   50
Ok
```

PRINT USING — оператор вывода информации на экран с учетом заданного формата вывода

```
PRINT USING X$;список выражений[; : ,]
```

Список выражений состоит из строковых или числовых выражений, которые будут печататься. Выражения в списке разделяются запятыми или точками с запятой; X$ — строковая константа или переменная, которая состоит из специальных символов формата (табл. 3.2).

Таблица 3.2

Версия	Специальные символы формата											
	!	\n пробе-лов\	&	#	+	-	**	$$	**$	,	^^^^	_
Стандарт	+		+		+	+						
Бейсик-TRS-80	+		+		+	+	+	+				
MBASIC	+		+		+	+	+	+	+	+	+	+
Бейсик-ПК8010	+		+		+	+	+	+	+	+	+	
Бейсик-ПК8020	+		+		+	+	+	+	+	+	+	
MSX-BASIC	+		+		+	+	+	+	+	+	+	+
BASICA	+		+		+	+	+	+	+	+	+	+

Примечание. В Бейсик-TRS-80 вместо \n пробелов\ используется %n пробелов%; в MSX-BASIC вместо \n пробелов\ используется &n пробелов& и вместо & используется @.

Эти символы определяют формат и поля печатаемых символов. Для печати строк применяются следующие символы формата:

! – первый символ в заданной строке должен быть выведен на экран;

\n пробелов\ – 2+n символов из строки должны быть выведены. Если обратные слеши (\\) не содержат пробелов, то будут выведены два символа, если содержат один пробел, то – три символа и т. д. Если строка длиннее, чем поле, то лишние символы игнорируются. Если поле длиннее, то строка будет справа дополняться пробелами. Например:

```
10 A$="LOOK" : B$="OUT"
30 PRINT USING "!";A$;B$
50 PRINT USING "\ \";A$;B$
70 PRINT USING "\ \";A$;B$;"!!"
RUN
LO
LOOKOUT
LOOK OUT !!
Ok
```

& – определяет поле строки переменной длины. Если поле определено с &, то строка выводится в том виде, как она была введена. Например:

```
10 A$="LOOK" : B$="OUT"
20 PRINT USING "!";A$;
30 PRINT USING "&";B$
40 PRINT "ПЕЧАТЬ & НА ЭКРАНЕ";B$
RUN
LOUT
ПЕЧАТЬ OUT НА ЭКРАНЕ
Ok
```

Для вывода чисел используются следующие символы:

– для представления каждой числовой позиции. Числовые позиции всегда заполнены. Если число, которое будет выводиться, содержит меньше цифр, чем определенных позиций, то число будет выводиться в поле справа и ему будут предшествовать пробелы. Точка может быть выведена в любую позицию поля. Если строка формата определяет, что цифры должны предшествовать точке, то перед точкой обязательно будут выводиться цифры (если необходимо, будут выводиться нули). При необходимости числа округляются. Например:

```
PRINT USING "##.##";.78
 0.78
PRINT USING "###.##";987.654
 987.65
PRINT USING "##.##    ";10.2,5.3,66.789,.234
 10.20    5.30    66.79    0.23
```

+ – знак "+" в конце строки формата используется для вывода знака числа (+ или –) перед или после числа.

– – знак "–" в конце строки формата используется для вывода отрицательного числа со знаком минус. Например:

```
PRINT USING "+##.##    ";-68.95,2.4,55.6,-.9
-68.95    +2.40    +55.60    -0.90
PRINT USING "##.##-    ";68.95,22.449,-7.08
 68.95-    22.45    7.04-
```

** — двойная звезда в начале строки формата для заполнения звездочками передних пробелов в числовом поле; для определения позиций при использовании больше двух цифр. Например:

```
PRINT USING "****.#    ";12.39,-0.9,765.1
*12.4    *-0.9    765.1
```

$$ — двойной знак "$$" для вывода знака $ непосредственно слева от числа; $$ определяет позиции более двух цифр, одна из которых $. Экспоненциальный формат не может быть использован со знаком "$". Отрицательные числа не могут быть использованы, если знак не указан справа. Например:

```
PRINT USING "$$####.##";456.78
$456:78
```

**$ — этот знак в начале строки формата комбинирует действия вышеназванных двух символов формата. Передние пробелы будут заполняться звездочками, а знак "$" будет выводиться перед числом. Знак "**$" определяет позиции более трех цифр, одна из которых $. Например:

```
PRINT USING "**$##.##";2.34
***$2.34
```

, — запятая, которая находится слева от точки в строке формата, используется для вывода запятой слева от каждой третьей цифры слева или справа от точки. При использовании экспоненциального формата запятая не применяется. Например:

```
PRINT USING "####,.##";1234.5
 1,234.5
PRINT USING "####.##,";1234.5
 1234.5,
```

^^^^ — этот знак может быть размещен после символов цифровой позиции для определения экспоненциального формата. Этот знак резервирует пространство для E+nn, которые будут выводиться. Значащие цифры выводятся слева. Если задан начальный или конечный знак "+" или "–", то одна цифровая позиция будет использована слева или справа от точки для вывода пробела или знака "–". Например:

```
PRINT USING "##.##^^^^";234.56
 2.35E+02
PRINT USING ".###^^^^-";-88888
.889E+05-
PRINT USING "+.##^^^^";123
+.12E+03
```

_ — знак подчеркивания в строке формата используется для вывода следующего символа как литерала. Сам литеральный символ может быть знаком подчеркивания, размещаясь как '_' в строке формата. Например:

```
PRINT USING " !##.##_!";12.34
!12.34!
```

Если число, которое должно выводиться, больше определенного числового поля, то в начале числа выводится знак %. Если при округлении число превышает поле, то знак % будет выводиться в начале округленного числа. Например:

```
PRINT USING"##.##";111.22
%111.22
PRINT USING ".##";.999
%1.0
```

Если число заданных цифр превышает 24, то появляется сообщение об ошибке Illegal function call (Неверный функциональный вызов). Например:

```
PRINT USING "THIS IS EXAMPLE ##";1
THIS IS EXAMPLE #1
```

RANDOM – оператор, включающий генератор случайных чисел и определяющий последовательность случайных величин при использовании функции RND

RANDOMIZE – оператор, обеспечивающий работу генератора случайных чисел

RANDOMIZE [n]

n – любое целое число в диапазоне чисел, с которыми работает используемая версия языка Бейсик.

Если n не указано, то интерпретатор приостанавливает выполнение программы и запрашивает значение; на экране появляется сообщение Random number seed (–32768 to 32767)? (Введите любое число (от –32768 до +32767)?) перед выполнением RANDOMIZE.

Чтобы изменить последовательность случайных чисел при каждом выполнении программы, необходимо поместить оператор RANDOMIZE в начале программы и каждый раз изменять число n.

READ – оператор считывания данных из оператора DATA

READ список переменных

Считывая значения из оператора DATA, присваивает их переменным. Оператор READ всегда должен быть использован в паре с оператором DATA. Оператор READ присваивает значение оператора DATA переменным один к одному. Например:

```
10 PRINT "СТРАНА","ГОРОД","ИНДЕКС"
20 READ C$,S$,Z
30 DATA "СССР","МОСКВА",115470
40 PRINT C$,S$,Z
RUN
СТРАНА          ГОРОД         ИНДЕКС
СССР            МОСКВА         115470
Ok
```

RESET и **SET** – функция установки в 0 или в 1 соответственно определенного бита в байте

Y=RESET(выражение1,выражение2)

Y=SET(выражение1,выражение2)

выражение1 – целочисленное выражение, значение которого изменяется;

выражение2 – номер бита (от 0 до 15).

Например:

```
10 X=16:Y=SET(0,4)
20 PRINT RESET(X,4);Y
RUN
 0   16
OK
```

В функциях SET и RESET выражение2 вычисляется по модулю 16.

RESTORE – оператор, используемый совместно с операторами READ и DATA при вводе данных

Формат 1 (кроме Бейсик-АГАТ).

RESTORE [номер строки]

Формат 2 (для Бейсик-АГАТ).

RESTORE

Разрешает данным, указанным оператором DATA, повторно считываться с определенной строки. После того как выполнится оператор RESTORE, следующий оператор READ имеет доступ к первому элементу первого оператора DATA программы. Например:

```
10 READ A,B,C
20 RESTORE
30 READ D,E,F
40 DATA 57,68,49
50 PRINT A,B,C,D,E,F
RUN
 57      68      49      57      68      49
Ok
```

Если номер строки задан, то следующий оператор READ имеет доступ к первому элементу в заданном операторе DATA.

RETURN – оператор возврата управления из подпрограммы в вызывающую программу.

Формат 1 (кроме MSX-BASIC и BASICA).

RETURN

Формат 2 (для MSX-BASIC и BASICA).

RETURN[номер строки]

Осуществляет возврат к оператору, следующему за оператором перехода к подпрограмме. В версиях MSX-BASIC и BASICA может быть осуществлен возврат по указанному номеру строки. Например:

```
50 GOSUB 400
:
400 REM subroutine
:
500 RETURN
```

**RIGHT$** – функция выделения n самых правых символов строки X$

Y$=RIGHT$(X$,n)

n – число от 0 до 255.

Если n больше или равно длине строки X$, то печатается вся строка X$. Если n равно 0, то печатается пустая строка.

Например:

```
10 A$="BOCA RATON,ABCDEFG"
20 PRINT RIGHT$(A$,7)
RUN
ABCDEFG
Ok
```

RND – функция генерации псевдослучайных чисел от 0 до 1

Формат 1 (кроме Бейсик-АГАТ).

```
Y=RND[(n)]
```

Формат 2 (для Бейсик-АГАТ).

```
Y=RND (n)
```

Например:

```
10 FOR I=1 TO 3
20 PRINT RND(I),
30 NEXT I
RUN
 .6291626     .1948297     .6305799
Ok
```

При выполнении программы каждый раз генерируется одинаковая последовательность случайных чисел, если не выбран генератор случайных чисел. Это можно сделать, используя оператор RANDOMIZE, или в функции RND задавать аргумент n каждый раз с другим значением; RND(0) повторяет последнее сгенерированное случайное число.

ROTATE, RSHIFT и LSHIFT – функции сдвига байта

```
Y=ROTATE(выражение1,выражение2)

Y=RSHIFT(выражение1,выражение2)

Y=LSHIFT(выражение1,выражение2)
```

выражение1 – целочисленное выражение, значение которого сдвигается;
выражение2 – число позиций.

Функция ROTATE осуществляет циклический сдвиг вправо, а функции RSHIFT и LSHIFT осуществляют сдвиг целочисленных значений, заданных в выражении1, вправо и влево на заданное число позиций соответственно.

Например:

```
PRINT ROTATE (1,2)
 16384
OK
PRINT LSHIFT (5,3)
 40
OK
```

В первом примере циклически сдвигается вправо число 1 на две позиции и получается число 16384 (#4000). Во втором примере число 5 (двоичное &0101) сдвигается влево на три позиции и получается число 40 (двоичное &101000).

SCALL – оператор вызова подпрограммы на машинном языке

```
SCALL число[,целочисленная переменная1,целочисленная
      переменная2,целочисленная переменная3]
```

число – адрес подпрограммы.

Вызывает подпрограмму на машинном языке по адресу, заданному числом. Если необязательные переменные присутствуют, то их значения передаются в регистры BC, DE и HL, а при возврате из машинной подпрограммы значения из этих регистров присваиваются соответствующим переменным.

Сообщение об ошибке MC появляется, если определено более трех переменных или если переменные не являются целочисленными.

Например:

```
SCALL #7400,X%
```

SENSE – функция определения значения бита в порте

```
Y=SENSE(номер бита,номер порта)
```

Например:

```
10 FOR I=0 TO 7
20 PRINT SENSE(5,I)
30 NEXT I
RUN
 1  0  1  1  0  0  0  0
OK
```

SGN – функция определения знака числа

```
Y=SGN(X)
```

Если аргумент X – положительное число, то функция определяет +1, если равен 0, то –0, если отрицательное число, то –1.

Например:

```
10 ON SGN(X)+2 GOTO 100,200,300
```

SIN – функция определения синуса угла X, заданного в радианах

```
Y=SIN(X)
```

Функция SIN(X) определяет вещественное число одинарной точности.

Например:

```
PRINT SIN(1.5)
 .09974951
Ok
```

SPACE$ – функция образования строки пробелов

```
Y$=SPACE$(n)
```

n – число от 0 до 255.

Например:

```
10 FOR I=1 TO 5
20 X$=SPACE$(I)
30 PRINT X$;I
40 NEXT I
RUN
 1
   2
      3
         4
            5
Ok
```

SPC – функция, используемая совместно с оператором PRINT для печати n пробелов

```
PRINT SPC(n)
```

n – число от 0 до 255.

Если n больше заданной ширины экрана, то используется значение (n MOD ширина). Функция SPC может быть использована только с операторами PRINT, LPRINT, PRINT#.

Например:

```
PRINT "OVER";SPC(15);"THERE"
OVER               THERE
Ok
```

SQR – функция извлечения квадратного корня неотрицательного числа X

```
Y=SQR(X)
```

Например:

```
10 FOR X=10 TO 25 STEP 5
20 PRINT X,SQR(X)
30 NEXT
RUN
 10        3.162278
 15        3.872984
 20        4.472136
 25        5
Ok
```

STR$ – функция формирования строки десятичных цифр. Образует строковое представление значения X

```
X$=STR$(X)
```

Функция VAL является инверсией функции STR$.

Например:

```
10 INPUT "NUMBER";N
20 ON LEN(STR$(N)) GOSUB 60,100,200
```

**STRING$** – функция повторения определенное число n раз заданного символа с кодом m или первого символа строки X$

```
Y$=STRING$(n,m)

Y$=STRING$(n,X$)
```

n и m – числа от 0 до 255.

Например:

```
10 X$=STRING$(10,45)
20 PRINT X$;"MONTHLY REPORT";X$
RUN
----------MONTHLY REPORT----------
Ok
10 X$=(ABCD)
20 Y$=STRING$(10,X$)
30 PRINT Y$
RUN
AAAAAAAAAA
Ok
```

SWAP – оператор обмена значений двух переменных

```
SWAP переменная1,переменная2
```

Могут быть обменены переменные любого типа, но обе переменные должны быть одного типа, иначе появляется сообщение Type Mismatch (Ошибочный тип). Переменными могут быть элементы массива.

Например:

```
10 A$="ONE": B$="ALL": C$="FOR ":PRINT A$;C$;B$
20 SWAP A$,B$:PRINT A$,C$,B$
RUN
ONE FOR ALL
ALL FOR ONE
Ok
```

TAB – функция табуляции в n-ю позицию. Записывается совместно с оператором PRINT

```
PRINT TAB(n)
```

n – число от 1 до 255.

Если номер позиции, в которой производится печать текущего символа больше n, то заданная табуляция отрабатывается на следующей строке. Позиция 1 – самая левая заданная ширина минус 1 (WIDTH–1) – самая правая позиция. Функция TAB может использоваться только с операторами PRINT, LPRINT, PRINT#.

Например:

```
10 PRINT "ИМЯ"TAB(25)"КОЛИЧЕСТВО"
20 READ A$,B$
30 PRINT A$TAB(25)B$
40 DATA "Л.М.Яншин","25.68"
RUN
ИМЯ                       КОЛИЧЕСТВО
Л.М.Яншин                 25.68
Ok
```

TAN – функция определения тангенса угла X, заданного в радианах

```
Y=TAN(X)
```

Функция TAN вычисляется с одинарной точностью.

TEST – функция определения значения бита

```
Y=TEST(выражение1,выражение2)
```

Проверяет бит в целочисленном значении, устанавливая его значение (0 или 1). Первый аргумент функции TEST – это переменная или выражение, которое необходимо проверить, второй – номер бита, который необходимо проверить.

Например

```
10 I=7
20 PRINT TEST(I,2)
 1
OK
```

USR – функция, используемая для вызова подпрограмм, написанных на машинном языке с аргументом X

Формат 1 (для Бейсик-Спектрум+2 и Бейсик-АГАТ).

```
Y=USR(X)
```

Формат 2 (для всех остальных версий).

```
Y=USR[цифра](X)
```

цифра – 0–9. Соответствует цифре в операторе DEF USR для заданной подпрограммы. Если цифра не указана, то принимается USR0.

X – адрес начала подпрограммы на машинном языке.

Формат 1. В Бейсик-Спектрум+2 результатом выполнения будет значение, лежащее в регистрах BC. Подпрограмму в машинных кодах можно сохранять командой SAVE "имя" CODE адрес (см. гл. 4).

Например:

```
PRINT USR 65268
```

В Бейсик-АГАТ значение X помещается в ячейках 157 – 163. Адрес подпрограммы на машинном языке должен быть подготовлен в ячейках 11 – 12, а в ячейке 10 должен быть код 76.

Формат 2. Оператор CALL другим способом вызывает подпрограмму на машинном языке.

Например:

```
10 B=T*SIN(Y)
20 C=USR(B/2)
30 D=USR(B/3)
```

VAL – функция определения числового значения строки X$

```
X=VAL(X$)
```

функция VAL при вычислении результата убирает начальные пробелы, табуляцию и перевод строки. Если X$ не число, то результатом будет 0.

Например:

```
PRINT VAL("        -3")
-3
PRINT VAL("PROBA")
 0
```

**VAL$** – функция обработки строки X$ (без ограничивающих кавычек) как строкового выражения

```
Y$=VAL$(X$)
```

Если аргумент функции содержит синтаксическую ошибку или дает числовое значение, то появляется сообщение об ошибке Nonsense in BASIC (Ерунда в Бейсике).

Следует помнить, что (в Бейсик-Спектрум+2) внутри строки символ кавычек должен быть введен дважды. Для более глубокого вложения строк кавычки потребуется повторять или четыре или восемь раз.

Для того чтобы разобраться, как работает функция VAL$, необходимо разобраться, как работает функция VAL.

Функция VAL выполняется в два приема: сначала ее аргумент вычисляется как строка, затем кавычки отбрасываются и оставшаяся величина вычисляется как число.

Для функции VAL$ первый шаг тот же, однако после отбрасывания кавычек оставшаяся величина снова вычисляется как строка. Например:

```
PRINT VAL$(""""STROKA""""")
"STROKA"
```

VARPTR – функция определения адреса памяти, где хранится заданная переменная

X=VARPTR(переменная)

Возвращает адрес первого байта данных, идентифицированных переменной. Перед использованием функции VARPTR значение должно быть присвоено переменной, иначе появится сообщение об ошибке Illegal function call (Неверный функциональный вызов). Может быть использовано имя переменной любого типа.

Функция VARPTR обычно используется для получения адреса переменной или массива для передачи его в подпрограмму на машинном языке. Например:

```
100 FR=VARPTR(A%)
110 PRINT PEEK(FR);PEEK(FR+1)
```

WAIT – оператор приостановки выполнения программы на период обработки состояния входного порта компьютера

WAIT порт,n[,m]

порт – номер порта от 0 до 255.

Приостанавливает выполнение программы, пока заданный порт машины не выработает определенного образа бит. Данные, считанные в порт, подвергаются операции XOR с целочисленным значением m, а затем операции AND с выражением n. Если результат равен 0, то выполнение продолжается со следующего оператора. Если m не указано, то принимается m, равное 0.

Например:

```
100 WAIT 32,2
```

WHILE...WEND – оператор цикла

WHILE выражение

операторы внутри цикла

WEND

Выполняет последовательность операторов в цикле до тех пор, пока заданное условие истинно (не 0). Если выражение истинно, операторы цикла выполняются до тех пор, пока не встретится оператор WEND. Затем управление передается оператору WHILE и проверяется выражение. Если заданное условие ложно, то выполнение программы продолжается с оператора, следующего за оператором WEND. Цикл WHILE...WEND может быть вложенным до любого уровня. Каждый WEND будет соответствовать последнему незакрытому оператору WHILE. Незакрытый цикл вызовет ошибку WHILE without WEND (WHILE без WEND).

Например:

```
10 FL=1
20 WHILE FL
30 FL=0
40 FOR I=1 TO 5:PRINT I:FL=1:NEXT I
50 WEND
```

WIDTH – оператор, определяющий ширину строки на экране дисплея или на печатающем устройстве

Формат 1 (кроме MBASIC и BASICA).

```
WIDTH размер
```

Формат 2 (для MBASIC и BASICA).

```
WIDTH [LPRINT] размер
```

размер – числовое выражение от 15 до 255.

Если задан параметр LPRINT, то для печатающего устройства будет устанавливаться новая ширина строки. Если **размер** равен 255, то интерпретатор никогда не будет вставлять возврат каретки. Однако позиция курсора или головки печатающего устройства, заданная функциями POS или LPOS, будет равна 0.

WRITE – оператор вывода информации на экран

```
WRITE [список выражений]
```

Если список выражений не указан, то выводится строка пробелов. А если задан список выражений, то их значения выводятся на экран дисплея. Выражения в списке могут быть числовыми и/или строковыми. Они отделяются друг от друга запятыми или точками с запятой. При выводе каждый печатаемый элемент будет отделяться от другого запятой.

Печатаемые строки будут заключаться в кавычки. После печати последнего элемента списка интерпретатор вводит возврат каретки/перевод строки. Различие между WRITE и PRINT в том, что WRITE вставляет запятые между выводимыми элементами, заключает строковые переменные в кавычки и положительным числам не предшествуют пробелы.

Например:

```
10 A=80:B=90:C$="THAT'S ALL"
20 WRITE A,B,C$
RUN
80,90,"THAT'S ALL"
Ok
```

Глава 4

Работа с внешними устройствами

4.1. Файловая организация

4.1.1. Основные определения

Запись – совокупность данных на устройстве внешней памяти, являющаяся элементарным объектом при единичном обращении к данным.

Спецификация – имя, однозначно идентифицирующее файл, обрабатываемый в любой операционной системе. Спецификация файла обычно состоит из трех компонентов: имени устройства, имени файла, расширения имени файла.

В табл. 4.1 представлена информация о синтаксисе этих компонентов в рассматриваемых версиях интерпретаторов языка Бейсик.

Таблица 4.1

Версия	Имя устройства	Имя файла	Расширение имени файла
XYBASIC	A:,B:,C:,D: или @:	до 8 символов или цифр	от 0 до 3 символов или цифр
Бейсик-Спектрум+2		до 10 символов	
Бейсик-АГАТ	1,2	от 1 до 30 символов или цифр	
MBASIC	A:,B: и т.д.	до 8 символов	от 0 до 3 символов
Бейсик-ПК8020	То же	То же	То же
MSX-BASIC	—||—	—||—	—||—
BASICA	—||—	—||—	—||—

Файл – организованный определенным образом набор связанных записей данных.

Файл данных – файл последовательного или произвольного доступа, создаваемый программистом для последующей обработки в программе.

Программный файл – программа пользователя, написанная на языке Бейсик или на машинном языке и хранящаяся на внешнем носителе.

К программным файлам также можно отнести и программы, написанные на каком-либо машинном языке, и которые использует в своей работе Бейсик-программа.

Бейсик-программа может храниться в текстовом, внутреннем или закодированном внутреннем формате. *Внутренный формат* зависит от конкретной версии интерпретатора языка Бейсик (пример внутреннего представления программ в MBASIC дан в приложении). *Внутренний закодированный формат* – это специально подобранная система кодировки программы, которая при загрузке программного файла не позволяет вносить в него изменения и распечатывать его текст. *Текстовый формат* Бейсик программы – это набор кодов ASCII (КОИ-8). Этот формат позволяет набирать тексты программ на языке Бейсик, пользуясь любыми текстовыми редакторами.

В большинстве версий языка Бейсик, работающих под управлением той или иной операционной системы, доступ может осуществляться не к одному дисководу, а к нескольким. Например, операционная система МикроДОС может поддерживать до 16 дисководов, если эти устройства подключены и инициализированы. Соответственно Бейсик также может различать эти 16 дисководов.

Как уже было сказано выше, любой файл определяется спецификацией файла. Однако в таких версиях, как XYBASIC, MBASIC, Бейсик-ПК8020, MSX-BASIC, BASICA в операторах обмена с файлом большую роль играет номер файла.

Номер файла – числовая константа, присваиваемая файлу при операции обмена с ним. Номер файла идентифицирует текущий файл. Одновременно может быть задействовано несколько файлов. Обращение к ним осуществляется по номеру. В вышеперечисленных версиях одновременно могут использоваться до 16 файлов.

Признаком номера файла является символ #. В XYBASIC признаком номера файла является символ @.

4.1.2. Общие файловые операторы и команды

Почти во всех версиях языка Бейсик существуют инструкции, которые могут применяться при работе как с файлами данных, так и с программными файлами (рис. 4.1 и 4.2). Они представлены в табл. 4.2.

Таблица 4.2

Инструкция	Номер версии									
	1	2	3	4	5	6	7	8	9	10
CAT!			+							
CATALOG				+						
CLEAR		+								
DELETE				+						
DIR		+								
DSKF									+	
ERASE			+							
ERASE!			+							
FILES	+				+	+		+	+	+
KILL					+	+		+	+	+
LFILES					+	+		+	+	+
NAME					+	+		+	+	+
RENAME				+						
SCRATCH	+									

CAT! – команда выдачи списка всех программ или файлов данных каталога, записанных на "электронном" диске

"Электронный" диск – это область ОЗУ, организованная как накопитель на гибком магнитном диске.

CAT!

Описание см. команду FILES. (Формат FILES*.*)

CATALOG – команда выдачи списка всех программ или файлов, находящихся на диске

CATALOG устройство

устройство – диск, каталог которого необходимо вывести на экран дисплея

Описание см. команду FILES.

CLEAR – команда определения количества одновременно используемых файлов на диске

CLEAR n

n – количество файлов, определяемое пользователем.

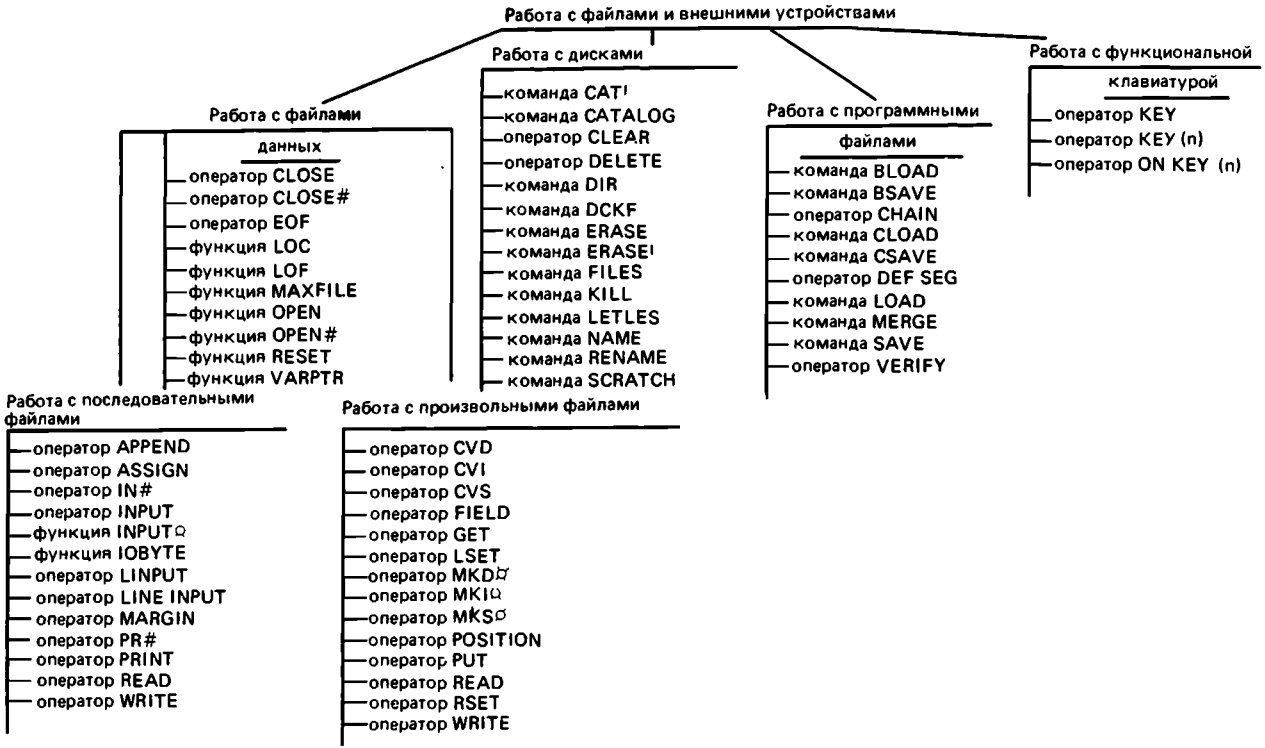

Работа с файлами и внешними устройствами
Работа с файлами
данных
оператор CLOSE
оператор CLOSE#
оператор EOF
функция LOC
функция LOF
функция MAXFILE
функция OPEN
функция OPEN#
функция RESET
функция VARPTR
Работа с последовательными файлами
оператор APPEND
оператор ASSIGN
оператор IN#
оператор INPUT
функция INPUT$
функция IOBYTE
оператор LINPUT
оператор LINE INPUT
оператор MARGIN
оператор PR#
оператор PRINT
оператор READ
оператор WRITE
Работа с произвольными файлами
оператор CVD
оператор CVI
оператор CVS
оператор FIELD
оператор GET
оператор LSET
оператор MKD$
оператор MKI$
оператор MKS$
оператор POSITION
оператор PUT
оператор READ
оператор RSET
оператор WRITE
Работа с дисками
команда CAT!
команда CATALOG
оператор CLEAR
оператор DELETE
команда DIR
команда DSKF
команда ERASE
команда ERASE!
команда FILES
команда KILL
команда LETLES
команда NAME
команда RENAME
команда SCRATCH
Работа с программными файлами
команда BLOAD
команда BSAVE
оператор CHAIN
команда CLOAD
команда CSAVE
оператор DEF SEG
команда LOAD
команда MERGE
команда SAVE
оператор VERIFY
Работа с функциональной клавиатурой
оператор KEY
оператор KEY (n)
оператор ON KEY (n)

Рис. 4.1

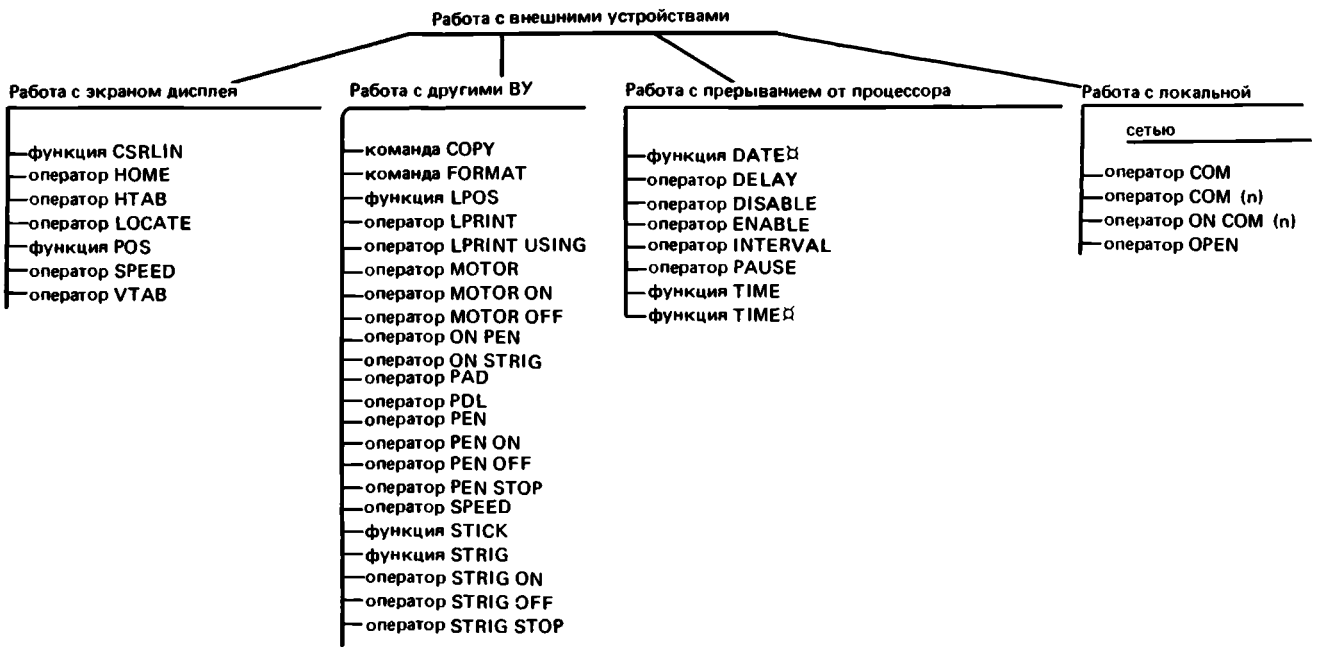

Работа с внешними устройствами
Работа с экраном дисплея
функция CSRLIN
оператор HOME
оператор HTAB
оператор LOCATE
функция POS
оператор SPEED
оператор VTAB
Работа с другими ВУ
команда COPY
команда FORMAT
функция LPOS
оператор LPRINT
оператор LPRINT USING
оператор MOTOR
оператор MOTOR ON
оператор MOTOR OFF
оператор ON PEN
оператор ON STRIG
оператор PAD
оператор PDL
оператор PEN
оператор PEN ON
оператор PEN OFF
оператор PEN STOP
оператор SPEED
функция STICK
функция STRIG
оператор STRIG ON
оператор STRIG OFF
оператор STRIG STOP
Работа с прерыванием от процессора
функция DATE¤
оператор DELAY
оператор DISABLE
оператор ENABLE
оператор INTERVAL
оператор PAUSE
функция TIME
функция TIME¤
Работа с локальной
сетью
оператор COM
оператор COM (n)
оператор ON COM (n)
оператор OPEN

Рис. 4.2

По умолчанию в XYBASIC в начале работы пользователь может работать с двумя файлами. Однако, если необходимо увеличить количество файлов, он должен выполнить команду CLEAR. Но надо помнить, что каждый открытый для записи или чтения файл занимает 166 байт памяти. Одновременное использование большого количества файлов может вызвать появление ошибки OM – Out of memory (Вне памяти).

Использование команды CLEAR для указания количества файлов не влияет на действие этой же команды при определении емкости памяти, выделяемой под строковое пространство. Строковое пространство, определенное в какой-либо ранее заданной команде CLEAR, остается неизменным. Но при этом все переменные, определенные ранее, очищаются.

Например:

```
CLEAR @3
```

По этой команде пользователю разрешается использовать в своих файловых операциях до трех файлов.

DELETE – команда удаления файла с заданным именем на диске

```
DELETE имя файла[,устройство]
```

имя файла – имя удаляемого файла;

устройство – обозначение диска, с которого удаляется файл.

Описание см. команду KILL.

Например:

```
DELETE "PROGRAMM.BAS"
```

Удаляет на текущем диске файл PROGRAMM.BAS.

DIR – команда выдачи на экран дисплея имен файлов или программ, записанных на диске

```
DIR ["имя файла"]
```

Описание см. команду FILES.

DSKF – функция определения свободной памяти на диске

```
DSKF (номер диска)
```

Определяет число оставшихся блоков памяти на диске, указанном в команде. Параметр **номер диска** может принимать значения 0, 1 или 2, где 0 – активизированный (текущий) диск; 1 – диск A; 2 – диск B.

ERASE – команда удаления файлов на диске

```
ERASE
```

Описание см. команду KILL.

ERASE! – команда удаления файла на "электронном" диске

```
ERASE! "имя файла"
```

Описание см. команду KILL.

FILES – команда вывода на экран дисплея каталога диска

```
FILES ["спецификация файла"]
```

Команда FILES выводит на экран дисплея имена всех файлов, которые существуют на диске. В спецификации файла имя файла может быть неявным, т. е. содержать на месте расширения файла или имени файла символы * или ?. В таком случае на экран будут выводиться не все файлы, а только указанные.

Например:

```
FILES "PROG.*"
PROG.BAS
PROG.BG
PROG.BSC
Ok
```

Если в спецификации файла не указан дисковод, то считывается каталог текущего диска. Если же спецификация файла не указана, то на экран выводятся имена всех файлов, находящихся на диске.

KILL – команда удаления заданного файла на диске

```
KILL "спецификация файла"
```

Может быть удален файл любого типа, но если обозначенный файл является открытым, то выдается сообщение File already open (Файл уже открыт) и удаления не происходит. Имя должно быть явно указано, т. е. не должно содержать символов * или ?. Если указанного файла на диске нет, то выдается сообщение File not found (Файл не найден). Команда KILL не удаляет также файлы, помеченные как системные, или файлы, имеющие атрибут Read only (только для чтения).

Например:

```
KILL "PROG.*"
```

Это означает, что с диска будут удалены все файлы, имеющие имя PROG с произвольным расширением.

LFILES – команда вывода каталога диска на печатающее устройство

```
LFILES ["спецификация файла"]
```

Формат и действие аналогично команде FILES, только вывод каталога осуществляется на печатающее устройство. Если печатающее устройство не включено, то выдается сообщение Device I/O error (Ошибка устройства ввода-вывода).

NAME – команда переименовывания файлов на диске

```
NAME "спецификация файла1" AS "спецификация файла2"
```

спецификация файла1 – имя файла, который подлежит переименованию;

спецификация файла2 – имя, которое будет дано файлу после переименования.

При операции переименовывания старое имя файла, определенное в спецификации файла1, должно существовать на диске, иначе будет выдаваться сообщение об ошибке File not found (Файл не найден). Нового имени файла на диске существовать не должно. Если в имени файла не указано устройство, то по умолчанию принимается текущий дисковод. После действия команды NAME файл под новым именем располагается на заданном диске в той же области дискового пространства.

RENAME – команда переименовывания файлов на диске

```
RENAME старое имя файла,новое имя файла[,устройство]
```

Описание см. команду NAME.

SCRATCH – команда удаления заданного файла на диске

```
SCRATCH "имя файла"
```

Описание см. команду KILL.

Например:

```
SCRATCH "TEMP.DAT"
```

Удаляется файл TEMP.DAT с текущего диска.

4.1.3. Команды и операторы, используемые при работе с файлами последовательного и произвольного доступа

Во многих версиях языка Бейсик существуют инструкции, которые можно использовать как с файлами последовательного, так и произвольного доступа. Их перечень и наличие в разных версиях приведены в табл. 4.3.

Таблица 4.3

Инструкция	Номер версии									
	1	2	3	4	5	6	7	8	9	10
CLOSE	+	+		+	+	+		+	+	+
CLOSE#			+							
EOF	+	+			+	+		+	+	+
LOC					+	+		+	+	+
LOF					+	+		+	+	+
MAXFILES									+	
OPEN	+	+		+	+	+		+	+	+
OPEN#			+							
RESET						+				
VARPTR					+	+		+	+	+

CLOSE – оператор закрытия файла на диске

Формат 1 (для Бейсик-ПК8020, BASICA, MSX-BASIC, MBASIC).

```
CLOSE [[#]номер файла[,#номер файла...]]
```

Формат 2 (для XYBASIC).

```
CLOSE [[@]номер файла[,@номер файла...]]
```

Завершает операции ввода-вывода с файлами и освобождает соответствующие буферы. Оператор CLOSE без указания операндов закрывает все файлы, открытые оператором OPEN. После выполнения CLOSE аннулируется условная связь между определенным файлом или устройством и его номером. Этот же файл или устройство могут быть опять открыты с тем же или другим номером файла. (Оператор END, команды NEW, RESET или SYSTEM автоматически закрывают все открытые файлы или устройства. Оператор STOP не закрывает файлов.)

Например:

```
100 CLOSE 1,#2,#3
```

Закрываются указанные файлы.

CLOSE# – оператор закрытия файла на диске

```
CLOSE#номер файла,номер файла...
```

Описание см. оператор CLOSE.

EOF – функция признака конца файла

```
Y=EOF(номер файла)
```

номер файла – номер, определенный в операторе OPEN.

Проверяет признак конца файла при операции чтения. Значение функции равно –1 (ИСТИНА), если найден конец файла, и равно 0 (ЛОЖЬ), если конец файла не найден. Например:

```
10 OPEN "I",#1,"DATA":C=0
20 IF EOF(1) THEN 100
30 INPUT #1,M(C)
40 C=C+1:GOTO 20
100 REM обработка конца файла
```

LOC – функция определения текущей позиции в файле

```
Y=LOC(номер файла)
```

При работе с файлами произвольного доступа LOC определяет номер последней записи, считанной или записанной в файл произвольного доступа. При работе с файлами с последовательным доступом LOC определяет число записей, считанных или записанных в файл с того момента, как он был открыт.

Когда файл открыт для последовательного ввода, интерпретатор считывает первый сектор файла, поэтому значение LOC будет равно 1.

Например:

```
10 IF LOC(1)>50 THEN STOP
20 PUT #1,LOC(1)
```

В строке 20 происходит перезапись считанной записи.

LOF – функция определения длины файла

```
Y=LOF(номер файла)
```

номер файла – номер, заданный в операторе OPEN.

Определяет число байтов, размещенных в файле (длина файла). Число, определяемое функцией LOF, кратно 128. Например, если действительные данные в файле занимают 257 байт, то число байтов, определенных в функции, будет равно 384.

Например:

```
10 OPEN "O",#1,"BIG"
20 GET #1,LOF(1)/128
```

MAXFILES – оператор резервирования определенного числа буферов управления файлом

```
MAXFILES=арифметическое выражение
```

арифметическое выражение – выражение, принимающее значения от 0 до 15.

Оператор используется для установления максимального числа одновременно открытых файлов и соответственно числа зарезервированных буферов управления файлом данных. Каждый буфер управления файлом требует 257 байт памяти.

Если параметр **арифметическое выражение** больше 15, то выдается сообщение "Illegal function call" (Неверный функциональный вызов), если не хватает памяти, то Out of memory (Вне памяти).

Например:

```
10 MAXFILES=2
```

OPEN – оператор открытия файла

Формат 1 (для Бейсик-ПК8020, MBASIC, BASICA).

```
OPEN "режим1",[#]номер файла,"имя файла"
[,длина записи]
```

Формат 2 (для XYBASIC).

```
OPEN режим2, @номер файла,"имя файла"
```

Формат 3 (для Бейсик-АГАТ).

```
OPEN имя файла[[,длина записи][,устройство]]
```

Формат 4 (для MSX-BASIC и BASICA).

```
OPEN "имя файла"[FOR "режим4"] AS [#] номер файла
[LEN=длина записи]
```

режим1 – строковое выражение, первый символ которого:
O – последовательный вывод в файл;
I – последовательный ввод из файла;
R – произвольный ввод-вывод.

режим2 – строковое выражение, первый символ которого:
O – последовательный вывод в файл;
I – последовательный ввод из файла;
U – добавление к последовательному файлу.

длина записи – числовое выражение, определяющее длину записи для файла произвольного доступа.

режим4 – один из нижепредставленного набора строк:
INPUT – последовательный ввод из файла;
OUTPUT – последовательный вывод в файл;
APPEND – добавление к последовательному файлу;

номер файла – целочисленное выражение, значение которого лежит в диапазоне от одного до максимального количества файлов; по умолчанию равно 3. Может быть изменено параметром /F: при загрузке интерпретатора.

Форматы 1–4. Открывают буфер для ввода-вывода в файл или устройство и определяют режим чтения или записи.

На НГМД могут размещаться файлы как последовательного, так и произвольного методов доступа; на других внешних устройствах памяти могут размещаться только файлы с последовательным методом доступа.

Формат 3. Оператору OPEN должен предшествовать вывод кода CONTROL–D:

```
PRINT CHR$(4);"OPEN MYFILE,n"
```

n – длина записи.

Если параметр длина записи указан, то определяемый файл будет произвольного доступа, если не указан – то последовательного.

Формат 4.

Например:

```
10 OPEN "DATA" FOR "INPUT" AS #1
```

Файл DATA открывается для последовательного ввода информации.

OPEN# – оператор открытия файла

```
OPEN# имя файла
```

Описание см. оператор OPEN.

RESET – оператор закрытия файлов

```
RESET
```

Закрывает все файлы и очищает системный буфер. Если все открытые файлы находятся на диске, то RESET действует так же, как и команда CLOSE без номера файла.

VARPTR – функция определения адреса блока управления файлом

```
X=VARPTR(#номер файла)
```

Определяемый адрес лежит в диапазоне от 0 до 65535. Файл должен быть предварительно открыт. Например:

```
10 OPEN "O",#1,"DATA"
20 FC=VARPTR(#1)
30 DA=FC+188
40 A$=PEEK(DA)
```

4.1.4. Файлы с последовательным доступом

Файлы с последовательным доступом создаются легче, чем файлы произвольного доступа, но они уменьшают гибкость и скорость доступа к данным. Данные, которые записываются в файл с последовательным доступом, запоминаются один за другим (последовательно) и также считываются в коде ASCII (КОИ-8).

Для работы с файлами последовательного доступа используются следующие инструкции, представленные в табл. 4.4.

Таблица 4.4

Инструкция	Номер версии									
	1	2	3	4	5	6	7	8	9	10
APPEND				+						
ASSIGN		+								
IN#				+						
INPUT	+	+	+		+	+		+	+	+
INPUT$					+	+		+	+	+
IOBYTE		+								
LINPUT		+								
LINE INPUT	+				+	+		+	+	+
MARGIN		+								
PR#				+						
PRINT	+	+	+		+	+		+	+	+
PRINT USING	+				+	+		+	+	+
READ				+						
WRITE				+						

APPEND – команда подсоединения данных к файлу

```
APPEND "имя файла"
```

Команда открывает файл и устанавливает указатель записи в конец файла. Данные, дополнительно записываемые в файл, будут вводиться только после последнего элемента в файле. Если после задания команды APPEND производится операция чтения, то это приводит к сообщению об ошибке. Команда APPEND не должна следовать за оператором открытия файла OPEN, так как оператор OPEN устанавливает указатель записи-чтения на начало файла.

ASSIGN – оператор установления связи между физическим устройством и обращением к нему в программе

```
ASSIGN устройство номер
```

устройство – CON# (клавиатура);
RDR# (устройство чтения с перфоленты);
PUN# (устройство вывода на перфоленту);
LST# (экран дисплея);

номер – значение от 0 до 3.

Оператор ASSIGN дает возможность передавать программе имя устройства ввода-вывода и управление этим устройством. Оператор изменяет значение байта ввода-вывода так, что последующие операции ввода-вывода будут осуществляться с выбранным устройством.

Например:

```
ASSIGN CON#1
```

В этом примере выбрано устройство ввода с клавиатуры с номером 1.

IN# – оператор определения устройства для ввода информации

```
IN# номер устройства
```

номер устройства – численное выражение от 0 до 7.

Если выбранное устройство не определено или не задействовано, то система "зависает". Для перезагрузки необходимо нажать клавиши RESET+CONTROL–C.

Оператор IN#0 указывает на ввод с клавиатуры.

Оператору IN# должен предшествовать вывод кода CONTROL–D:

```
PRINT CHR$(4);"IN#6"
```

INPUT – оператор ввода данных в программу из файла

Формат 1 (кроме XYBASIC и Бейсик-TRS–80).

```
INPUT #номер файла,переменная[,переменная]..
```

Формат 2 (только для XYBASIC).

```
INPUT @номер файла,переменная[,переменная]..
```

Формат 3 (только в Бейсик-TRS–80).

```
INPUT #-1,переменная[,переменная]...
```

номер файла – номер, с которым файл был открыт для ввода;

переменная – имя переменной, которой будет присвоен элемент данных.

Формат 1. Считывает элементы данных из файла с последовательным доступом и присваивает их переменным программы. Тип данных в файле должен соответствовать типу переменной. Знак вопроса не печатается. Элементы данных появляются в файле сразу, как только они введутся. Числовые значения (управляющие коды от 0 до 10), начальные пробелы, возврат каретки, перевод строки игнорируются. Первый вычисленный символ, который не является пробелом, возвратом каретки или переводом строки, будет началом числа.

Число ограничивается пробелом, возвратом каретки, переводом строки или запятой. Если Бейсик разделяет данные на строковые элементы, то начальные пробелы, перевод строки, возврат каретки также игнорируются. Началом элемента строки является первый

вычисленный символ. Если кавычки (") являются первым символом, то элемент строки будет состоять из всех символов, находящихся между первыми и вторыми кавычками.

Строка, заключенная в кавычки, не может содержать кавычки как элемент строки. А если строка не заключена в кавычки, то она ограничивается запятой, возвратом каретки, переводом строки или длиной 255 символов. Если найден признак конца файла при вводе числового или строкового элемента, то ввод элемента будет закончен. Оператор INPUT# может быть использован при вводе данных из файла произвольного доступа.

Формат 2. Если номер файла равен 0, то ввод данных осуществляется с клавиатуры.

Формат 3. Используется для ввода данных с НКМЛ.

INPUT$ – функция определения последовательности заданного количества символов из файла

```
Y$=INPUT$(n,[#]m)
```

n – количество символов;

m – номер файла, определенный в операторе OPEN.

Из файла выбирается заданное количество символов, в том числе и управляющих. Например:

```
10 OPEN "O",#1,"DATA"
20 IF EOF(1) THEN 50
30 PRINT HEX$(ASC(INPUT$(1,#1)));
40 GOTO 20
50 PRINT : END
```

IOBYTE – функция определения текущего значения байта ввода-вывода

```
X=IOBYTE
```

При использовании этой функции пользователь может всегда проконтролировать активные в данный момент устройства ввода-вывода информации.

Переменная X, стоящая в левой части выражения, будет содержать байт ввода-вывода, который побитно расписывается следующим образом:

поле LST		поле PUN		поле RDR		поле CON	
бит 7	бит 6	бит 5	бит 4	бит 3	бит 2	бит1	бит 0

Каждое двухбитовое поле выделяется под одно из четырех устройств (см. оператор ASSIGN).

LINPUT – оператор ввода строки символов из файла

```
LINPUT @номер файла,строковая переменная
```

номер файла – численное выражение, определенное в операторе OPEN;

строковая переменная – имя строковой переменной, которой будет присвоена введенная строка.

Описание см. оператор LINE INPUT.

LINE INPUT – оператор ввода в оперативную память строковой переменной из файла

```
LINE INPUT# номер файла, строковая переменная
```

номер файла – номер открытого файла;

строковая переменная – имя строковой переменной, которой будет присвоена введенная строка.

Считывает строку (длиной до 254 символов) без ограничителей из файла с последовательным доступом в строковую переменную.

Оператор LINE INPUT# считывает из файла с последовательным доступом все символы до символа возврата каретки. При повторном использовании оператора происходит считывание следующей последовательности символов до символа возврата каретки.

Оператор LINE INPUT# особенно удобен, если каждая строка файла разбита на поля или программа на языке Бейсик, сохраняемая в коде ASCII (КОИ-8), считывается как данные другой программой. Оператор LINE INPUT# может использоваться и для работы с файлом произвольного доступа. Например:

```
10 OPEN "O",#1,"LIST"
20 LINE INPUT "СТРОКА?";C$
30 PRINT #1,C$ : CLOSE 1
40 OPEN "I",#1,"LIST"
50 LINE INPUT #1,C$
60 PRINT C$ : CLOSE 1
RUN
СТРОКА? МОСКВА, ЛЕНИНСКИЙ ПРОСПЕКТ
МОСКВА, ЛЕНИНСКИЙ ПРОСПЕКТ
Ok
```

MARGIN – оператор форматирования выходных файлов

```
MARGIN @номер файла[,длина строки]
```

номер файла – номер, определенный в операторе OPEN;

длина строки – целочисленное выражение, меньшее 256.

С помощью этого оператора устанавливается длина выходной строки данных. Если параметр длина строки не указан, то по умолчанию принимается длина строки, равная 72 символам.

Если номер файла равен 0, то изменится длина строки на экране дисплея. Если длина строки больше 255, то выдается сообщение об ошибке BY, если номер файла не совпадает ни с одним номером открытых файлов, то выдается сообщение об ошибке BF.

Например:

```
MARGIN @1,100
MARGIN @0,50
```

PR# – оператор определения устройства для вывода информации

```
PR# номер устройства
```

номер устройства – численное выражение от 0 до 7.

Если выбранное устройство не определено или не задействовано, то система "зависает". Для перезагрузки необходимо нажать клавиши CONTROL–C. Оператор PR#0 выбирает вывод на экран дисплея.

Оператору PR# должен предшествовать вывод кода CONTROL–D:

```
PRINT CHR$(4);"PR#6"
```

PRINT и PRINT USING – операторы вывода данных и форматированного вывода данных в файл

Формат 1 (кроме XYBASIC и Бейсик TRS-80).

```
PRINT #номер файла,[USING X$;]список выражений
```

Формат 2 (только для XYBASIC).

```
PRINT @номер файла,[формат]список выражений
```

Формат 3 (только в Бейсик-TRS–80).

```
PRINT #-1, список выражений
```

номер файла – номер, с которым файл был открыт для вывода;

X$ – строковое выражение, определяющее формат;

список выражений – числовые или строковые выражения, которые должны быть записаны в файл;

формат – строковое выражение.

Формат 1. Оператор PRINT не сжимает данные в файле. В списке выражений необходимо разделять выражения точками с запятой (;). Если запятые (,) используются как разделители, то лишние пробелы, которые вводятся между печатными полями, также будут записываться в файл. Строковые выражения в списке выражений должны разделяться точками с запятой (;).

Например,

если `A$="HELLO" и B$="DOLLY"` тогда оператор

```
PRINT #1,A$,B$
```

запишет в файл HELLODOLLY. Поскольку нет разграничителей, то эти данные не могут быть выведены как две отдельные строки. Чтобы устранить проблему, необходимо ввести с клавиатуры ограничители

```
PRINT #1,A$,",";B$
```

и в файл запишется строка HELLO, DOLLY, которая затем может быть введена в две строки. Если сами строки содержат запятые и точки с запятой, то их надо записать в файл, заключая в кавычки.

Например,

если `A$="HELLO,FUNNY" и B$=" DOLLY"` тогда оператор

```
PRINT #1,A$,B$
```

запишет в файл HELLO, FUNNY DOLLY и оператор

```
INPUT #1,A$,B$
```

введет строку "HELLO" в A$, а строку "FUNNY DOLLY" в строку B$.

Поэтому надо использовать двойные кавычки и оператор

```
PRINT #1,CHR$(34);A$;CHR$(34);CHR$(34);B$;CHR$(34)
```

запишет в файл "HELLO, FUNNY"" DOLLY", а оператор

```
INPUT #1,A$,B$
```

введет "HELLO, FUNNY" в A$, а "DOLLY" в B$.

Оператор PRINT может быть использован с параметром USING для управления форматом вывода.

Например:

```
PRINT #1,USING "$$###.##",J;K;L
```

Формат 2. Если указан параметр формат, то строка символов, определенная в этом параметре, будет записываться в файл.

Формат 3. Используется для вывода данных на НКМЛ.

READ – оператор чтения данных из файла с последовательным доступом

```
READ имя файла, список выражений
```

Описание см. оператор INPUT в этой главе.

Оператору READ должен предшествовать вывод кода CONTROL–D:

```
PRINT CHR$(4);"READ MYFILE,P"
```

WRITE – оператор записи данных в файл с последовательным доступом

Формат 1 (кроме Бейсик-АГАТ).

```
WRITE #номер файла, список выражений
```

Формат 2 (только Бейсик-АГАТ).

```
WRITE имя файла, список выражений
```

номер файла – номер, с которым файл был открыт для вывода.

Выражения в списке могут быть числовыми и/или строковыми, они отделяются запятыми или точками с запятой.

Различие между операторами WRITE# и PRINT# заключается в том, что оператор WRITE вставляет запятые между элементами и заключает строки в кавычки. Поэтому нет необходимости ставить ограничители в списке.

После записи последнего элемента в список выражений вводится символ возврат каретки/перевод строки.

Формат 2. Оператору WRITE должен предшествовать вывод кода CONTROL–D:

```
PRINT CHR$(4);"WRITE MYFILE,P"
```

4.1.5. Файлы произвольного доступа

Создание файла и обращение к файлу произвольного доступа требует больше программных шагов, чем к файлу с последовательным доступом, но у файлов произвольного доступа существует ряд преимуществ:

требует меньше места на диске, так как они запоминаются в закодированном (упакованном) двоичном формате;

для доступа к данным не надо считывать всю информацию, так как информация записывается и считывается записями, причем записи пронумерованы.

Для работы с файлами произвольного доступа используются инструкции, представленные в табл. 4.5.

Таблица 4.5

Инструкция	Номер версии									
	1	2	3	4	5	6	7	8	9	10
CVD					+	+			+	+
CVI					+	+			+	+
CVS					+	+			+	+
FIELD					+	+			+	+
GET					+	+			+	+
LSET					+	+			+	+
MKD$					+	+			+	+
MKI$					+	+			+	+
MKS$					+	+			+	+
POSITION				+						
PUT					+	+			+	+
READ				+						
RSET					+	+			+	+
WRITE				+						

CVI, CVS, CVD – функции преобразования строковых переменных в числовые

```
Y=CVI(строка из двух байт)

Y=CVS(строка из четырех байт)

Y=CVD(строка из восьми байт)
```

При считывании из файла произвольного доступа строковые переменные преобразуются в числа. Функция CVI преобразует двухбайтовые строковые переменные в целые числа; функция CVS – четырехбайтовые строковые переменные в числа одинарной точности; функция CVD – восьмибайтовые строковые переменные в числа двойной точности.

Например:

```
10 FIELD #1,4 AS N$,12 AS B$
20 GET #1
30 Y=CVS(N$)
```

FIELD – оператор выделения памяти для переменных в буфере файла произвольного доступа

```
FIELD [#[номер,ширина AS строковая переменная][,ширина AS
                строковая переменная]...]
```

номер – номер файла, указанный в операторе OPEN;

ширина – числовое выражение, определяющее число символов для резервирования строковой переменной;

строковая переменная – строковая переменная, которая будет вводиться в файл произвольного доступа.

Оператор FIELD должен быть выполнен до получения данных из буфера файла произвольного доступа после оператора GET или после введения данных в буфер для оператора PUT. Общее число байтов, зарезервированных оператором FIELD,не должно превышать длину записи, которая была определена при открытии файла. В противном случае выдается сообщение об ошибке FIELD overflow (Переполнение поля).

Для файла с одним номером может быть выполнено любое число операторов FIELD, и все выполняемые операторы FIELD находятся в обработке одновременно. Каждый новый оператор FIELD переопределяет буфер с первой символьной позиции, так что это имеет эффект многократных определений полей для строковых данных. Например:

```
FIELD 1,20 AS N$,10 AS ID$,40 AS ADD$
```

Резервирует первые двадцать позиций в буфере файла произвольного доступа для строковой переменной N$, следующие десять позиций для ID$ и следующие сорок позиций для ADD$.

Сам оператор не размещает данные в буфере файла. Это делают операторы LSET и RSET.

Например:

```
10 OPEN "R",#1,"B:CUST"
20 FIELD 1,20 AS CUSTM$,30 AS ADD$,35 AS CITY$
30 GET 1
40 N$=CUSTM$
50 PRINT N$,ADD$,CITY$
```

GET – оператор чтения записи из файла произвольного доступа в буфер

```
GET [#]номер файла [,номер записи]
```

номер – номер файла, который был открыт оператором OPEN;

номер записи – номер записи для чтения от 1 до 32767.

Если не указан номер записи, то в буфер считывается следующая (после последнего оператора GET) запись. После выполнения оператора GET могут выполняться операторы INPUT# и LINE INPUT# для считывания символов из буфера файла произвольного доступа.

Например:

```
10 OPEN "A:CUST" AS #1
20 FIELD 1,30 AS CUSTM$
30 GET 1
40 PRINT CUSTM$
50 GET 2
```

LSET и RSET – операторы пересылки данных из памяти в буфер файла произвольного доступа.

```
LSET строковая переменная = X$

RSET строковая переменная = X$
```

Если X$ требует меньше байтов, чем было отведено для строковой переменной в операторе FIELD, то LSET пересылает символы строки, определенной в операторе FIELD, в левую часть буфера, а RSET – в правую. Пробелы используются для заполнения лишних позиций. Если X$ длиннее строковой переменной, то лишние символы отбрасываются оператором LSET справа, а RSET – слева. Числовые значения перед использованием операторами LSET и RSET должны быть преобразованы в строковые. Операторы LSET и RSET можно так же использовать со строковыми переменными, которые не были определены в FIELD. Например:

```
10 A$=SPACE(20)
20 RSET A$=N$
30 LSET A$=MKS$(ANT)
```

MKI$, MKS$ и MKD$ – функции преобразования числовых значений в строковые

```
Y$=MKI$(целочисленное выражение)

Y$=MKS$(выражение одинарной точности)

Y$=MKD$(выражение двойной точности)
```

Любое строковое значение, которое размещается в буфере файла произвольного доступа операторами LSET и RSET, должно быть переведено в строковое значение. Функция MKI$ переводит целое число в двухбайтную строку; функция MKS$ – число одинарной точности в четырехбайтную строку; функция MKD$ – число двойной точности в восьмибайтную строку.

Например:

```
90 AMT=(K+T)
100 FIELD #1,4 AS D$,20 AS N$
110 LSET D$=MKS$(AMT)
120 LSET NS=A$
130 PUT #1
```

POSITION – оператор пропуска определенного числа записей (полей) в файле произвольного доступа

```
POSITION имя файла[,Rпозиция]
```

позиция – число пропускаемых записей. Значение может меняться от 0 до 32767. По умолчанию – 0.

Символ R служит признаком файла произвольного доступа. Оператор POSITION устанавливает указатель записи в начало поля после числа пропускаемых записей, определенного в параметре **позиция**. Операторы READ и WRITE начинают работу с этого указателя. Каждая запись (поле) – это последовательность символов, оканчивающаяся символом возврата каретки/перевода строки.

Оператору POSITION должен предшествовать вывод кода CONTROL–D:

```
PRINT CHR$(4);"POSITION MYFILE,R100"
```

PUT – оператор записи из буфера файла произвольного доступа в файл данных

```
PUT [#]номер файла[,номер записи]
```

Заносит запись из буфера файла произвольного доступа в файл данных. Если номер записи не указан, то запись будет иметь следующий по порядку и разрешенный номер. Для записи символа в буфер файла произвольного доступа перед оператором PUT могут быть использованы операторы PRINT#, PRINT# USING и WRITE#. Любые попытки прочитать или записать файл данных, который превышает емкость буфера, вызовут ошибку FIELD overflow (Переполнение поля).

READ – оператор чтения данных из файла произвольного доступа

```
READ имя файла[R,запись], список выражений
```

Если указан параметр запись, то чтение осуществляется начиная с указанной записи. Запись может содержать от 1 до 32767 символов и определяется как последовательность символов, заканчивающаяся символом возврата каретки.

Оператору READ должен предшествовать вывод кода CONTROL–D:

```
PRINT CHR$(4);"READ MYFILE,P"
```

RSET – см. описание оператора LSET

WRITE – оператор записи данных в файл произвольного доступа

```
WRITE имя файла[,Rзапись],список выражений
```

номер файла – номер, под которым файл был открыт для вывода;

запись – число выводимых записей (полей).

Оператору WRITE должен предшествовать вывод кода CONTROL–D:

```
PRINT CHR$(4);"WRITE MYFILE,P"
```

4.1.6. Программные файлы

Как было сказано выше, программные файлы делятся на:

файлы на языке Бейсик, сохраненные во внутреннем формате (в промежуточном коде, полученном после преобразования исходного текста);

файлы на языке Бейсик, сохраненные в закодированном внутреннем формате (во внутреннем коде, дополнительно обработанном по специальным формулам кодирования);

файлы на языке Бейсик, сохраненные в текстовом (коде ASCII) формате;

файлы на машинном языке.

В табл. 4.6. представлены инструкции, которые обрабатывают программные файлы.

Таблица 4.6

Инструкция	Номер версии									
	1	2	3	4	5	6	7	8	9	10
BLOAD									+	+
BSAVE									+	+
CHAIN				+		+				+
CLOAD	+				+				+	
CSAVE	+				+				+	
DEF SEG										+
LOAD	+	+	+	+		+	+	+	+	+
MERGE			+			+	+	+	+	+
SAVE	+	+	+	+		+	+	+	+	+
VERIFY			+							

BLOAD – команда загрузки программы, написанной на машинном языке

```
BLOAD "имя файла"[,[R][,смещение]]
```

смещение – величина, определяющая смещение физического адреса загрузки от заданного Команда BLOAD загружает в память файлы с НГМД или НКМЛ, сохраненные командой BSAVE. После загрузки программы, если задан параметр R, управление передается загруженной программе. Иначе интерпретатор переходит в командный режим.

Например:

```
BLOAD"CAS:MOUSE.GM",R
```

С НКМЛ загружается программа MOUSE.GM и ей передается управление.

При чтении с НКМЛ, если заданное имя файла не совпадает с записанным на ленте, выдается сообщение Skip (Пропущен), если совпадает, то выдается сообщение Found (Найден).

Использование команды BLOAD в программе не вызывает стирания информации в памяти, которую занимает Бейсик-программа, поэтому эту команду можно использовать для загрузки отдельных машинных программ, которые, например, ускоряли бы действие интерпретатора.

BSAVE – команда сохранения на определенном носителе программы в машинных кодах или определенной области памяти

```
BSAVE "спецификация файла",начальный адрес,конечный
адрес[,стартовый адрес]
```

начальный адрес – начальный адрес программы или области памяти;

конечный адрес – конечный адрес программы или области памяти;

стартовый адрес – стартовый адрес программы.

Все адреса могут быть заданы в любой форме представления. Если не указан стартовый адрес, то вместо него записывается начальный адрес программы.

Команда BSAVE записывает программу или область данных на НКМД или НКМЛ в следующем формате:

1-й байт – признак двоичного файла (0FEH);

2–3-й байты – начальный адрес;

4–5-й байты – конечный адрес;
6–7-й байты – стартовый адрес;
далее – данные.

Если какой-нибудь параметр не указан, то выдается сообщение об ошибке Missing operand (Ошибочный операнд) или Sintax error (Синтаксическая ошибка).

CHAIN – оператор передачи управления другой подпрограмме с передачей переменных из текущей программы

```
CHAIN [MERGE]"имя файла"[,[номер строки] [,[ALL][,DELETE
                диапазон]]]
```

имя файла – имя файла вызываемой программы;

номер строки – номер строки или выражение, которое вычисляет номер строки в вызванной программе. Это начальная точка, с которой начинается выполнение вызванной программы. Если этот оператор не указан, то выполнение программы начинается с первой строки. На этот параметр не действует команда RENUM;

ALL – если указан параметр ALL, то все переменные из текущей программы передаются в вызванную программу. Если параметр ALL не указан, то текущая программа должна содержать оператор COMMON;

MERGE – если указан параметр MERGE, то подпрограмма включается в программу на языке Бейсик как оверлей, т. е. параметр MERGE выполняется с текущей и вызванной программами. Вызванная программа должна быть файлом в текстовом формате (в коде ASCII (КОИ-8));

DELETE – используется для удаления использованного оверлея. Действие аналогично команде DELETE;

диапазон – диапазон номеров строк, которые надо удалить перед загрузкой. На номера строк в диапазоне действует команда RENUM.

Оператор CHAIN с параметром MERGE оставляет открытыми файлы и защищает текущую установку OPTION BASE. Если параметр MERGE не указан, то не защищены типы переменных и функций, определенных пользователем для сцепленной программы, и должны быть заново определены в сцепленной программе. Например:

```
CHAIN "A:PROG1"
CHAIN "A:PROG1",1000
CHAIN "A:PROG1",1000,ALL
CHAIN MERGE "A:OUERLAY",1000
CHAIN MERGE "A:OUERLAY2",1000,DELETE 1000-5000
```

CLOAD – команда загрузки программы на языке Бейсик, находящейся на НКМЛ в упакованном формате

Формат 1 (кроме Бейсик-TRS–80).

```
CLOAD [?]"имя файла"
```

Формат 2 (только для Бейсик-TRS–80).

```
CLOAD? "имя файла"
```

Загружаемая программа считывается с НКМЛ в память. Имя программы должно быть не более 6 символов. Если имя не задано, то выдается сообщение Illegal function call (Неверный функциональный вызов).

Перед загрузкой файла происходит очистка памяти (выполняется команда NEW), затем данные считываются единым блоком.

Если задан параметр ?, то команда NEW не выполняется, вместо этого происходит побайтное сравнение загружаемой программы и программы, находящейся в памяти. Если при сравнении найдено несоответствие, то выдается сообщение Verify error (Ошибка проверки).

Например:

```
CLOAD "TEST"
Found: TEST
Ok
```

CSAVE – команда записи Бейсик-программы на НКМЛ

Формат 1 (для MSX-BASIC).

CSAVE "имя файла"[,скорость]

Формат 2 (для Бейсик-TRS-80).

CSAVE "имя файла"

имя файла – строковое выражение, из которого воспринимаются первые 6 символов;

скорость – арифметическое выражение, принимающее значение 1 или 2.

Формат 1. Записывает текст программы на языке Бейсик, на НКМЛ в упакованном формате. Команда CSAVE сохраняет текст Бейсик-программы, используя внутренний формат. В качестве имени файла можно использовать практически любую строку символов, но воспринимаются только 6 первых символов. Если заданное имя содержит меньше 6 символов, то оно дополняется пробелами.

Скорость передачи информации на ленту из соображений надежности равна 1200 бод (по умолчанию) и может быть увеличена до 2400 бод (для этой скорости требуется хорошая лентопротяжка). Скорость можно задать в операторе SCREEN, см. гл. 6 (для Бейсик-ПК8010 и Бейсик-ПК8020 – в операторе SPEED, см. 4.3) или с помощью второго аргумента в CSAVE. Если второй аргумент равен 1, то устанавливается скорость 1200 бод, если 2 – то 2400 бод.

При отсутствии или неверном задании имени выдается сообщение Missing operand (Ошибочный операнд), Illegal function call (Неправильный функциональный вызов).

Формат 2. Параметра **скорость** нет. Например:

```
CSAVE "PROGR",2
```

Файл PROGR сохраняется на ленте, скорость считывания 2400 бод.

DEF SEG – оператор определения текущего сегмента памяти

DEF SEG[=адрес]

адрес – числовое выражение от 0 до 65535.

При использовании этого оператора некоторые команды и операторы, такие как BLOAD, BSAVE, CALL, PEEK, POKE или USR будут определять при своей работе действительный физический адрес как смещение в этом сегменте памяти. Если параметр **адрес** не указан, то сегмент устанавливается по адресу сегмента данных интерпретатора, который является начальным адресом по умолчанию. Сегмент данных – это начало рабочего пространства в памяти. Если адрес задан, то его значение должно быть выравнено на 16-байтовую границу. Значение сдвигается влево на 4 бита (что дает умножение на 16) для формирования адреса сегмента для следующей операции. Интерпретатор не проверяет адрес сегмента.

Слова DEF и SEG должны быть разделены пробелом.

Любое значение параметра адрес, выходящее за указанный диапазон, вызывает сообщение об ошибке Illegal function call (Неправильный функциональный вызов).

Например:

```
100 DEF SEG=0
110 Y=PEEK(72) : REM выбор байта по адресу 0:72
```

LOAD – команда загрузки программы с определенного устройства в память

Формат 1 (для XYBASIC).

```
                                        [,H]
LOAD "спецификация файла"{              }[,R]
                                        [,A]
```

Формат 2 (для Бейсик-Спектрум+2).

```
                                    [,DATA имя массива
LOAD (!) "спецификация файла" { [,CODE[адрес,количество]]
                                    [,SCREEN$]
```

Формат 3 (для Бейсик-АГАТ).

```
LOAD "спецификация файла"[,устройство]
```

Формат 4 (для MBASIC, MSX-BASIC, BASICA).

```
LOAD "спецификация файла"[,R]
```

Формат 5 (для Бейсик-ПК8010, Бейсик-ПК8020)

```
                                    [,A]
LOAD "спецификация файла"] {        }[,R]
                                    [,B[,смещение]]
```

Формат 1. Параметры **A** и **H** определяют тип загружаемого программного файла. Если указан параметр **A**, то загружается программный файл, который был сохранен в текстовом формате (в коде ASCII (КОИ-8)). Если в имени файла не указано расширение, то по умолчанию подставляется расширение .BAS. Если указан параметр **H**, то загружается программа, сохраненная в шестнадцатеричном формате. Расширение по умолчанию – .HEX. Если не указаны параметры **A** или **H**, то будет загружаться программа, сохраненная во внутреннем формате и имеющая расширение .XYB.

Формат 2. Если указан символ !, то загрузка происходит с "электронного" диска, иначе – с НКМЛ. Если имя программы не указано, то с НКМЛ считывается первый встреченный файл заданного типа. Ниже приведено описание параметров.

DATA – с магнитной ленты загружается массив (числовой или строковый) и при этом из памяти удаляется массив, имеющий то же имя, что и указанное в операторе LOAD. При нахождении числового массива на экране появляется сообщение Number array: (Числовой массив:), при нахождении строкового массива – Character array: (Строковый массив:). При загрузке строкового массива из памяти стираются не только строковые массивы с тем же именем, но и строковые переменные, имеющие то же имя.

CODE – с магнитной ленты загружается набор байтов. Параметр **адрес** определяет начало области размещения байтов, параметр **количество** определяет количество считываемых

байтов. Если записанных байтов больше, чем указано, то считывания не происходит и появляется сообщение Tape Loading error (Ошибка ввода с ленты).

SCREEN$ – специальная форма CODE 16384,6912. Загружает данный набор байтов в компьютер.

Формат 3. Загружает программный файл с НКМД.

Формат 4. Загружает программные файлы на языке Бейсик во внутреннем и в текстовом форматах. Если вместо имени дисковода в имени программы указано "CAS:", то происходит загрузка текстового программного файла с НКМЛ (кроме MBASIC). При загрузке с НКМЛ, если не задано имя, считывается первый встречный текстовой файл. Имя загружаемой программы не должно быть больше 6 символов. Если в имени файла устройство не указано, то выбирается текущий дисковод. К имени файла прибавляется расширение .BAS, если имя файла меньше или равно 8 символам (кроме MSX-BASIC). Если имя определено ие по правилам, то выдается сообщение Bad file name (Неверное имя файла) и загрузка не производится.

Если указанного файла нет на устройстве, то выдается сообщение File not found (Файл не иайден), и загрузка не производится.

Формат 5. В Бейсик-ПК8010 программа загружается только с НКМЛ при задании устройства "CAS:". Имя программы не должно быть больше 6 символов. Если имя, заданное в команде, или тип не совпали с записанным на ленте, то выдается сообщение ПРОПУЩЕН, если же имя и тип файла совпали, то выдается сообщение НАЙДЕН. При чтении с НКМЛ текстового файла блоками по 256 байт после каждого корректио считанного блока данных выдается сообщение *. Если указан параметр **A**, то происходит загрузка текстового программного файла, если параметр **B**, то загружается набор байтов (двоичный файл) в область, адрес которой записан на НКМЛ. Параметр **смещение** позволяет загруженную программу перемещать в ОЗУ на заданный адрес.

В Бейсик-ПК8020, если в спецификации файла имя устройства не указано, выбирается текущий дисковод. К имени файла прибавляется расширение .BAS, если имя файла меньше или равно 8 символам. Если имя определено не по правилам, то выдается сообщение Bad file name (Неверное имя файла), и загрузка не производится. Если указанного файла нет на устройстве, то выдается сообщение File not found (Файл не найден), и загрузка также не производится.

Если в вышеперечисленных форматах команды указан параметр R, то программа загружается и сразу запускается на выполнение. Если параметр R отсутствует, то после загрузки программы интерпретатор переходит в командный режим. В большинстве рассматриваемых версий интерпретаторов языка Бейсик команде **LOAD "спецификация файла", R** эквивалентна команда **RUN "спецификация файла"**.

Команда LOAD закрывает все открытые файлы и удаляет все переменные и строки программы в памяти, однако при использовании параметра R все открытые файлы данных остаются открытыми. Например:

```
LOAD "INVENT",R
```

С текущего диска загружается программа INVENT.BAS и выполняется.

MERGE – команда подзагрузки программы с определенного устройства в оперативную память и слияния ее с существующей в памяти программой

Формат 1. (для Бейсик-Спектрум+2).

```
MERGE (!) "спецификация файла"
```

Формат 2. (для MBASIC, MSX-BASIC, BASICA, Бейсик-ПК8010 и Бейсик-ПК8020).

```
MERGE "спецификация файла"[,R]
```

Формат 1. Если указан символ !, то загрузка происходит с "электронного" диска, иначе – с НКМЛ. Если имя программы не указано, то с НКМЛ подзагружается первый встреченный файл.

Формат 2. Подзагружает программные файлы на языке Бейсик в текстовом формате. Если вместо имени дисковода в спецификации файла указано "CAS:", то происходит загрузка текстового программного файла с НКМЛ (кроме MBASIC). При загрузке с НКМЛ, если не задано имя программы, считывается первый встреченный текстовый файл. Имя программы не должно быть больше 6 символов. Если в имени файла устройство не указано, то выбирается текущий дисковод. К имени файла прибавляется расширение .BAS, если имя файла меньше или равно 8 символам (кроме MSX-BASIC). Если имя определено не по правилам, то выдается сообщение Bad file name (Неверное имя файла) и загрузка не производится.

Если указанного файла нет на устройстве, то выдается сообщение File not found (Файл не найден) и загрузка также не производится.

В Бейсик-ПК8010 подзагрузка производится только с НКМЛ. Если имя, заданное в команде, или тип не совпали с записанными на ленте, то выдается сообщение ПРОПУЩЕН, если же имя и тип файла совпали, то выдается сообщение НАЙДЕН. При чтении с НКМЛ текстового файла блоками по 256 байт после каждого корректно считанного блока данных выдается сообщение *.

Если в спецификации файла устройство не указано, то выбирается текущий дисковод. К имени файла прибавляется расширение .BAS, если имя файла меньше или равно 8 символам. Если имя определено не по правилам, то выдается сообщение Bad file name (Неверное имя файла), и загрузка не производится. Если указанного файла нет на устройстве, то выдается сообщение File not found (Файл не найден), и загрузка также не производится.

Если указан параметр R, то после загрузки программы она запускается на выполнение. Если параметр R не указан, то после загрузки программы интерпретатор переходит в командный режим.

Команда MERGE не закрывает все открытые файлы и не удаляет все переменные и строки программы в памяти, если номера загружаемых программных строк не совпадают с уже существующими в памяти. Например:

```
MERGE "INVENT",R
```

С текущего диска подзагружается программа INVENT.BAS и выполняется.

SAVE – команда записи программы из памяти на определенное устройство

Формат 1 (для XYBASIC).

```
                                  [,H]
SAVE "спецификация файла"{        }
                                  [,A]
```

Формат 2 (для Бейсик-Спектрум+2).

```
                                [,LINE номер строки]
                                [,DATA имя массива
SAVE (!) "спецификация файла" { [,CODE[адрес,количество]] }
                                [,SCREEN$]
```

Формат 3 (для Бейсик-АГАТ).

```
SAVE "спецификация файла"[,устройство]
```

Формат 4 (для MBASIC, MSX-BASIC, BASICA).

```
                                [,A]
SAVE "спецификация файла"{      }
                                [,P]
```

Формат 5 (для Бейсик-ПК8020, Бейсик-ПК8010).

```
                                    [,A]
SAVE "спецификация файла"] {    }
[,B,начальный адрес,конечный адрес[стартовый адрес]]
```

Формат 1. Параметры **A** и **H** определяют тип сохраняемого программного файла. Если указан параметр A, то сохраняется программный файл в текстовом формате (в коде ASCII (КОИ-8)), и если в спецификации файла не указано расширение, то по умолчанию подставляется расширение .BAS. Если указан параметр H, то сохраняется программа в шестнадцатеричном формате. Расширение по умолчанию – .HEX. Если не указаны параметры A или H, то программа сохраняется во внутреннем формате с расширением .XYB.

Формат 2. Если указан символ !, то программа сохраняется на "электронном" диске, иначе – на НКМЛ. Если имя программы не указано, то на НКМЛ записывается файл с именем, состоящим из пробелов. Ниже приведено описание параметров:

LINE – сохраняет программу и переменные таким образом, что LOAD при загрузке автоматически выполняет команду **GOTO номер строки**;

DATA – сохраняет массив (числовой или строковый);

CODE – сохраняет содержимое области памяти – набор байтов. Параметры **адрес** и **количество** определяют начало области, откуда начинается запись байтов, и количество записываемых байтов соответственно;

SCREEN$ – специальная форма CODE: 16384,6912; сохраняет данный набор байтов из экранной области памяти компьютера.

Формат 3. Записывает программный файл на НГМД.

Формат 4. Записывает программные файлы на языке Бейсик во внутреннем формате и, если указан параметр **A**, в текстовом формате. Если указан параметр **P**, то программа сохраняется в закодированном внутреннем формате, который после загрузки запрещает внесение изменений в текст программы, а также запрещает просмотр текста программы. Если вместо имени дисковода в спецификации файла указано "CAS:", то происходит запись программного файла на НКМЛ (кроме MBASIC). Если имя программы не указано, то на НКМЛ записывается файл с именем, состоящим из пробелов. Имя программы на НКМЛ не должно быть больше 6 символов. Если в спецификации файла устройство не указано, то выбирается текущий дисковод. К имени файла прибавляется расширение .BAS, если имя файла меньше или равно 8 символам (кроме MSX-BASIC). Если имя определено не по правилам, то выдается сообщение Bad file name (Неверное имя файла) и программа не сохраняется.

Формат 5. В Бейсик-ПК8010 программа сохраняется только на НКМЛ при задании устройства "CAS:". Имя программы не должно быть больше 6 символов. При сохранении на

НКМЛ текстового файла запись производится блоками по 256 байт. Если указан параметр **A**, то происходит запись текстового программного файла, если параметр **B**, то записывается набор байтов (двоичный файл) из области, начальный и конечный адреса которой указаны в команде. Параметр **стартовый адрес** позволяет определить стартовый адрес программы, на который передается управление при загрузке программы командой LOAD с параметром **R**. Если в спецификации файла устройство не указано, то выбирается текущий дисковод. К имени файла прибавляется расширение .BAS, если имя файла меньше или равно 8 символам. Если имя определено не по правилам, то выдается сообщение НЕВЕРНОЕ ИМЯ ФАЙЛА, и запись не производится. Команда SAVE закрывает все открытые файлы.

Например:

```
SAVE "INVENT",A
```

На текущий диск записывается программа INVENT.BAS в текстовом формате.

VERIFY – команда сравнения программы и переменных, считываемых с НКМЛ, с программами и переменными в памяти. (Используется только в Бейсик-Спектрум+2.)

```
                              [,DATA имя массива]
VERIFY "спецификация файла" {                    }
                              [,CODE[адрес,количество]]
```

DATA – сравнивает массив, имя которого задано в команде, с существующим массивом в памяти. Эти массивы могут быть числовыми или строковыми;

CODE – если явно заданы параметры **адрес** и **количество** и записанное на НКМЛ количество байтов не больше указанного в команде, то выполняется сравнение блока байтов, находящихся на НКМЛ с блоком байтов в памяти, начиная с адреса. Если же параметры адрес и количество не указаны, то происходит сравнение всех байтов, записанных на НКМЛ, с байтами в памяти, начиная с адреса, записанного на НКМЛ.

4.2. Работа с функциональными клавишами

Функциональные клавиши представляют собой специальные клавиши, при нажатии на которые происходят заранее определенные действия. В некоторых компьютерах функциональные клавиши запрограммированы на ввод наиболее распространенных и часто используемых операторов и команд языка Бейсик. Это значительно облегчает работу пользователя по набору и отладке программ.

В табл. 4.7 приведены исходные значения функциональных клавиш, которые используются в КУВТ "КОРВЕТ" (Бейсик-ПК8010, Бейсик-ПК8020), MSX-BASIC, BASICA.

Таблица 4.7

Функциональная клавиша	КУВТ "КОРВЕТ"	MSX-BASIC	BASICA
F1	CLS [вк/пс]	color [пр]	LIST
F2	LOAD"	auto [пр]	RUN [вк/пс]
F3	EDIT [пр]	goto [пр]	LOAD"
F4	AUTO [пр]	list [пр]	SAVE"

Окончание табл. 4.7

Функциональная клавиша	КУВТ "КОРВЕТ"	MSX-BASIC	BASICA
F5	RUN [вк/пс]	run [вк/пс]	CONT [вк/пс]
F6	PCLS [вк/пс]	color 15,4,4,	"LPT1:" [вк/пс]
F7	SAVE"	cload "	TRON [вк/пс]
F8	LIST [пр]	cont [вк/пс]	TROFF [вк/пс]
F9	RENUN [пр]	list [вк/пс]	KEY
F10	CONT [вк/пс]	[o]run[вк/пс]	SCREEN 0,0,0 [вк/пс]

Примечание. [вк/пс] – возврат каретки/перевод строки; [пр] – пробел; [в] – курсор вверх; [о] – очистка экрана.

В MSX-BASIC и BASICA функциональные клавиши можно перекодировать, т. е. придавать нужные пользователю значения. Для этой цели используются операторы: KEY; KEY LIST; KEY ON; KEY OFF; KEY (n); ON KEY.

KEY – оператор установки и распечатки значений функциональных клавиш

```
KEY n,X$

KEY LIST

KEY ON

KEY OFF
```

n – номер функциональной клавиши;
X$ – текст, присваеваемый функциональной клавише.

С помощью оператора KEY любой функциональной клавише можно присвоить новое строковое значение длиной до 15 символов. Когда клавиша нажимается, в буфер клавиатуры вводится ее строковое значение.

Оператор KEY ON распечатывает значения программируемых клавиш на 25-й строке экрана. Когда ширина экрана равна 40 символам, распечатываются значения только пяти клавиш из десяти. При ширине экрана, равной 80 символам, распечатываются значения всех десяти клавиш. При любой ширине экрана распечатываются только шесть первых символов каждого значения.

Оператор KEY OFF удаляет распечатку значений программируемых клавиш с 25-й строки экрана, делая эту строку доступной для программного использования. Оператор KEY OFF не блокирует функциональные клавиши.

Оператор KEY LIST распечатывает все десять значений программируемых клавиш на экране. Распечатываются все 15 символов каждого значения.

Оператор KEY n,X$ присваивает строку X$ функциональной клавише n. Строка X$ может быть длиной до 15 символов. Если длина строки больше 15 символов, то лишние символы отбрасываются. Присваивание пустой строки, т. е. строки длиной 0 символов, блокирует функциональную клавишу. Если значение n не лежит в диапазоне 1 – 10, то появляется сообщение об ошибке Illegal function call (Неверный функциональный вызов), и значение клавиши остается прежним.

При нажатии программируемой клавиши функция INKEY$ определяет один символ строки значения клавиши. Если функциональная клавиша блокируется, то функция INKEY$ определяет строку длиной два символа. Первый символ – двоичный 0, второй символ – код сканируемой клавиши.

После выключения распечатки функциональных клавиш оператором KEY OFF можно использовать оператор LOCATE 25,1, за которым следует оператор PRINT для вывода информации на экран.

Например:

```
10 KEY 1,"FILES"+CHR$(13)
```

Присваивает функциональной клавише F1 значение FILES с возвратом каретки.

KEY (n) – оператор блокировки/разблокировки ловушки определенных клавиш

```
KEY (n)

KEY (n) ON

KEY (n) OFF

KEY (n) STOP
```

n – номер клавиши от 1 до 14; указывает клавишу, нажатие которой должно быть определено: 1 – 10 – функциональные клавиши F1–F10; 11 – курсор вверх; 12 – курсор влево; 13 – курсор вправо; 14 – курсор вниз.

Оператор KEY (n) ON должен быть выполнен, чтобы разблокировать ловушку функциональной клавиши или клавиши управления курсором. Если в операторе ON KEY (n) задан ненулевой номер строки, то после выполнения оператора KEY (n) ON каждый раз будет выполняться оператор, который проверяет, была ли нажата заданная клавиша. Если заданная клавиша нажата, то будет выполняться переход (COSUB), определенный в операторе ON KEY.

После выполнения оператора KEY (n) OFF ловушка блокируется и событие нажатия клавиши не запоминается, если оно имело место.

После выполнения оператора KEY (n) STOP ловушка блокируется, но если клавиша нажимается, то это событие запоминается, и ловушка срабатывает, как только выполнится оператор KEY (n) ON.

ON KEY – оператор установки номера строки для перехода в подпрограмму ловушки нажатия клавиши

```
ON KEY (n) GOSUB номер строки
```

n – номер клавиши от 1 до 14: 1 – 10 – функциональные клавиши F1–F10; 11 – курсор вверх; 12 – курсор влево; 13 – курсор вправо; 14 – курсор вниз.

Если задан нулевой номер строки, то ловушка функциональной клавиши блокируется.

До выполнения оператора ON KEY должен быть выполнен оператор KEY (n) ON. После этого, если в операторе ON KEY задан номер строки, каждый раз будет проверяться, нажата ли

клавиша n. Если эта клавиша нажата, то будет осуществляться переход на заданный номер строки.

Если выполнился оператор KEY (n) OFF, то ловушка для определенных клавиш заблокировалась, и событие нажатия клавиши не зафиксировалось, даже если оно имело место.

После выполнения оператора KEY (n) STOP ловушка для определенной клавиши блокируется, но событие нажатия клавиши запоминается, и как только будет выполнен оператор KEY (n) ON, ловушка сработает.

После того как ловушка сработает автоматически, выполнится оператор KEY (n) STOP, поэтому не может быть рекурсивных ловушек. Возврат из подпрограммы ловушки автоматически выполняет оператор KEY (n) ON, если внутри подпрограммы не выполнился явный KEY (n) ON.

4.3. Дополнительные возможности работы с экраном дисплея

В дополнение к уже описанным командам и операторам работы с экраном дисплея здесь будут рассмотрены дополнительные инструкции, представленные в табл. 4.8.

Таблица 4.8

Инструкция	Номер версии									
	1	2	3	4	5	6	7	8	9	10
CSRLIN	+						+	+	+	+
HOME				+						
HTAB				+						
LOCATE							+	+	+	+
POS				+			+	+	+	+
SPEED				+						
VTAB				+						

CSRLIN – переменная, определяющая вертикальную координату курсора

```
Y=CSRLIN
```

Полученное значение будет лежать в диапазоне 1 – 25 для BASICA; 0 – 22(23) для MSX-BASIC; 1 – 15 для Бейсик-ПК8010 и Бейсик-ПК8020. Например:

```
10 Y=CSRLIN
20 X=POS(0)
30 LOCATE 24,1 : PRINT "HELLO"
40 LOCATE Y,X
```

HOME – оператор очистки экрана

```
HOME
```

Очищает экран и переводит курсор в левый верхний угол экрана.

HTAB – оператор горизонтального перемещения курсора в заданную позицию текущей строки

```
HTAB X
```

X – имеет значение от 0 до 31.

LOCATE – оператор установки курсора в нужную позицию экрана

LOCATE [X],[Y],[ключ]

X – X-позиция (столбец), в который следует поместить курсор;

Y – Y-позиция (строка), в которую следует поместить курсор;

ключ – числовое выражение.

Устанавливает курсор в любую позицию текстового экрана. Если задан ключ, равный 0, то курсор исчезает, если равный 1, то курсор высвечивается снова. Все значения описанных выше переменных должны находиться в диапазоне от 0 до 255, в противном случае выдается сообщение об ошибке Illegal function call (Неверный функциональный вызов). Кроме того, любой элемент может быть пропущен, но его запятая должна присутствовать перед любым следующим элементом.

Оператор LOCATE управляет курсором в текстовом режиме работы. Он воспринимается и в других режимах, но его действие сохраняется и используется в следующем появляющемся тексте на экране.

Строки текста на экране нумеруются сверху вниз. В MSX-BASIC – от 0 до 22, в BASICA – от 0 до 25, в Бейсик-ПК8010 и Бейсик ПК-8020 – от 1 до 16. Аналогично столбцы нумеруются слева направо от 0 до последней колонки, значение которой может быть изменено в операторе WIDTH. Оператор LOCATE перемещает курсор в указанное знакоместо. Если X или Y пропущены, то сохраняется предыдущее значение. Например:

```
10 LOCATE 1,10 : PRINT "ПРОВЕРКА LOCATE"

20 LOCATE ,,0 : PRINT "ОТКЛЮЧЕНИЕ КУРСОРА"
```

POS – функция, определяющая горизонтальную координату курсора

X=POS(n)

n – фиктивный аргумент.

Полученное значение должно находиться в пределах, установленных каждой из версий языка. Например, для версий языка Бейсик, используемых в машинах ПК8010 и ПК8020, полученное значение изменяется от 1 до 64, для PC IBM – от 1 до 40 (или от 1 до 80).

SPEED – оператор, устанавливающий скорость вывода информации на экран

SPEED=n

n – численное выражение от 0 до 255.

VTAB – оператор вертикального перемещения курсора по столбцу

VTAB Y

Y – значение от 0 до 31

4.4. Работа с периферийными устройствами

В этом разделе будут рассмотрены инструкции (табл. 4.9), обеспечивающие работу с джойстиками, световыми перьями и другими устройствами, расширяющими возможности компьютера.

Таблица 4.9

Инструкция	Номер версии									
	1	2	3	4	5	6	7	8	9	10
COPY			+							
FORMAT			+							
LPOS	+					+		+	+	+
LPRINT	+		+		+	+		+	+	+
LPRINT USING	+		+		+	+		+	+	+
MOTOR	+						+	+	+	+
MOTOR ON	+						+	+	+	+
MOTOR OFF	+						+	+	+	+
ON PEN										+
ON STRIG								+	+	+
PAD									+	
PDL									+	
PEN										+
PEN ON										+
PEN OFF										+
PEN STOP										+
SPEED							+	+		
STICK								+	+	+
STRIG								+	+	+
STRIG ON									+	+
STRIG OFF									+	+
STRIG STOP									+	+

COPY – команда, выводящая на печатающее устройство копию изображения на экране

```
COPY
```

Команда COPY требует 15 – 30 с для того, чтобы началась печать. На печатающем устройстве появляется полная копия изображения на экране.

FORMAT – команда установки скорости передачи информации на печатающее устройство

```
FORMAT "p";скорость
```

Для того чтобы печатающее устройство и компьютер могли обмениваться сообщениями, необходимо установить скорость обмена, заданную для используемого типа печатающего устройства. Если в печатающем устройстве используется скорость 9600 бод, то команду FORMAT можно не использовать.

LPOS – функция, определяющая текущую позицию головки печатающего устройства в пределах буфера памяти

```
Y=LPOS(n)
```

n – фиктивный аргумент.

Например:

```
100 IF LPOS(X)>60 THEN LPRINT CHR$(13)
```

LPRINT и **LPRINT USING** – операторы вывода данных на печатающее устройство

```
LPRINT [список выражений][;/,]

LPRINT USING X$;список выражений[;/,]
```

Эти операторы функционируют подобно операторам PRINT и PRINT USING за исключением того, что вывод идет на печатающее устройство. Оператор LPRINT допускает длину строки до 128 символов. Печать свыше 128 символов происходит с переводом строки.

MOTOR – команда управления двигателем магнитофона

MOTOR [ON/OFF]

Включает и выключает двигатель кассетного магнитофона: MOTOR ON – включение; MOTOR OFF – выключение. Если не указано ON или OFF, то предыдущее состояние двигателя магнитофона инвертируется на обратное, т. е. если магнитофон был включен, то он выключается; если выключен, то включается.

ON PEN – оператор установки номера строки при работе со световым пером. (Используется только в BASICA.)

ON PEN GOSUB номер строки

Номер строки, равный 0, блокирует ловушку светового пера. Для активизации оператора ON PEN должен выполниться оператор PEN ON, после чего, если в операторе ON PEN задан не нулевой номер строки, каждый раз будет выполняться новый оператор PEN ON, который будет проверять, активизировалось ли световое перо. Если это так, то будет выполняться переход (GOSUB) к определенной строке. Если перо выключено (PEN OFF), то ловушка не будет работать и событие не запомнится, если даже оно и произошло.

При встрече ловушки автоматически выполняется оператор PEN STOP, поэтому рекурсивные ловушки запрограммировать не удается. Возврат из подпрограммы ловушки автоматически делает оператор PEN ON, если внутри программы не был задан явный оператор PEN OFF.

При включенном пере запрещается делать операции ввода-вывода на кассетную магнитную ленту.

ON STRIG – оператор, разрешающий обработку запроса от кнопки активного джойстика

ON STRIG (номер) GOSUB номер строки

номер – число 0 – 2, определяющее источник прерываний. Обрабатывает прерывания от трех источников для перехода на заданный номер строки: при значении номера 0 опрашивается клавиша "пробел" на клавиатуре; при значении номера 1 и 2 – кнопка джойстика 1 или кнопка джойстика 2 соответственно.

PAD – функция считывания данных с графического планшета

X=PAD(выражение)

выражение – любое арифметическое выражение, имеющее значение 0 – 7.

Эта функция предназначена для ввода координат с графического планшета, подключенного к одному из портов расширения.

Выражение определяет вид информации, считываемой с графического планшета (табл. 4.10).

При выполнении PAD(0) одновременно считываются значения, которые получаются с помощью PAD(1) и PAD(2); то же касается PAD(4), PAD(5) и PAD(6). Таким образом, когда PAD(0) или PAD(4) возвращают значение ИСТИНА (–1) и X,Y – координаты считываются и хранятся до запроса.

Таблица 4.10

Тип информации	Порт расширения 1	Порт расширения 2
"блокнот" (-1, если нажато, иначе 0)	0	4
координата X после чтения	1	5
координата Y после чтения	2	6
состояние кнопки (-1, если нажата, иначе 0)	3	7

Например:

```
10 IF PAD(0) THEN X=PAD(1):Y=PAD(2):PSET(X,Y)
```

PDL – функция определения кода положения игрового рычажка

X=PDL(выражение)

выражение – любое арифметическое выражение, принимающее значение от 1 до 12.

К любому порту расширения можно подключить только один рычажок. Считывание производится с помощью дискретных импульсов, время считывания примерно 0,01 с.

Если значение выражения нечетное, то информация считывается с первого порта расширения, если четное – со второго.

Функция PDL упрощает ввод данных в программу с некоторых устройств. Обычно PDL(1) и PDL(2) используются для считывания с портов расширения 1 и 2. Если нет подключенного рычажка, то функция образует значение 255, если есть – от 0 до 254.

Например:

```
10 SCREEN 2
20 PSET(PDL(1),PDL(2)):GOTO 20
```

PEN – оператор и функция, вводящие значения, считанные со светового пера

PEN ON/OFF/STOP

X=PEN(n)

n – целочисленное выражение от 0 до 9.

Функция X=PEN(n) считывает координаты светового пера, n может принимать следующие значения:

0 – флаг, указывающий, внизу ли перо с последнего опроса. Если внизу, возвращается 1, если нет – 0;

1 – возвращает координату X, где перо было последний раз активизировано;

2 – возвращает текущее значение координат Y;

3 – возвращает текущее значение координат X;

4 – возвращает последнее известное значение координаты X;

5 – возвращает последнее известное значение координаты Y;

6 – возвращает позицию ряда символов, где перо было последний раз модифицировано;

7 – возвращает позицию столбца символов, где перо было модифицировано;

8 – возвращает последнее известное значение ряда символа;

9 – возвращает последнее известное значение позиции символа.

Когда перо находится внизу, в граничной области экрана, возвращаемые значения могут быть неточны.

Оператор PEN ON разблокирует функцию чтения PEN. Первоначально функция PEN выключена. Перед любыми вызовами функции чтения светового пера, оператор PEN ON должен быть включен.

Оператор PEN OFF блокирует функцию PEN, т. е. разблокирует ловушку оператора ON PEN, но действие светового пера не запоминается.

Оператор PEN STOP блокирует ловушку активности светового пера, но если событие происходит, то оно запоминается, и после выполнения оператора PEN ON сработает ловушка. Например:

```
10 PEN ON
20 FOR I=1 TO 500
30 X=PEN(0):X1=PEN(3)
40 PRINT X,X1
50 NEXT
60 PEN OFF
```

В данном примере печатается значение пера с последнего опроса и текущее значение.

SPEED – команда установки скорости записи на магнитную ленту

SPEED выражение

выражение – если выражение равно 1, то скорость 1200 бод, если равно 2, то скорость 2400 бод.

STICK – функция, определяющая код положения ручки джойстика

X=STICK(число)

число – арифметическое выражение от 0 до 2 (см. ON STRIG).

Функция выдает состояние джойстика 1 (STICK(1)) или джойстика 2 (STICK(2)). Положение ручки джойстика определяется числом от 0 до 8:

0 – нейтральное положение ручки джойстика;

1 – ручка вперед;

2 – ручка вверх и вправо по диагонали;

3 – ручка вправо;

4 – ручка вниз и вправо по диагонали;

5 – ручка вниз;

6 – ручка вниз и влево по диагонали;

7 – ручка влево;

8 – ручка влево и вверх по диагонали.

Если задан STICK(0), то в качестве джойстика используется клавиатура.

STRIG – оператор и функция определения признака нажатия кнопки на рукоятке джойстика

X=STRIG(число)

ON

STRIG (число) { OFF }

STOP

число — целочисленное выражение от 0 до 4.

Как функция инструкция STRIG образует признак нажатия кнопки на рукоятке джойстика:

0 — выбор в качестве пусковой кнопки на клавиатуре клавиши "пробел";

1 — опрос состояния кнопки 1 джойстика 1;

2 — опрос состояния кнопки 1 джойстика 2;

3 — опрос состояния кнопки 2 джойстика 1;

4 — опрос состояния кнопки 2 джойстика 2.

Если кнопка нажата, то образуется значение 1, иначе — 0.

Как оператор инструкция STRIG управляет обработкой прерываний от кнопок джойстика. Предварительно он должен быть активизирован оператором ON STRIG GOSUB, иначе этот оператор не даст никакого эффекта. Например:

```
10 ON STRIG GOSUB 1000,2000,,4000:STRIG (0) ON
20 STRIG (1) ON : STRIG (3) ON}
:
1000 'эта часть работает при нажатии клавиши
     пробел
:
1100 RETURN
2000 'эта часть работает при нажатии кнопки 1
     джойстика1
:
2100 RETURN
4000 'эта часть работает при нажатии кнопки 2
     джойстика 1
```

4.5. Работа с временны́ми интервалами

Инструкции работы с прерываниями, временными задержками и т. п. представлены в табл. 4.11.

Таблица 4.11

Инструкция	Номер версии									
	1	2	3	4	5	6	7	8	9	10
DATE$										+
DELAY		+								
DISABLE		+								
ENABLE		+								
INTERVAL									+	
PAUSE			+							
TIME		+							+	
TIME$					+					+

DATE$ – переменная, которая хранит текущую дату

DATE$

Дата устанавливается при входе в интерпретатор автоматически и хранится в виде мм–дд–гггг, где мм – месяц; дд – день; гггг – год.
Например:

```
PRINT DATE$
07-11-89
DATE$="07-12-1989"
PRINT DATE$
07-12-89
```

DELAY – оператор, приостанавливающий выполнение программы на определенное время

DELAY выражение1[,выражение2[,выражение3]]

выражение1 – число минут;
выражение2 – число секунд;
выражение3 – сотые доли секунды.

Действие оператора DELAY прерывается по клавишам CTRL–S или abort. В этом случае интерпретатор переходит к обработке следующей инструкции после нажатия клавиши с любым другим символом. Разблокированные оператором ENABLE прерывания не активны во время действия оператора DELAY.

Если компьютер не стандартен, то оператор DELAY не будет работать до введения оператора TIME, который используется для калибровки.

В примере текущее время вводит пользователь, а затем текущее время печатается с интервалом 5 с.

Например:

```
10 INPUT "TIME" H,M,S
20 PRINT H;":";M;":";S
30 S=S+5
40 IF S>60 THEN GOSUB 100
50 IF M>60 THEN GOSUB 200
60 IF H>24 THEN H=H-24
70 DELAY 0,5
80 GOTO 20
100 S=S-60 : M=M+1 : RETURN
200 M=M-60 : H=H+1 : RETURN
RUN
TIME? 7,15,45
 7 : 15 : 45
 7 : 15 : 50
 7 : 15 : 55
 7 : 16 : 0
```

DISABLE – оператор блокировки прерываний

DISABLE номер строки

Блокирует прерывания, установленные заданным номером строки. Если номер строки не задан, то блокируются все прерывания.

Например:

```
DISABLE 10
```

Блокирует прерывание, которое было разблокировано в программной строке 10 оператором ENABLE. При попытке блокировать несуществующие прерывания появляется сообщение об ошибке EN.

Например:

```
10 ENABLE 20,5,&11111,&11100000,$
20 GOTO 100
30 PRINT "BET LAST! HERE HE COMES!"
40 DISABLE 10
50 RETURN
60 'START PROGRAMM
```

ENABLE – оператор, определяющий условие прерывания

```
ENABLE номер,порт,выражение1,[маска[,$]]
```

номер – номер строки подпрограммы;

порт – входной порт;

выражение1 – необходимое значение;

маска – накладываемая маска.

Условие прерывания должно проверяться перед выполнением каждого оператора в программе. Условие выполняется, если значение на входном порте соответствует значению выражения1, маскируемого маской. Параметр **маска** необязательный.

Если указан символ $, то условие выполняется при полном соответствии битов значения порта выражению 1. Например:

```
10 ENABLE 100,6,1,&11111110
20 GOTO 200
100 'ПОДПРОГРАММА ПОСЫЛКИ СЛЕДУЮЩЕГО БЛОКА НА
    ПЕЧАТЬ
:
190 RETURN
200 'ГЛАВНАЯ ПРОГРАММА
:
```

Главная программа выполняется до тех пор, пока бит 0 порта 6 не станет равным 1. Тогда программа прерывается и управление передается подпрограмме с номера 100.

INTERVAL – оператор разрешающий, запрещающий или задерживающий прерывания по таймеру

```
ON
INTERVAL {OFF}
STOP
```

Этот оператор управляет отсчетом реального времени для прерывания в регулярные интервалы. Но прежде надо выполнить оператор **ON INTERVAL-квант времени GOSUB номер строки**, чтобы проинформировать Бейсик-систему, как часто, после какого количества квантов времени (1 квант — 0,02 с), Бейсик-система должна прерывать работающую программу для выполнения специальной программы обработки прерывания.

В отличие от ON INTERVAL оператор INTERVAL ON не начинает и не прерывает процесса. Однако, если интервал и номер строки не заданы, оператор INTERVAL ON не работает. Когда выполняется оператор INTERVAL ON, то в заданном интервале в заданной строке начинается обработка прерывания. Оператор INLTRVAL OFF отключает обработку прерываний. Оператор INTERVAL STOP выполняет приостанов обработки прерывания до следующего оператора INTERVAL ON.

Используя INTERVAL STOP, можно отложить только одно прерывание, остальные будут потеряны. Надо помнить, что неявно INTERVAL STOP выполняется при входе в программу обработки прерываний и отменяется при выходе из нее. Этого можно избежать с помощью явных операторов INTERVAL ON и INTERVAL OFF в начале и в конце программы обработки прерывания.

Оператор INTERVAL ON/OFF/STOP позволяет контролировать количество одновременных событий в работающей программе.

Например:

```
10 ON INTERVAL=20 GOSUB 1000:INTERVAL ON
:
100 INTERVAL STOP:GOSUB 500:INTERVAL ON
:
1000 INTERVAL OFF
:
1090 INTERVAL ON:RETURN
```

PAUSE — команда установки временной задержки

PAUSE n

n — любое число от 0 до 65535.

Приостанавливает вычисления и отображает картинку в продолжении n кадров (с частотой 50 кадров в секунду). Максимальное значение n дает задержку до 22 мин. Паузу всегда можно прервать, нажав на любую клавишу

TIME — для MSX-BASIC — псевдопеременная, которая устанавливает или определяет системное время; для XYBASIC — оператор, который калибрует оператор DELAY

Формат 1 (для MSX-BASIC).

X=TIME или TIME=выражение

Формат 2 (для XYBASIC).

TIME

Формат 1. Псевдопеременная MSX-BASIC имеет внутренний 16-разрядный счетчик, значение которого изменяется с частотой 50 раз в секунду. Когда счетчик достигает значения 65535, он сбрасывается в 0.

Когда TIME стоит слева от выражения, которое может принимать значения от —32768 до 65535, устанавливается новое значение счетчика. Счетчик обновляется каждые 21,8 мин.

Счетчик используется для отсчета времени в операторе ON INTERVAL GOSUB.

Формат 2. Оператор TIME калибрует (формирует) оператор DELAY для систем, которые используют процессоры Z-80, 8085 и NEC8080. Интервал в 60 с используется как стандарт для выполняющейся последовательности операторов DELAY.

TIME$ – функция, которая формирует текущее время

TIME$

Выходная строка имеет вид: чч:мм:сс, где чч – часы; мм – минуты; сс – секунды

4.6. Работа в коммуникационной системе

Коммуникационная система – система, выполняющая вспомогательные функции, связанные с передачей информации между другими системами. При создании коммуникационных систем используются коммуникационные сети телефонные, телекоммуникационные или локальные).

В данном разделе будут рассмотрены операторы работы с коммуникационной сетью.

Коммуникационная сеть – это сеть передачи информации, образуемая множеством взаимосвязанных коммуникацинных модулей.

Для работы с коммуникационными сетями используются операторы COM(n) ON, COM(n) OFF, COM(n) STOP, ON COM(n) GOSUB, OPEN"COM...", которые реализованы в версии BASICA и оператор COM, который реализован в Бейсик-ПК8010.

4.6.1. Организация работы с файлом связи

Оператор OPEN"COM..." распределяет буфер для ввода-вывода таким же способом, как и оператор OPEN для файлов произвольного доступа.

Так как каждый адаптер связи открывается как файл, то все операторы ввода-вывода, которые действительны для файлов произвольного доступа, действительны для файлов связи.

Операторы последовательного ввода из файла связи такие же, как и для файлов произвольного доступа: INPUT# номер файла, LINE INPUT# номер файла, INPUT$.

Операторы последовательного вывода в файл связи такие же, как и для файлов произвольного доступа: PRINT# номер файла, PRINT# номер файла USING, WRITE# номер файла.

Операторы GET и PUT для файлов связи незначительно отличаются от операторов для файлов произвольного доступа. Они используются для ввода-вывода блоков фиксированной длины в файл или из файла связи. Вместо задания номера записи, которая должна быть считана или записана, задается число байтов, которые будут переданы в буфер или из буфера файла. Это число не может превышать значения, установленного параметром /S: в команде вызова интерпретатора.

При скорости обмена свыше 2400 бод (бит/с) необходимо приостановить передачу символа из другого компьютера на время, достаточное для обработки получаемого символа. Это может быть посылкой символа с кодом 19 в передающий компьютер, а затем символа с кодом 17 в этот же компьютер, когда приемный компьютер готов к получению информации. Символ с кодом 19 указывает передающему компьютеру остановить передачу информации, а символ с кодом 17 указывает на разрешение передачи.

Это общеиспользуемое соглашение, но оно не универсальное и зависит от протокола обмена, реализованного в данной коммуникационной системе.

Для работы с файлом связи существуют три встроенные функции, которые определяют окончание передачи информации:

LOC(X) – определяет число символов в буфере ввода-вывода; если число больше 255, то функция LOC возвращает значение, равное 255;

LOF(X) – определяет количество свободного пространства во входном буфере; размер буфера связи устанавливается параметром /C:n в команде вызова интерпретатора; по умолчанию n равен 256;

EOF(X) – определяет конец данных в буфере ввода и возвращает значение –1, если буфер ввода пустой, или 0, если в буфере есть символы.

Переполнение буфера связи может встретиться при попытке чтения после заполнения входного буфера, т. е. LOF(X) возвращает 0.

В следующем примере указаны наиболее эффективные операторы для считывания содержимого файла связи.

```
10 WHILE NOT EOF(1)
20 A$=INPUT$(LOC(1),#1
:
обработка данных строки A$
:
100 WEND
```

4.6.2. Описание инструкций

COM – оператор занесения сообщения в "почтовый ящик" локальной сети

```
COM "сообщение"
```

сообщение – последовательность символов длиной до 240.

Используется для установления связи ученического компьютера с компьютером преподавателя.

COM(n) – оператор обработки прерываний при работе с коммуникационной сетью

```
COM (n) ON

COM (n) OFF

COM (n) STOP
```

n – номер адаптера связи (1 или 2).

Для разрешения прерывания оператором ON COM (n) должен быть выполнен оператор COM (n) ON. После этого, если в операторе ON COM (n) задан ненулевой номер строки, будет происходить обработка прерывания.

Если был задан оператор COM (n) OFF, то прерывание не обрабатывается, даже если оно и произошло.

Если был задан оператор COM (n) STOP, то прерывание не обрабатывается, но произошедшее событие запоминается для последующей обработки, когда будет задействован оператор COM (n) ON.

ON COM (n) – определяет подпрограмму обработки прерывания от коммуникационной сети

```
ON COM(n) GOSUB номер строки
```

n – номер адаптера связи (1 или 2).

Если задан нулевой номер строки, то прерывание блокируется.

Перед заданием оператора ON COM (n) GOSUB для адаптера n должен быть выполнен оператор COM (n) ON. Если в операторе ON COM (n) GOSUB задан ненулевой номер строки, то каждый раз происходит проверка на наличие символов в определенном адаптере связи и если символы в буфере есть, то будет вызываться подпрограмма по заданному номеру строки.

Если был выполнен оператор COM (n) OFF, то ловушка прерывания от адаптера блокируется и прерывание не запоминается, даже если оно и произошло.

После выполнения оператора COM (n) STOP ловушка прерываний для адаптера блокируется, а если получен символ, то это событие запоминается, и после выполнения оператора COM (n) ON ловушка срабатывает.

Не рекомендуется использовать ловушки прерываний от адаптера связи для односимвольного сообщения.

OPEN – оператор, открывающий файл связи

```
OPEN"COM n:[скорость][,четность][,данные][,останов]" AS [#]
                        номер файла
```

n – номер адаптера связи (1 или 2);

скорость – целочисленная константа, определяющая передачу или получение отношения (бит/секунда); значения скоростей: 75, 110, 150, 300, 600, 1200, 1800, 2400, 4800 и 9600 бод (по умолчанию 300 бод);

четность – односимвольная константа, определяющая четность для передачи и получения:

S – (SPACE) бит четности всегда передается и получается как пробел (бит 0);

O – (ODD) нечетная передача, нечетное получение;

M – (MAPK) бит четности всегда передается и получается как флаг (метка)(бит 1);

E – (EVEN) четная передача, проверка, четное получение;

N – (NONE) не передается и не проверяется бит четности; по умолчанию принимается значение E (EVEN);

данные – целочисленная константа, указывающая число битов переданных или полученных данных; принимаемые значения от 4 до 8 (по умолчанию 7);

останов – целочисленная константа, указывающая число битов параметра останов; принимаемые значения 1 или 2; по умолчанию два бита останова для 75 и 110 бод, один бит останова – для всех других скоростей;

номер файла – целочисленное выражение, которое затем будет использовано в операторах ввода-вывода информации в файл или из файла связи.

Оператор OPEN"COM..." при открытии организовывает буфер ввода-вывода так же, как это делает оператор OPEN для обмена с дисками. Для работы оператора требуется адаптер асинхронной связи с другими компьютерами на базе RS232.

Устройство связи может работать одновременно только с одним файлом. Если асинхронный адаптер связи не установлен в соответствии с форматом данной системы, то появляется сообщение об ошибке.

Например:

```
5 TRIES=6
10 ON ERROR GOTO 100
20 OPEN"COM1:300,N,8,2,CS,DS,CD10000" AS #1
30 ON ERROR GOTO 100
40 CLOSE #1
50 GOTO 1000
:
100 TRIES=TRIES-1
110 IF TRIES=0 THEN ON ERROR GOTO 100
120 RESUME
:
1000 OPEN"COM1:300,N,8,2,CS,DS,CD2000" AS #2
```

Глава 5

Работа с программами, написанными на языке Бейсик, и средства отладки программ

5.1. Ввод новой программы

В гл. 2 было сказано, что программная строка состоит из номера строки, за которым следуют элементы языка. Программные строки могут вводиться в любом порядке, но их выполнение будет осуществляться в порядке увеличения номеров строк, не считая безусловных переходов и вызовов подпрограмм. Для автоматической нумерации программных строк рекомендуется использовать команду AUTO, что позволит сократить время ввода программы (рис. 5.1).

Для добавления новой строки к программе нужно ввести новый номер строки, за которым следует хотя бы один символ (не пробел), и нажать клавишу возврата каретки. Строка будет сохраняться в памяти как часть программы. Если строка с таким номером уже существует, то старая строка стирается и заменяется новой.

Программные строки не проверяются на наличие синтаксических ошибок перед добавлением к программе. Они проверяются только при выполнении программы. Для того чтобы легко можно было вставить строки, рекомендуется нумеровать их с шагом, не равным единице, а например десять или сто. Шаг (или инкремент) выбирается произвольно и необязательно должен быть постоянным по всей программе. Для перенумерации строк, если уже невозможно вставить программную строку или по каким-либо другим причинам, следует использовать команду RENUM.

При работе с интерпретаторами версий языка Бейсик в операционной среде CP/M и МикроДОС ошибку во время ввода строки можно исправить до нажатия клавиши возврата каретки следующими способами:

удалить всю строку, нажав клавиши CTRL—U;

удалить символ, нажав клавиши Del, Backspace или CTRL—H.

Чтобы удалить уже существующую программную строку, надо набрать номер удаляемой строки и нажать клавишу возврата каретки или использовать команду DELETE (для Бейсик-АГАТ команду DEL).

Для редактирования уже существующей программной строки можно использовать команды редактора или ввести строку заново под тем же номером.

Для вывода программных строк на экран дисплея или печатающее устройство используются команды LIST и LLIST соответственно. Любую введенную или отредактированную программу необходимо запомнить на НГМД или НМЛ.

Если программа или часть программы уже существует на НГМД или НМЛ, то сначала она загружается в память ЭВМ. После этого можно продолжать ввод программы, добавлять, удалять или редактировать программные строки.

Для ввода новой программы, которой еще нет на диске или на магнитной ленте, необходимо очистить память командой NEW перед вводом программных строк новой программы. Если этого не сделать, то вводимые программные строки будут замещать программные строки с этими номерами уже загруженной в память программы.

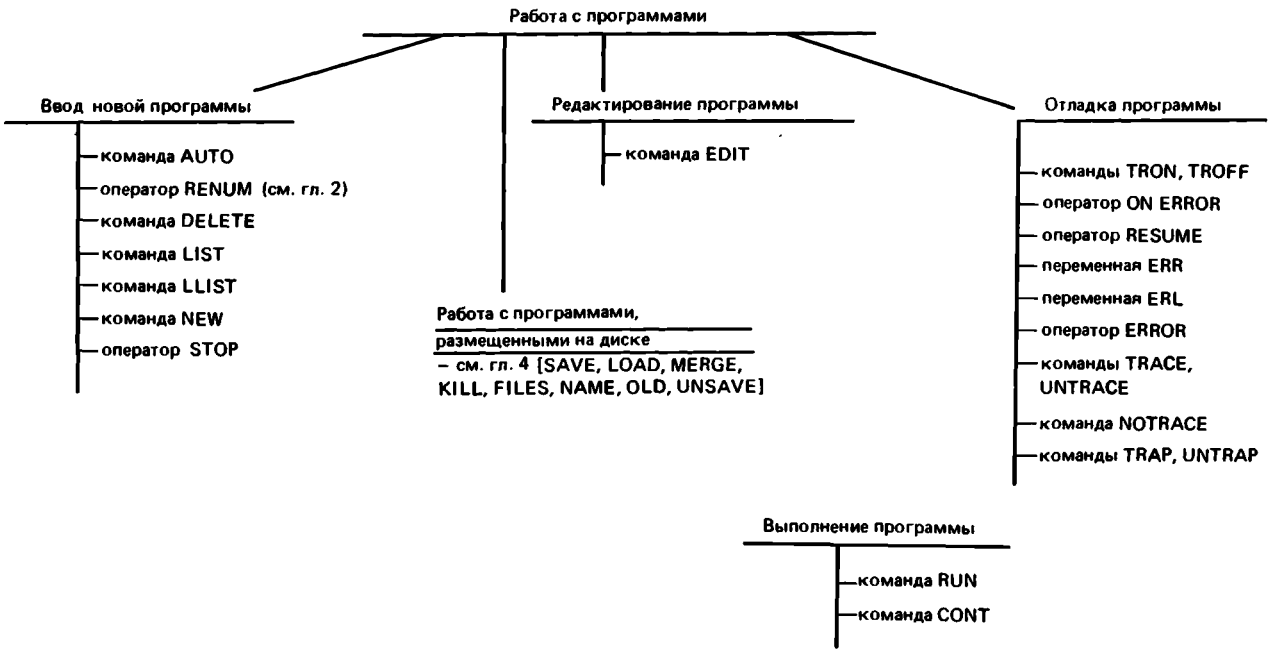

Работа с программами
Ввод новой программы
команда AUTO
оператор RENUM (см. гл. 2)
команда DELETE
команда LIST
команда LLIST
команда NEW
оператор STOP
Редактирование программы
команда EDIT
Работа с программами, размещенными на диске – см. гл. 4 [SAVE, LOAD, MERGE, KILL, FILES, NAME, OLD, UNSAVE]
Отладка программы
команды TRON, TROFF
оператор ON ERROR
оператор RESUME
переменная ERR
переменная ERL
оператор ERROR
команды TRACE, UNTRACE
команда NOTRACE
команды TRAP, UNTRAP
Выполнение программы
команда RUN
команда CONT

Рис. 5.1

5.2. Редактирование программ

Редактирование программы — это процесс изменения текста программы, в котором могут использоваться различные режимы редактирования. Практически все современные версии языка Бейсик содержат средства для редактирования программ, т. е. средства, используемые для изменения отдельных символов или программных строк. Эти средства, как правило, входят в состав встроенного редактора, который обеспечивает выполнение следующих функций:

управления курсором;

ввода текста;

удаления текста;

добавления текста в конец логической строки (т. е. строки программы, а не экрана);

поиска текста;

замены текста;

управления режимом редактирования.

Различные версии встроенного редактора могут по-разному обеспечивать выполнение перечисленных выше функций; при этом полнота выполнения этих функций и способы обращения к ним отличают версии друг от друга.

Рассматриваемые версии языка Бейсик имеют два типа встроенного редактора: командный (строчный) и экранный. Данные о наличии редакторов приведены в табл. 5.1, где представлены также команды вызова редактируемой строки.

Таблица 5.1

Версия	Редактор		Команда
	Командный	Экранный	
XYBASIC	+		EDIT
Бейсик-Спектрум+2		+	LIST
Бейсик-TRS-80	+		EDIT
MBASIC	+		EDIT
Бейсик-ПК8010	+		EDIT
Бейсик-ПК8020	+		EDIT
MSX-BASIC		+	LIST
BASICA	+	+	LIST EDIT
Бейсик-АГАТ		+	LIST

5.2.1. Командный (строчный) редактор

Введем следующие обозначения:

<символ> – произвольный символ или произвольная клавиша;

<текст> – строка символов произвольной длины;

[n] – целое число, по умолчанию единица.

Редактор XYBASIC

<возврат каретки> – заканчивает процесс редактирования, печатаются символы справа от курсора, программная строка добавляется к текущей программе; возврат в прямой режим с подсказкой OK (см. гл. 2);

<печатный символ> – любой <печатный символ> удваивается (эхо) и вводится в редактируемую строку в текущую позицию курсора; курсор будет находиться в позиции справа от введенного символа; если строка не содержит больше символов, то вместо символа будет удваиваться <Ctrl–G>;

<Rubout> – удаляет символ слева от курсора, удаленный символ изображается в слешах (/и/);

<Ctrl–B> – выход из XYBASIC и возврат в операционную систему или монитор;

<Ctrl–C> – выход из системы редактирования и возврат в прямой режим, оставляя неизменным предыдущее содержимое строки.

<Ctrl–D> – удаляет символ справа от курсора; стертые символы не печатаются;

<Ctrl–E> – позволяет использовать возможности редактирования всякий раз, как вводится строка в XYBASIC; если <Ctrl–E> введен как первый символ строки, то XYBASIC входит в редактор с содержимым самой последней определенной строки; эта особенность полезна для коррекции ошибок при работе в командном режиме и при вводе данных; если <Ctrl–E> вводится после первого символа строки, то XYBASIC входит в редактор с символами, предшествующими <Ctrl–E>;

<Ctrl–F><печатный символ> – посылает курсор вправо до следующей встречи <печатного символа> в редактируемой строке, печатая все символы, которые прошел курсор; если остаток строки не содержит <печатного символа>, то XYBASIC удваивает <Ctrl–G> и оставляет позицию курсора неизменной; исследуемый символ не печатается;

<Ctrl–G> – вызывает звуковой сигнал;

<Ctrl–H> – возврат на одну позицию; стирает символ слева от курсора;

<Ctrl–K> – стирает все символы справа от курсора.

<Ctrl–L> – выводит на экран символы, оставшиеся справа от курсора в редактируемой строке, и переводит курсор в начало следующей строки;

<Ctrl–N> – находит следующий <печатный символ>, заданный последним в команде <Ctrl–F>; удваивает <Ctrl–G> и оставляет позицию курсора неизменной, если команда <Ctrl–F> не задана или строка не содержит <печатного символа>;

<Ctrl–R> – выводит на экран символы справа от курсора в редактируемой строке, а затем переводит курсор в начало следующей строки и выводит на экран символы слева от позиции курсора, в которой он находился в предыдущей строке, когда была нажата клавиша;

<Ctrl–T> – пересылает курсор вправо;

<Ctrl–U> – уничтожает текущее содержимое редактируемой строки и начинает редактирование заново с первоначального содержимого строки, позволяя легко устранять ошибки редактирования.

Команды управления курсором.

<пробел> – нажатие клавиши <пробел> пересылает курсор вправо. Нажатие клавиши <пробел> n раз пересылает курсор на n позиций вправо. После каждого нажатия клавиши <пробел> появляется символ редактируемой строки

<Del> – В режиме редактирования нажатие клавиши <Del> n раз пересылает курсор на n позиций влево

Команды ввода текста.

I<текст><Escape> – Вводит текст с текущей позиции курсора. Вводимые символы печатаются на экране дисплея. Чтобы закончить ввод, нажимается клавиша <Escape>. Если во время команды ввода нажимается клавиша возврата каретки, то прекращается ввод текста и редактирование строки. Если при вводе символа общая длина строки превышает 255 символов, то символ не печатается и выдается звуковой сигнал

X<текст><Escape> – Ввод текста в конец редактируемой строки. Команда X пересылает курсор в конец строки и включает режим ввода. Нажатие клавиши <Escape> или возврата каретки заканчивает ввод

Команды удаления текста.

[n]D – Удаляет n символов справа от курсора. Удаленные символы печатаются в обратных слешах, и курсор располагается справа от последнего удаленного символа. Если справа от курсора меньше, чем n символов, то удаляется вся оставшаяся строка

H<текст><Escape> – Удаляет все символы справа от курсора, а затем автоматически включает режим ввода

Команды поиска текста.

[n]S<символ> – Находит позицию n-го включения символа в строке и оставляет курсор перед ним. Символ в текущей позиции курсора не рассматривается. Если заданный символ не найден, то курсор останавливается в конце строки. Все символы, рассмотренные за время проверки, печатаются

[n]K<символ> – Подобна команде S, за исключением того, что все рассмотренные во время проверки символы удаляются. Курсор устанавливается перед заданным символом, и все стертые символы печатаются в обратных слешах

Команды замены текста. Замена текста производится в режиме строчного редактирования командой C.

[n]C<текст> – Изменяет следующие n символов на <текст>. <Текст> должен содержать n символов. После того, как n-й новый символ напечатан, осуществляется возврат в режим редактирования

Команды управления режимом редактирования.

<возврат каретки> – Печатает остаток строки, сохраняет сделанные изменения и возвращает управление на уровень команд

E – Сохраняет сделанные изменения и возвращает управление в режим редактирования. Остаток строки не печатается

Q – Возвращает управление на уровень команд без сохранения изменений, которые были сделаны во время редактирования

L – Печатает остаток редактируемой строки и устанавливает курсор в начало строки, оставаясь в режиме редактирования

A – Позволяет начать редактирование сначала. Запоминает первоначальную строку и восстанавливает курсор в начало строки. Редактирование производится только на экране

дисплея или на экране и в памяти в зависимости от того, как была введена строка перед нажатием Ctrl—A

После нажатия клавиш Ctrl—A номер редактируемой строки будет появляться после вспомогательного знака ! и пробела. Если строка была введена с клавиатуры, то после нажатия клавиш Ctrl—A она будет редактироваться как на экране, так и в памяти.

Например:

```
400 NEXT
(нажатие "Ctrl-A")
! 400 NEXT [IJ,K]<возврат каретки>
LIST 400
400 NEXT J,K
```

Если Ctrl—A используется после распечатки программы, то редактируется последняя строка программы и причем только на экране, а не в памяти.

Например:

```
LIST 200
200 PRINT 5
Ok
(нажатие "Ctrl-A")
! 200 PRINT [D\5\[I8$L]<возврат каретки>
200 PRINT 8
```

Если еще раз ввести команду LIST 200, то на экране все равно появится строка

```
200 PRINT 5
Ok
```

В приведенных выше примерах в квадратных скобках действия оператора ЭВМ, а символ $ — нажатие клавиши <Escape>.

С помощью Ctrl—A можно редактировать и вызывать не только программные строки, но и любые директивы, вводимые с клавиатуры.

Например:

```
PRINT "Проверка"
Проверка
(нажатие "Ctrl-A")
! PRINT "Проверка"
```

5.2.2. Экранный редактор

Экранный редактор позволяет пользователю вводить строки программы обычным путем, а затем редактировать полный экран до внесения изменений в оперативную память. Эта ускоряющая работу возможность обеспечивается с помощью специальных функциональных клавиш, управляющих перемещением курсора, режимом вставки или удаления текста и т. д.

С помощью экранного редактора пользователь может быстро передвигать курсор по экрану, внося при необходимости исправления. Строка программы не изменится до тех пор, пока не будет нажата клавиша возврата каретки при нахождении курсора в этой строке.

Чтобы отредактировать программу, необходимо поместить ее или часть ее строк на экран с помощью команды LIST.

Функции экранного редактора существенно зависят от аппаратной реализации клавиатуры и программной организации интерпретатора языка Бейсик, поставленного на микроЭВМ.

Для ввода редактора из начального меню выбирается функция "BASIC" с помощью клавиш управления курсором и клавиши <ENTER>.

Экран будет выглядеть следующим образом: бело-голубой курсор будет установлен в левом верхнем углу экрана. В нижней части экрана имеется черная полоса. Она называется полосой подсказки и сообщает о том, какая часть встроенного программного обеспечения используется. В данный момент полоса подсказки будет содержать "Бейсик-128", поскольку именно так называется редактор. Между полосой подсказки и нижней границей экрана образуется часть экрана, которая называется малым экраном. Эта часть экрана содержит место только для двух строк текста и наиболее часто используется при обнаружении ошибки и выводе сообщения об ошибке.

Теперь нажимается клавиша <EDIT>. Курсор исчезнет и появится новое меню — меню редактирования. Меню редактирования содержит следующие функции:

Бейсик-128 — удаляет меню редактирования и восстанавливает курсор

RENUM — (Перенумерация) Перенумеровывает номера программных строк с шагом 10. Программы всегда будут начинаться с номера 10. Если перенумерация невозможна, то раздается короткий низкочастотный сигнал и меню исчезает. Для того чтобы перенумеровать программу с шагом отличным от 10 и начальным номером строки отличным от 10, используются следующие команды:

```
LET start=5:LET stepsize=2:
LET histart=INT(start/256):
LET histep=INT(stersize/256):
POKE 23444,start-256*histart:
POKE 23445,histart:
POKE 23446,stepsize-256*histep:
POKE 23447,histep
```

Вследствие изменения значения переменных start и stepsize команда RENUM будет выполнять перенумерацию, начиная с любого (допустимого) номера строки и с любым шагом. Введите указанную команду, а затем используйте команду RENUM

SCREEN — (Экран) Перемещает курсор в нижнюю часть экрана и обеспечивает ввод и редактирование команд языка в этой части экрана. Любые операции редактирования в нижней части экрана не портят верхнюю часть экрана, что особенно удобно при работе с графическими средствами. Для возврата к работе с верхней частью экрана снова выберите функцию SCREEN в меню редактирования

PRINT — (Печать) Распечатывает листинг текущей программы, если подключено печатающее устройство. После завершения печати листинга меню исчезает с экрана, а курсор возвратится в прежнюю позицию. Если печатающее устройство не подключено или находится в состоянии НЕ ГОТОВ, то при двукратном нажатии клавиши <BREAK> будет осуществлен возврат в режим редактирования

EXIT — (Выход) Осуществляет возврат в начальное меню, сохраняя в памяти любую программу, с которой в данный момент осуществлялась работа. Если необходимо снова вернуться к программе, то выбирается функция "Бейсик-128". При выборе функции "Бейсик-48", включении компьютера или нажатии клавиши <RESET>, всякая программа, находящаяся в памяти, будет стерта.

При вводе программы, если строка окажется введенной строчными буквами и цвет курсора изменился на красный, то это означает, что обнаружена ошибка. Строка должна быть

исправлена перед тем, как она будет принята компьютером. При коррекции строки используются клавиши управления курсором для подведения курсора к части строки, которую необходимо исправить. Затем можно вводить любые символы, которые требуется вставить, или, используя клавишу <DELETE>, удалять лишние. После того, как закончена коррекция строки, нажимается клавиша <ENTER>.

Редактор MSX-BASIC

В табл. 5.2 приводятся шестнадцатеричные коды для управляющих функций, а также последовательности, полученные при нажатии клавиши Control (Ctrl) и других клавиш, соответствующих этим функциям. Они максимально соответствуют коду ASCII. Некоторые управляющие функции далее (стр. 149) описаны более подробно. В описании используются понятия физическая и логическая строка.

Строка, появляющаяся на экране, называется физической строкой, а строка программы в оперативной памяти от 1 до 255 символов — логической.

Таблица 5.2

Шестнадцатеричный код	Клавиша Ctrl нажата	Специальная клавиша	Функция
01	A		Заголовок графического символа
* 02	B		Перемещение курсора к началу предыдущей строки
* 03	C		Прерывание, когда система MSX-BASIC ожидает ввода
* 04	D		Игнорируется
* 05	E		Усечение строки (очистка до конца логической строки)
* 06	F		Перемещение курсора к началу следующего слова
* 07	G		Звуковой сигнал
08	H	Back space	Удаление символа слева от курсора
09	I	TAB	Табуляция (8 символов)

Продолжение табл. 5.2

Шестнадцатеричный код	Клавиша Ctrl нажата	Специальная клавиша	Функция
* 0A	J		Перевод строки
* 0B	K	HOME	Установка курсора в верхний левый угол экрана
* 0C	L	CLS	Очистка экрана
* 0D	M	RETURN	Возврат каретки, ввод логической строки
* 0E	N		Дополнение в конец строки
* 0F	O		Игнорируется
* 10	P		-"-
* 11	Q		-"-
* 12	R	INS	Вставка символа
* 13	S		Игнорируется
* 14	T		-"-
* 15	U		Удаление логической строки
* 16	V		Игнорируется
* 17	W		-"-
* 18	X	SELECT	-"-
* 19	Y		-"-
* 1A	Z		-"-
* 1B	[	ESC	-"-
* 1C		-->	Курсор вправо
* 1D	]	<--	Курсор влево
* 1E	^		Курсор вверх

Окончание табл. 5.2

Шестнадцатеричный код	Клавиша Ctrl нажата	Специальная клавиша	Функция
* 1F			Курсор вниз
7F	DELETE	DEL	Удаление символа под курсором. Сдвиг текста влево

Примечание. Коды, отмеченные символом "*", выводят из режима вставки, если происходила работа в этом режиме.

Предыдущее слово – Курсор смещается влево к предыдущему слову. Предыдущим словом считается знак слева от курсора, принадлежащий одному из следующих множеств: A–Z; А–Я; a–z; а–я; 0–9

Прерывание – Возвращает MSX-Бейсик в режим прямого управления при нажатии клавиши RETURN, без сохранения изменений, внесенных в редактируемую логическую строку

Усечение – Все символы логической строки, начиная от текущего положения курсора до конца логической строки, удаляются

Следующее слово – Курсор передвигается вправо к следующему слову. Следующее слово – знак справа от курсора, принадлежащий одному из множеств: A–Z; А–Я; a–z; а–я; 0–9

Звуковой сигнал – Слышен звуковой сигнал, такой же, как и при выполнении оператора BEEP

Стирание последнего символа – Удаляет знак слева от курсора. Все знаки от курсора перемещаются влево на одну позицию. Последующие знаки и строки в логической строке передвигаются вверх

Табуляция – В режиме вставки функция TAB вводит пробелы с обозначенной курсором позиции до следующей позиции табуляции. Происходит раздвижка строк с переходом на следующую строку. В режиме, отличном от режима вставки, функция TAB перемещает курсор к следующей позиции табуляции через 8 позиций

Перевод курсора в исходное положение – Курсор перемещается в верхний левый угол экрана. Экран не очищается

Очистка экрана – При нажатии этой клавиши курсор переходит в исходное положение и очищает весь экран, независимо от исходной позиции курсора

Возврат каретки – Вставляет код "возврат каретки" в текущую позицию курсора. Нажатие этой клавиши указывает Бейсику на логический конец этой строки. Курсор перемещается в начало следующей логической строки. Если на экране недостаточно места, то все изображение сдвигается вверх и снизу добавляется пустая строка

Дополнение в конец строки – Переводит курсор в конец строки, остающиеся знаки в строке не стираются. С новой позиции все внесенные знаки добавляются к логической строке, пока не произойдет нажатие клавиши "возврат каретки"

Вставка – Триггерный переключатель для режима вставки. При режиме вставки размер курсора уменьшается и, начиная с данной позиции, происходит вставка символов. При вставке новых символов знаки справа от курсора перемещаются вправо. Соблюдается непре-

рывный переход на новые строки. По окончании режима вставки размер курсора вновь увеличивается

Очистка логической строки – При нажатии этой клавиши при курсоре в любой части строки удаляется вся логическая строка

Курсор вправо – Курсор перемещается вправо на одну позицию. Соблюдается непрерывный переход на следующие строки

Курсор влево – Курсор перемещается влево на одну позицию. Соблюдается непрерывный переход на предыдущие строки

Курсор вверх – Курсор переводится вверх на одну физическую строку (в данной колонке)

Курсор вниз – Курсор переводится вниз на одну физическую строку (в данной колонке)

Редактор BASICA и EC1840

Для машин фирмы IBM ниже представлены специальные команды (клавиши) редактора и описаны функциональные возможности этого полноэкранного редактора.

Специальные клавиши редактора.

Home – (Числовая клавиша 7) пересылает курсор в верхнюю левую позицию экрана

CTRL–Home – Очищает экран. Устанавливает курсор в верхнюю левую позицию экрана

↑ – (Курсор вверх – числовая клавиша 8) Пересылает курсор на одну позицию вверх

↓ – (Курсор вниз – числовая клавиша 2) Пересылает курсор на одну позицию вниз

← – (Курсор влево – числовая клавиша 4) Пересылает курсор на одну позицию влево. Если курсор достиг левой границы экрана, то он пересылается в самую правую позицию предыдущей строки

→ – (Курсор вправо – числовая клавиша 6) Пересылает курсор на одну позицию вправо. Если курсор достиг правой границы экрана, то он пересылается в самую левую позицию следующей строки

CTRL– → – (Следующее слово) Пересылает курсор вправо до следующего "слова". "Слово" определяется как символ или группа символов, которые начинаются с буквы или цифры. Слова разделяются пробелами или специальными символами

CTRL– ←**–** (Предыдущее слово) Пересылает курсор влево до предыдущего слова

End – (Числовая клавиша 1) Пересылает курсор в конец логической строки. Символы, введенные начиная с этой позиции, добавляются в конец строки

CTRL–End – Удаляет символы от текущей позиции до конца логической строки. Все физические строки экрана удаляются до тех пор, пока не будет найден символ конца ввода

Ins – (Числовая клавиша 0) Устанавливает режим Ins (ввода символов). Если этот режим не установлен, то нажатие этой клавиши будет включать его. Режим Ins отмечается мерцанием курсора, определяя нижнюю половину символьной позиции. Очередной вводимый символ помещается в позицию, на которую указывает курсор, все символы, расположенные справа от курсора, смещаются вправо на одну позицию. Если последний символ вытесняется за пределы строки экрана, то он помещается в крайнюю левую позицию следующей строки. Когда режим Ins выключен, вводимые символы будут замещать в строке существующие символы

Del – (Числовая клавиша – десятичная точка) Удаляет символ с текущей позиции. Все символы справа от удаленной позиции пересылаются на одну позицию влево

← – (Backspace – возврат на одну позицию) Удаляет символ слева от курсора. Все символы справа от удаленного пересылаются влево на одну позицию

ESC – При нажатии ESC, когда курсор находится где-либо в строке, с экрана удаляется эта физическая строка. Строка не передается в интерпретатор Бейсик для обработки. Если она является программной строкой, то из памяти не удаляется

CTRL–Break – Возвращает редактор в командный режим без сохранения каких-либо изменений, сделанных при редактировании текущей строки, не удаляя строку с экрана, подобно ESC

TAB – (Табуляция) Пересылает курсор в следующую установку табуляции, которая встречается каждые восемь символьных позиций

Коррекция текущей строки при работе с экранным редактором MSX-BASIC и BASICA

В режиме уровня команды любая строка будет обрабатываться встроенным редактором. Бейсик всегда находится на уровне команды после напоминания Ok.

Изменение символов. Используя клавиши пересылки курсора, курсор пересылается в позицию, где обнаружена ошибка, и вводится нужный символ. Затем можно переслать курсор в конец строки, используя клавиши "курсор вправо" или "End", и продолжить ввод.

Удаление символов. Если в строке необходимо удалить символ, то используется клавиша Del. С помощью клавиши пересылки курсора курсор передвигается на символ, который надо удалить. Символ удаляется нажатием клавиши Del. Затем оператор пересылает курсор обратно в конец строки. Если неправильный символ был только что введен, то его можно удалить, используя клавишу "Backspace". Затем можно продолжить ввод строки.

Добавление символов. Для добавления пропущенных символов надо переслать курсор в позицию, с которой необходимо ввести новые символы. Нажмите клавишу "Ins" для установления режима ввода символов. Введите символы, которые надо добавить, эти символы будут вставлены, начиная с этой позиции. Символы слева от курсора остаются неизменными. Символы справа от курсора будут сдвигаться вправо по мере введения новых символов. Режим ввода автоматически отключается при нажатии любой клавиши пересылки.

Удаление части строки. Чтобы удалить часть строки, начиная с текущей позиции курсора, введите CTRL–End, за которым следует Enter.

Аннулирование строки. Чтобы аннулировать строку, которая находится в обработке, надо нажать клавишу Esc где-либо в строке. Нельзя нажимать клавишу возврата каретки. Это приведет к удалению входной логической строки.

Введение или изменение программы. Любая введенная строка, которая начинается с номера, рассматривается как строка программы. Строки программы могут содержать максимум 255 символов, включая возврат каретки. Если строка содержит больше 255 символов, то лишние символы игнорируются при нажатии клавиши возврата каретки. Хотя эти символы появляются на экране, они не обрабатываются Бейсиком.

Ключевые слова и имена переменных должны быть записаны прописными буквами. Однако их можно вводить как комбинации прописных и строчных букв, редактор программ будет переводить их в прописные за исключением комментариев и строк, заключенных в кавычки.

Добавление новой строки к программе. Введите номер строки (0 – 65529), за которым следует хотя бы один символ – не пробел и не возврат каретки. Строка будет сохраняться как часть программы в памяти. Если строка с этим номером уже существует, то старая строка замещается новой. Программные строки не проверяются на наличие синтаксических ошибок перед добавлением к программе. Они проверяются только при выполнении программы.

Замещение или изменение существующей программной строки. Существующая строка изменяется, когда номер вводимой строки уже существует в программе. Старая строка замещается новой.

Удаление программных строк. Чтобы удалить существующую строку, сразу за номером этой строки надо нажать клавишу возврата каретки. Чтобы удалить одну строку или группу строк, можно использовать команду DELETE. При попытке удалить несуществующую строку появляется сообщение Undefined line number (Неопределенный номер строки). Нельзя использовать клавишу Esc для удаления программной строки, так как нажатие клавиши Esc удаляет строку только с экрана.

Удаление введенной программы. Чтобы удалить введенную программу, находящуюся в памяти, надо ввести команду NEW. Команда NEW обычно используется для очистки памяти перед вводом новой программы.

Работа с новой строкой. Редактируя любую строку на экране, можно использовать клавиши пересылки курсора в позицию, выбранную для изменения. Затем можно использовать средства для изменения, добавления или удаления символов в строке. Для вывода программных строк на экран можно применить команду LIST. Для этого курсор устанавливается в позицию в строке, которая будет редактироваться; эта позиция изменится.

Чтобы запомнить отредактированную строку, следует нажать клавишу возврата каретки. Можно продублировать строку в программе следующим способом: переслать курсор в строку, которая будет дублироваться, изменить номер строки. Затем нажать клавишу возврата каретки; обе строки, старая и новая, будут в программе.

Программная строка никогда не изменится до тех пор, пока не будет нажата клавиша возврата каретки. Перед нажатием клавиши возврата каретки, курсор надо перевести в конец логической строки. Редактор программы "знает", где заканчивается каждая логическая строка и обрабатывает полную строку, даже если клавиша возврата каретки Enter нажимается в начале строки. Чтобы сохранить программу с новыми изменениями, следует использовать команду SAVE (см. гл. 4) перед введением команды NEW.

Синтаксические ошибки. Если во время выполнения программы обнаруживается синтаксическая ошибка, то интерпретатор автоматически распечатывает строку, которая вызвала ошибку, и ее можно корректировать.

Например:

```
10 A=2$45
RUN
? Syntax error in 10
10 A=2$45
```

Редактор распечатывает строку с ошибкой и устанавливает курсор в позицию первой цифры номера строки. Нужно переслать курсор вправо до знака "$" и изменить его на ".", затем ввести возврат каретки. Скорректированная строка запоминается в программе.

5.3. Работа программиста по исправлению ошибок

5.3.1. Средства отладки программ

Обычно программисты тратят около 40% рабочего времени на написание программы, а 60% — на отладку программы. Встроенные средства отладки позволяют обрабатывать некоторые ошибочные ситуации, а также выводить на терминал номера выполняемых строк программы.

Важной сервисной функцией, обеспечивающей отладку программы, обладает команда, включающая трассировку программы. С помощью трассировки выполнение программы может быть проверено строка за строкой, что позволяет определить точное местонахождение ошибки. Трассировка включается командой TRON (в некоторых версиях – TRACE) и выключается командой TROFF (UNTRACE или NOTRACE). После команд TRON(TRACE) и RUN на экране дисплея появляются номера строк каждого исполнительного оператора и выводимые значения переменных и констант.

В XYBASIC команда BREAK позволяет установить точку приостанова по номеру строки или переменной. Если установить точку приостанова по номеру строки, то номер строки будет печататься каждый раз, когда строка будет выполняться, а затем или продолжается выполнение программы, или выполнение программы завершается. Если установить приостанов по имени переменной, то будет печататься имя переменной и ее новое значение всякий раз при ее изменении. Команда UNBREAK удаляет точку приостанова. Сведения о командах для разных версий языка представлены в табл. 5.3.

Таблица 5.3

Версия	Команды трассировки	Скобки	Приостанов	Ловушки	Примечание
Стандарт	TRON TROFF			ERL, ERR, ERROR, ON ERROR	
XYBASIC	TRACE UNTRACE	[ ]	BREAK UNBREAK	TRAP UNTRAP	
Бейсик-АГАТ	TRACE NOTRACE			ONERR	**
Бейсик-TRS-80	TRON TROFF	< >		ERL, ERR, ERROR, ON ERROR	*
MBASIC	TRON TROFF	[ ]		ERL, ERR, ERROR, ON ERROR	*
Бейсик-ПК8010	TRON TROFF	[ ]		ERL, ERR, ERROR, ON ERROR	*
Бейсик-ПК8020	TRON TROFF	[ ]		ERL, ERR, ERROR, ON ERROR	*
MSX-BASIC	TRON TROFF	[ ]		ERL, ERR, ERROR, ON ERROR	*
BASICA	TRON TROFF	[ ]		ERL, ERR, ERROR, ON ERROR	*

Окончание табл. 5.3

Примечание. * — При трассировке номер строки печатается только один раз, даже если эта строка содержит несколько операторов. ** — При трассировке номер строки печатается столько раз, сколько операторов содержит эта программная строка.

5.3.2. Ошибки

В рассматриваемых версиях интерпретаторов при появлении сообщения об ошибке возможны различные ситуации. Одни ситуации приводят к прерыванию работы программы, другие прерывания не вызывают.

Ошибки, препятствующие выполнению программы, можно разделить на несколько групп:

Группа 1. Синтаксические ошибки. В операторах PRINT, INPUT или LET может быть:

забыта одна пара кавычек;

использовано неправомочное имя переменной;

забыты двоеточие или точка с запятой, или запятая, отделяющая переменные или текст;

забыт номер строки, или в номере строки присутствуют буквы, или номер строки больше максимально разрешенного;

двойные кавычки внутри текста;

строка больше 255 символов;

пропущено ключевое слово;

определен лишний символ, особенно лишняя запятая или двоеточие.

Группа 2. Выполнение оператора READ, когда нет данных в операторе DATA.

Группа 3. Ошибки цикла:

отсутствие оператора NEXT или FOR; WHILE или WEND;

забыто ключевое слово FOR;

вложены два цикла, использующие одинаковую переменную в обоих циклах;

имя переменной в цикле и имя счетчика цикла совпадают;

циклы некорректно вложены один в другой.

Группа 4. Несуществующий номер строки, к которому обращаются операторы GOTO, THEN и т. п.

Группа 5. Значение параметра или индекса вне диапазона:

задание неправильных значений в графических операторах (например, в Бейсик-ПК8010 и Бейсик-ПК8020);

переполнение или деление на нуль.

Группа 6. Ошибки при работе с подпрограммами:

отсутствие RETURN при наличии GOSUB;

отсутствие GOSUB при наличии RETURN;

отсутствие RESUME при наличии ON ERROR GOTO;

отсутствие ON ERROR GOTO при наличии RESUME.

Группа 7. Строковые ошибки:

задание строкового значения вместо числового или наоборот;

превышено количество свободной памяти для строк;

попытка создать строку длиннее 255 символов;

строковое выражение слишком длинное или сложное.

Группа 8. Ошибки при работе с массивами:

неправильное задание индекса массива;

определение размерности одного или того же массива дважды.

Группа 9. Ошибки при работе с файлами:

обращение к файлу, которого нет на диске;

файл с заданным именем уже существует или открыт;

ошибка в спецификации файла;

несоответствие режима файла выполняемым действиям;

создание нового файла при полном каталоге диска;

попытка записи на диск, когда места на диске нет;

переполнение буфера;

ошибочный номер файла или записи;

ошибка определения конца файла.

Группа 10. Ошибки устройств:

ошибка в операции ввода-вывода устройства;

нет информации из устройства ввода-вывода по истечении определенного времени;

печатающее устройство отключено;

ошибка аппаратуры, обнаруживаемая адаптером интерфейса;

отсутствие устройства или заблокированное устройство;

открыта дверца дисковода или диска нет в дисководе;

плохой диск;

попытка записи на защищенный диск.

Группа 11. Ошибки коммуникации:

попытка установления связи при полном входном буфере.

Группа 12. Прочие ошибки:

программа слишком велика или слишком сложна;

ввод в прямом режиме оператора, который может быть использован только в программном режиме;

попытка продолжить программу, которая остановлена из-за ошибки, не существует или модифицирована во время прерывания выполнения;

функция пользователя вызывается перед ее определением оператором DEF FN;

выражение содержит оператор, такой как * или OR, без операнда, следующего за ним.

Все эти ошибки прерывают выполнение программы, вызывают сообщение об ошибке с указанием строки, где эта ошибка встретилась.

Типичные логические (несообщаемые) ошибки:

1. Переинициализация переменной – особенно при использовании циклов.

```
10 FOR N=1 TO 3
20 READ A
30 PRINT A
40 RESTORE
50 NEXT N
60 DATA 1,2,3
```

2. Реверсия условий, т. е. использование знака "=" вместо знака "<>" или ">" и т. п.

3. Включение "равно" в условие "меньше или равно", когда должно быть "меньше" и т.п.

4. Перепутаны подобные имена переменных, особенно переменная A, строка A$ и массив A(X).

5. Забыт порядок программного выполнения – слева направо, но умножение и деление всегда имеют приоритет перед сложением и вычитанием, а функции (INT, RND, ABS и т. п.) перед другими функциями.

6. Вычисления некорректны в цикле, так как FOR I=0 TO 7 выполняет цикл 8 раз, а не 7.

7. Использование одной переменной в нескольких различных местах.

Подробное описание ошибок, приводящих к прерыванию выполнения программы, дано далее.

Сообщения об ошибках XYBASIC

Когда обнаруживается ошибка в программе, интерпретатор выводит на экран сообщение о характере и месте появления ошибки (табл. 5.4).

Например:

```
10 LET 3 = L
```

Интерпретатор выдает сообщение

```
SN ERROR: 10 LET
                3 = L
OK
```

SN – код, отражающий появление синтаксической ошибки

BF	Bad File number	Неправильный номер файла
BS	Bad Subscript	Неправильный индекс
BY	BYte	Байт
CN	CoNtinue	Продолжение
CS	CheckSum	Контрольная сумма
DD	Double Defined	Дважды определенный
DF	Disk Full	Диск полный
EN	ENable	Блокирование
EX	EXeption	Исключение
FC	Function Call	Функциональный вызов
FI	File Input	Ввод файла
FM	File Mode	Режим файла
FN	File Not found	Файл не найден
FO	File Open	Файл открыт
FR	FOR:	Оператор FOR
ID	Illegal Direct	Неправомочная команда
II	Illegal Indirect	Неправомочна вне прямого режима
LS	Long String	Длинная строка
MC	Machine Call	Машинный вызов
NF	NEXT without FOR	NEXT без FOR
OD	Out of Data	Вне данных
OM	Out of Memory	Вне памяти
ON	ON	Оператор ON

OP	OPen	Оператор OPEN
OS	Out of String space	Вне строкового пространства
OV	OVerflow	Переполнение
RG	RETURN without GOSUB	RETURN без GOSUB
RO	ROmsg feature	Особенности ROM
SN	SyNtax	Синтаксис
ST	STring	Строка
TM	Type Mismatch	Несоответствие типов
UF	Unimplemented Feature	Нереализованные возможности
US	Undefined Statement	Неопределенный оператор

Сообщения об ошибках Бейсик-Спектрум+2

Сообщения появляются внизу экрана при останове выполнения программы. Сообщение имеет код, короткое сообщение, объясняющее что произошло, номер строки и номер оператора внутри строки, где произошел останов. Подсказка Ok имеет код 0.

1	NEXT without FOR	NEXT без FOR
2	Variable not found	Переменная не найдена
3	Subscript wrong	Неверный индекс
4	Out of memory	Вне памяти
5	Out of screen	Вне экрана
6	Number too big	Слишком большое число
7	RETURN without GOSUB	RETURN без GOSUB
8	End of file	Конец файла
9	STOP statement	Оператор STOP
A	Invalid argument	Неправильный аргумент
B	Integer out of range	Аргумент вне диапазона
C	Nonsense in BASIC	"Ерунда" в Бейсике
D	BREAK-CONT repeats	Продолжение BREAK-CONT
E	Out of DATA	Вне данных
F	Invalid file name	Неверное имя файла
G	No room for line	Нет места для строк
H	STOP in INPUT	Стоп при вводе
I	FOR without NEXT	FOR без NEXT
J	Invalid I/O device	Устройство ввода-вывода указано неверно
K	Invalid colour	Неверный цвет
L	BREAK into programm	Перерыв в программе

M	RAMTOP no good	Верхняя граница памяти испорчена
O	Invalid stream	Неверный ввод
P	FN without DEF	FN без DEF
Q	Parameter error	Ошибка параметра
R	Tape loading error	Ошибка загрузки ленты
a	MERGE error	Ошибка слияния
b	Wrong file type	Неправильный тип файла
c	CODE error	Ошибка в коде
d	Too many brackets	Слишком много скобок
e	File already exists	Файл уже существует
f	Invalid name	Неверное имя
h	File does not exist	Файл не существует
i	Invalid device	Неправильное устройство
j	Invalid baud rate	Неверная скорость передачи
k	Invalid note name	Неверное имя ноты
l	Number too big	Слишком большое число (PLAY)
m	Note out of range	Нота вне диапазона
n	Out of range	Вне диапазона
o	Too many tied notes	Слишком много связанных нот

Сообщения об ошибках Бейсик-АГАТ

NEXT без FOR	Переопределенный массив
Синтаксическая ошибка	Деление на нуль
RETURN без GOSUB	Ты не в программе
Мало данных	Не тот тип
Не тот тип величины	Очень длинная строка
Переполнение	Очень сложно
Мало памяти	Не могу продолжить
Плохой оператор	Функция не определена
Плохой индекс	Ошибка

Сообщения об ошибках Бейсик-TRS-80

1	NF	NEXT without FOR	NEXT без FOR
2	SN	Syntax error	Синтаксическая ошибка
3	RG	RETURN without GOSUB	RETURN без GOSUB
4	OD	Out of data	Вне данных
5	FC	Illegal function call	Неправильный функциональный вызов
6	OV	Overflow	Переполнение
7	OM	Out of memory	Вне памяти
8	UL	Undefined line	Неопределенная строка
9	BS	Subscript out of range	Индекс вне диапазона
10	DD	Redimensioned array	Переопределение массива
11	/0	Division by zero	Деление на нуль
12	ID	Illegal direct	Неправомочная команда
13	TM	Type mismatch	Несоответствие типа
14	OS	Out of string space	Вне строкового пространства
15	LS	String too long	Слишком длинная строка
16	ST	String fomula too complex	Слишком сложная формула строки
17	CN	Can't continue	Нельзя продолжить
18	NR	No RESUME	Нет RESUME
19	RW	RESUME without error	RESUME без ошибки
20	UE	Unprintable error	Непечатаемая ошибка
21	MO	Missing operand	Отсутствующий операнд
22	FD	Bad file data	Плохие данные в файле
23	L3	Disk BASIC only	Только дисковый Бейсик

Сообщения об ошибках MBASIC

1	NEXT without FOR	NEXT без FOR
2	Syntax error	Синтаксическая ошибка
3	RETURN without GOSUB	RETURN без GOSUB
4	Out of DATA	Вне данных

5	Illegal function call	Неправомочный функциональный вызов
6	Overflow	Переполнение
7	Out of memory	Вне памяти
8	Undefined line	Неопределенный номер строки
9	Subscript out of range	Индекс вне диапазона
10	Redimensioned array	Переопределение массива
11	Division by zero	Деление на нуль
12	Illegal direct	Неправомочная команда
13	Type mismatch	Несоответствие типа
14	Out of string space	Вне строкового пространства
15	String too long	Слишком длинная строка
16	String formula too complex	Слишком сложная формула строки
17	Can't continue	Нельзя продолжить
18	Undefined user function	Неопределенная функция пользователя
19	No RESUME	Нет оператора RESUME
20	RESUME without error	RESUME без ошибки
21	Unprintable error	Непечатаемая ошибка
22	Missing operand	Отсутствующий операнд
23	Line buffer overflow	Буфер строки полный
24-25	Unprintable error	Непечатаемая ошибка
26	FOR without NEXT	FOR без NEXT
27-28	Unprintable error	Непечатаемая ошибка
29	WHILE without WEND	WHILE без WEND
30	WEND without WHILE	WEND без WHILE
31-49	Unprintable error	Непечатаемая ошибка
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	Bad file mode	Неправильный режим файла
55	File already open	Файл уже открыт
56	Unprintable error	Непечатаемая ошибка
57	Disk I/O error	Ошибка при вводе-выводе на диск
58	File already exists	Файл уже существует
59-60	Unprintable error	Непечатаемая ошибка
61	Disk full	Диск полный
62	Input past end	Ввод после конца файла
63	Bad record number	Неправильный номер записи
64	Bad file name	Неправильное имя файла
65	Unprintable error	Непечатаемая ошибка
66	Direct statement in file	Прямой оператор в файле
67	Too many files	Слишком много файлов
68 255	Unprintable error	Непечатаемая ошибка

Коды 31–49 и 68–255 можно использовать для организации сообщений об ошибках, не вошедших в стандартный перечень (о моделируемых ошибках)

Сообщения об ошибках Бейсик-Корвет

Таблица 5.4

Код	Сообщение об ошибке	Бейсик-ПК8010	Бейсик-ПК8020
1	NEXT БЕЗ FOR	+	+
2	ОШИБКА СИНТАКСИСА	+	+
3	RETURN БЕЗ GOSUB	+	+
4	ВНЕ DATA	+	+
5	НЕВЕРЕН ВЫЗОВ ФУНКЦИИ	+	+
6	ПЕРЕПОЛНЕНИЕ	+	+
7	НЕТ ПАМЯТИ	+	+
8	НЕОПРЕДЕЛЕННЫЙ НОМЕР СТРОКИ	+	+
9	ИНДЕКС ВНЕ ДИАПАЗОНА	+	+
10	ПЕРЕОПРЕДЕЛЕНИЕ МАССИВА	+	+
11	ДЕЛЕНИЕ НА 0	+	+
12	НЕВЕРНАЯ КОМАНДА	+	+
13	НЕВЕРНЫЙ ТИП	+	+
14	НЕТ ПАМЯТИ ДЛЯ СТРОК	+	+
15	ДЛИННАЯ СТРОКА	+	+
16	СЛОЖНАЯ СТРОКА	+	+
17	ОШИБКА CONT	+	+
18	НЕТ ОПРЕДЕЛЕНИЯ FN	+	+
19	НЕТ RESUME	+	+
20	RESUME БЕЗ ERROR	+	+
21	НЕОПРЕДЕЛЕННАЯ ОШИБКА	+	+
22	НЕТ ОПЕРАНДА	+	+
23	БУФЕР СТРОКИ ПОЛНЫЙ	+	+
24	ОШИБКА ЧТЕНИЯ	+	+
25-50	НЕОПРЕДЕЛЕННАЯ ОШИБКА	+	
51	ВНУТРЕННЯЯ ОШИБКА	+	
52	НЕВЕРЕН НОМЕР ФАЙЛА	+	
53	ФАЙЛ НЕ НАЙДЕН	+	
54	НЕВЕРЕН РЕЖИМ ФАЙЛА	+	
55	ФАЙЛ УЖЕ ОТКРЫТ	+	
56	НЕОПРЕДЕЛЕННАЯ ОШИБКА	+	
57	ОШИБКА ВВОДА-ВЫВОДА С ДИСКА	+	
58	ФАЙЛ УЖЕ СУЩЕСТВУЕТ	+	
59-60	НЕОПРЕДЕЛЕННАЯ ОШИБКА	+	
61	ДИСК ПОЛНЫЙ	+	
62	ВВОД ПОСЛЕ КОНЦА ФАЙЛА	+	
63	НЕОПРЕДЕЛЕННАЯ ОШИБКА	+	
64	НЕВЕРНО ИМЯ ФАЙЛА	+	
65	НЕОПРЕДЕЛЕННАЯ ОШИБКА	+	
66	ПРЯМОЙ ОПЕРАТОР В ФАЙЛЕ	+	
67	МНОГО ФАЙЛОВ	+	

Коды 25–50, 68–255 можно использовать для сообщений о моделируемых ошибках.

1	NEXT without FOR	NEXT без FOR
2	Syntax error	Синтаксическая ошибка
3	RETURN without GOSUB	RETURN без GOSUB
4	Out of DATA	Вне данных
5	Illegal function call	Неправомочный функциональный вызов
6	Overflow	Переполнение
7	Out of memory	Вне памяти
8	Undefined line number	Неопределенный номер строки
9	Subscript out of range	Индекс вне диапазона
10	Redimensioned array	Переопределение массива
11	Division by zero	Деление на нуль
12	Illegal direct	Неправомочная директива
13	Type mismatch	Несоответствие типа
14	Out of string space	Вне строкового пространства
15	String too long	Слишком длинная строка
16	String formula too complex	Слишком сложная формула строки
17	Can't continue	Нельзя продолжить
18	Undefined user function	Неопределенная функция пользователя
19	Devise I/O error	Ошибка устройства ввода-вывода
20	Verify error	Ошибка контроля
21	No RESUME	Нет оператора RESUME
22	RESUME without error	RESUME без ошибки
23	Unprintable error	Непечатаемая ошибка
24	Missing operand	Отсутствующий операнд
25	Line bufer overflow	Переполнение буфера строки
26-49	Unprintable error	Непечатаемая ошибка
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	Прямой оператор в файле
58	Sequential I/O only	Только последовательный ввод-вывод
59	File not OPEN	Файл не открыт
60	Bad FAT	Неверная информация в FAT (таблице распределения файлов)
61	Bad file mode	Неправильный режим файла}
62	Bad drive name	Неправильное имя диска
63	Bad sector number	Неправильный номер сектора
64	File still open	Файл еще не открыт
65	File already exists	Файл уже существует
66	Disk full	Диск полный

67	Too many files	Слишком много файлов
68	Disk write protected	Защита записи диска
69	Disk I/O error	Ошибка при вводе-выводе на диск
70	Disk offline	Дисковод выключен
71	Rename across disks	Переименование одного диска на другой
72-255	Unprintable error	Непечатаемая ошибка

Коды 26—49, 72—255 можно использовать для сообщений о моделируемых ошибках.

Сообщения об ошибках BASICA (EC1840)

1	NEXT without FOR	NEXT без FOR
2	Syntax error	Синтаксическая ошибка
3	RETURN without GOSUB	RETURN без GOSUB
4	Out of data	Вне данных
5	Illegal function call	Неправомочный функциональный вызов
6	Overflow	Переполнение
7	Out of memory	Вне памяти
8	Undefined line number	Неопределенный номер строки
9	Subscript out of range	Индекс вне диапазона
10	Duplicate definition	Двойное определение
11	Division by zero	Деление на нуль
12	Illegal direct	Неправомочная команда
13	Type mismatch	Несоответствие типа
14	Out of string space	Вне строкового пространства
15	String too long	Слишком длинная строка
16	String formula too complex	Слишком сложная формула строки
17	Can't continue	Нельзя продолжить
18	Undefined user function	Неопределенная функция пользователя
19	No RESUME	Нет оператора RESUME
20	RESUME without error	RESUME без ошибки
21	Unprintable error	Непечатаемая ошибка
22	Missing operand	Отсутствующий операнд
23	Line buffer overflow	Переполнение буфера строки
24	Device Timeout	Тайм-аут устройства
25	Device Fault	Ошибка устройства
26	FOR without NEXT	FOR без NEXT
27	Out of paper	Нет бумаги
28	Undefined error	Неопределенная ошибка
29	WHILE without WEND	WHILE без WEND
30	WEND without WHILE	WEND без WHILE
31-49	Undifined error	Неопределенная ошибка
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	Bad file mode	Неправильный режим файла
55	File already open	Файл уже открыт
56	Undefined error	Неопределенная ошибка
57	Device I/O error	Ошибка при вводе-выводе на диск
58	File already exists	Файл уже существует
59-60	Undefined error	Неопределенная ошибка
61	Disk full	Диск полный
62	Input past end	Ввод после конца файла
63	Bad record number	Неправильный номер записи
64	Bad file name	Неправильное имя файла
65	Undefined error	Неопределенная ошибка
66	Direct statement in file	Прямой оператор в файле
67	Too many files	Слишком много файлов
68	Device unavailable	Устройство недоступно
69	Communication buffer overflow	Переполнение буфера коммуникации
70	Disk write protect	Защита записи диска
71	Disk not Ready	Диск не готов
72	Disk Media Error	Ошибка середины диска
73	Advanced Feature	Расширенные возможности
74-255	Undefined error	Неопределенная ошибка

Коды 31 – 49, 74 – 255 можно использовать для сообщений о моделируемых ошибках.

5.4. Моделирование ошибочных ситуаций

Все перечисленные выше ошибки, имеющие код, можно "отлавливать" в процессе работы программы и обрабатывать так, как нужно программисту. Обычно ошибки приводят к прерыванию работы программы, но если встречается смоделированная ошибка (ошибка, не указанная в стандартном перечне данной версии), то оператор ON ERROR... GOTO (в АГАТе – ONERR GOTO) передает управление оператору, номер строки которого задан в ON ERROR... GOTO. Этот оператор запрещает вывод на экран сообщения, отличного от заданного пользователем, и не передает управление интерпретатору.

Например:

```
10 ON ERROR GOTO 100 : REM "Уход по ошибке на
   строку 100"
20 INPUT X
30 PRINT SQR(X)
40 GOTO 20
100 PRINT "Аргумент не может быть отрицательным"
110 PRINT "Введите новое значение"
120 RESUME NEXT
RUN
?3
 1.732015
?-3
Аргумент не может быть отрицательным
Введите новое значение
?
```

Другой пример демонстрирует использование псевдопеременных ERL и ERR при моделировании ошибки, которые определяют номер строки, в которой произошла ошибка, и код ошибки соответственно.

```
10 ON ERROR GOTO 100
20 INPUT "Введите число";A
30 PRINT SQR(A)
40 GOTO 20
100 IF ERR=5 AND ERL=30 THEN PRINT"Введите
    положительное число";
110 RESUME 20
RUN
Введите число?4
 2
Введите число?-4
Введите положительное число
Введите число?
```

Оператор RESUME обеспечивает возобновление выполнения работы программы после обнаружения и обработки ошибки.

В XYBASIC ловушка ошибок включается и выключается с помощью команд TRUP и UNTRAP.

Для локализации логической ошибки рекомендуется вставлять в программу операторы STOP и PRINT, которые останавливают выполнение программы в нужном месте и печатают значения переменных, по которым можно проследить правильность выполнения программы. Небольшие куски программы можно отлаживать в командном режиме.

Например:

```
FOR I=1 TO 100 : PRINT I,EXP(I) : NEXT
```

Для отладки одной из ветвей программы можно использовать команду RUN с номером строки, с которого нужно начать выполнение. Для этой же цели служит оператор GOTO с номером строки.

5.5. Описание инструкций

Таблица 5.5

Инструкция	Номер версии									
	1	2	3	4	5	6	7	8	9	10
AUTO		+			+	+	+	+	+	+
CONT	+	+		+	+	+	+	+	+	+
CONTINUE			+							
DEL				+						
DELETE	+	+			+	+	+	+	+	+
EDIT		+			+	+	+		+	+
ERL	+				+	+	+	+	+	+
ERR	+				+	+	+	+	+	+
ERROR	+				+	+	+	+	+	+
LIST	+	+	+	+	+	+	+	+	+	+

Окончание табл. 5.5

Инструкция	Номер версии									
	1	2	3	4	5	6	7	8	9	10
LLIST			+		+	+		+	+	+
NEW	+	+	+	+	+	+	+	+	+	+
NOTRACE				+						
ONERR				+						
ON ERROR	+				+	+	+	+	+	+
RESUME	+			+	+	+	+	+	+	+
RUN	+	+	+	+	+	+	+	+	+	+
STOP	+	+	+	+	+	+	+	+	+	+
TRACE		+		+						
TRAP		+								
TROFF	+				+	+	+	+	+	+
TRON	+				+	+	+	+	+	+
UNTRACE		+								
UNTRAP		+								

AUTO – команда, автоматически генерирующая номер строки. Обычно используется для ввода новых программ.

```
AUTO [номер][,инкремент]
```

номер – номер, который будет использован в качестве начального при нумерации строк;

инкремент – значение, которое будет прибавляться к каждому номеру строки для получения следующего номера.

Нумерация начинается с номера, для получения следующего номера строки к предыдущему прибавляется инкремент. Если оба этих параметра не указаны, то по умолчанию принимается AUTO 10,10. Если за номером следует запятая, а инкремент не определен, то принимается инкремент, определенный в предыдущей команде AUTO. Если номер не указан, а инкремент определен, то нумерация начинается с 0. На месте номера строки, чтобы указать текущую строку, может быть использована точка (.).

Если AUTO генерирует номер строки, уже существующий в программе, то после номера строки на экране дисплея появится символ "*", чтобы предупредить о том, что любая введенная строка будет замещать уже существующую строку. Однако, если сразу после "*" нажать клавишу возврата каретки, то существующая строка не будет замещена, и AUTO будет генерировать следующий номер строки.

Команда AUTO останавливается нажатием клавиш CTRL–C и возврата каретки. Строка, в которой нажимается CTRL–C, не сохраняется. Управление передается на уровень команд.

Например:

```
AUTO
```

Эта команда генерирует номера строк: 10, 20, 30...

```
AUTO 100,50
```

Эта команда генерирует номера строк: 100, 150, 200...

```
AUTO 500
```

Эта команда генерирует номера строк: 500, 550, 600...

```
AUTO ,20
```

Эта команда генерирует номера строк: 0, 20, 40...

CONT – команда, продолжающая выполнение программы после прерывания

CONT

Команда CONT может быть использована для возобновления выполнения программы после нажатия клавиши останова или после выполнения операторов STOP и END. Выполнение продолжается с той точки, где осуществилось прерывание. Если прерывание встречается после напоминания в операторе INPUT, то выполнение продолжается после повторения напоминания.

Команда CONT обычно используется вместе с оператором STOP для устранения ошибок. Когда выполнение останавливается, можно проверить и изменить значения переменных, используя операторы в прямом режиме.

Для продолжения выполнения программы можно использовать команду CONT или оператор GOTO в прямом режиме, который восстанавливает выполнение с определенного номера строки. Если программа редактируется во время прерывания, то команда CONT не выполняется и выдается сообщение об ошибке.

Например:

```
10 FOR A=1 TO 50
20 PRINT A;
30 NEXT A
RUN
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
 20 21 22 23 24 25 26 27 28 29 <нажатие клавиши
останова>
^C
ВЫХОД В 20
Ok
CONT
 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
 46 47 48 49 50
Ok
```

CONTINUE – команда продолжения выполнения программы после приостанова

CONTINUE

См. команду CONT.

DEL – команда, предназначенная для удаления программных строк в памяти

DEL номер строки

См. команду DELETE.

DELETE – команда, предназначенная для удаления программных строк в памяти

DELETE [номер строки[-[номер строки]]]

Команда DELETE удаляет из памяти определенную группу строк. После выполнения команды управление передается на уровень команд. Точка (.) может быть использована на месте номера строки для определения текущей строки. Если один из двух указанных номеров строк отсутствует в программе, то выдается сообщение об ошибке Undefined line number (Неопределенный номер строки).

Например:

DELETE 40	стереть строку 40.
DELETE 40-100	стереть все строки с 40 по 100 включительно.
DELETE -40	стереть все строки с начала программы до 40 включительно.

EDIT – команда, с помощью которой выводится на экран для редактирования строка с указанным номером

EDIT номер строки

номер строки – номер строки для редактирования или (·)

Курсор устанавливается в позицию первого символа после номера строки. Затем строка может редактироваться. Если строки с таким номером в программе нет, то выдается сообщение Undefined line number (Неопределенный номер строки). Для редактирования только что введенной строки можно использовать точку (·).

Команды редактирования были рассмотрены выше.

Например:

EDIT 40

EDIT

ERR и **ERL** – переменные, которые выводят на экран код ошибки и номер строки, в которой произошла ошибка

X=ERR

Y=ERL

Переменная ERR содержит код последней ошибки, а переменная ERL содержит номер строки, где эта ошибка обнаружилась. Переменные ERR и ERL обычно используются в операторах IF ... THEN, чтобы организовать переход программы на подпрограмму, обрабатывающую ошибку. При использовании переменной ERL в операторе IF ... THEN надо убедиться, что номер строки находится справа от ERL.

Например:

IF ERL=<номер строки> THEN PRINT "ОШИБКА"

Номер строки должен находиться справа от переменной ERL, чтобы командой RENUM он мог быть перенумерован. Если оператор, который вызывает ошибку, был задан в прямом режиме, то ERL сообщает номер 65535. Так как нежелательно, чтобы этот номер строки изменился во время выполнения команды RENUM, то при проверке ошибки в прямом режиме следует использовать форму:

IF 65535=ERL THEN...

Переменные ERR и ERL могут быть заданы с помощью оператора ERROR. Например:

```
10 ON ERROR GOTO 100
20 LPRINT "This goes to the printer"
30 END
:
100 IF ERR=27 THEN PRINT "Turn printer on";:RESUME
```

ERROR — оператор, используемый для распознавания кода ошибки, определенного пользователем

ERROR n

n — целочисленное выражение от 0 до 255.

Если указанное n соответствует коду ошибки, то программа будет выполняться так, как будто встретилась ошибка, указанная этим кодом (моделируется ошибка).

Если оператором ON ERROR определена подпрограмма обработки ошибки, то будет осуществляться переход на эту подпрограмму, в противном случае будет печататься сообщение об ошибке, и выполнение программы будет остановлено. Для определения кода моделируемой ошибки используется значение, отличное от значений, придаваемых интерпретатором. Рекомендуется использовать коды больше 200. Если ваша собственная моделируемая ошибка определяется таким образом и не обрабатывается в подпрограмме обработки ошибок, то Бейсик-система будет печатать сообщение Undefined error (Неопределенная ошибка), и выполнение программы будет приостановлено.

Например:

```
10 T=15
20 ERROR T
RUN
ДЛИННАЯ СТРОКА В 20
Ok
110 ON ERROR GOTO 400
120 INPUT "WHAT IS YOUR BET";B
130 IF B>5000 THEN ERROR 210
:
400 IF ERR=210 THEN PRINT "HOUSE LIMIT IS 5000"
410 IF ERL=130 THEN RESUME 120
```

LIST — команда вывода на экран дисплея текста программы

LIST [номер строки[-[номер строки]]]

Вывод программы на экран может быть прерван нажатием клавиш CTRL—C. После этого управление передается на уровень команд. Нажатие клавиш CTRL—S задерживает вывод текста на экран, а нажатие CTRL—Q продолжает вывод.

Если номера строк не указаны, то выводится вся программа, находящаяся в памяти. Если задан только первый номер строки, то выводится на экран только строка с этим номером. Если заданы первый номер и тире, то выводятся программные строки, начиная с заданного номера и до конца программы. Если заданы тире и второй номер строки, то выводятся строки с начала программы до заданного номера включительно. Если заданы оба номера строк, то выводятся все строки, номера которых лежат в заданном диапазоне, включая строки с заданными номерами. Для указания текущей строки можно использовать точку (.).

Например:

LIST	выводится вся программа;
LIST 35	выводится только строка 35;
LIST 100-200	выводятся строки с 100 по 200 включительно;
LIST .	выводится текущая строка.

LLIST – команда, выводящая на печатающее устройство программные строки

```
LLIST [номер строки[-[номер строки]]]
```

На печатающее устройство выводится вся программа или ее часть, находящаяся в текущий момент в памяти. Номера строк для команды LLIST указываются так же, как и для команды LIST. Работа по команде LLIST не может быть остановлена нажатием клавиш. Для останова печати следует выключить печатающее устройство на 10 с. После команды LLIST интерпретатор всегда возвращается на уровень команд.

Например:

`LLIST`	распечатывает на печатающем устройстве всю введенную программу;
`LLIST 30-100`	распечатывает строки программы с 30 по 100 включительно.

NEW – команда, которая удаляет программу, находящуюся в текущий момент в памяти и очищает все переменные

```
NEW
```

Команда NEW обычно используется для освобождения памяти перед введением новой программы. Интерпретатор всегда возвращает управление на уровень команд после выполнения команды NEW. Команда NEW закрывает все файлы.

NOTRACE – команда отмены режима трассировки

```
NOTRACE
```

См. команды UNTRACE или TROFF.

ONERR – оператор организации ловушки ошибок

```
ONERR GOTO номер строки
```

См. оператор ON ERROR.

ON ERROR – оператор организации ловушки ошибок

```
ON ERROR [GOTO/GOSUB] номер строки
```

Если ловушка ошибок блокирует любые ошибки, включая синтаксические, то это будет вызывать переход на подпрограмму обработки определенной ошибки. Если номер строки, указанной в операторе ON ERROR GOTO, не существует в программе, то появляется сообщение об ошибке Undefined line number (Неопределенный номер строки). Чтобы заблокировать ловушку ошибок, необходимо выполнить оператор ON ERROR GOTO 0. Последовательные ошибки будут выдавать сообщения об ошибках и останавливать выполнение программы.

Оператор ON ERROR GOTO 0, который появляется в подпрограмме ловушки ошибок, останавливает выполнение программы и печатает сообщение об ошибке, которая вызвала прерывание. Оператор RESUME используется для выхода из подпрограммы обработки ошибки.

Например:

```
10 ON ERROR GOTO 100
20 LPRINT "This goes to the printer"
30 END
100 IF ERR=27 THEN PRINT "Turn printer on":RESUME
```

RESUME – оператор, продолжающий работу программы после выполнения подпрограммы обработки ошибок

```
RESUME [0]

RESUME NEXT

RESUME номер строки
```

В зависимости от того, где выполнение должно быть восстановлено, используется один из вышеприведенных форматов.

При первом формате выполнение восстанавливается с оператора, который вызвал ошибку. При втором формате восстанавливается выполнение с оператора, непосредственно следующего за оператором, который вызвал ошибку. При третьем формате восстанавливается выполнение с заданного номера строки. Если встретился оператор RESUME, для которого не была определена подпрограмма обработки ошибки, то выдается сообщение RESUME without error (RESUME без ошибки).

Например:

```
10 ON ERROR GOTO 900
:
900 IF (ERR=230) AND (ERL=90) THEN PRINT "TRY
AGAIN": RESUME 80
```

RUN – команда, которая начинает выполнение программы

```
RUN [номер строки]

RUN "спецификация файла"[,R]
```

При первом формате команда начинает выполнение программы, находящейся в данный момент в памяти. Если задан номер строки, то выполнение начинается со строки с этим номером. Во втором формате команда загружает программу с внешнего устройства (диска или кассеты) и выполняет ее. При этом удаляется предыдущая программа в памяти, закрываются все открытые файлы и очищаются все переменные. Но если указан параметр **R**, то все открытые файлы остаются открытыми. Например:

```
RUN 100

RUN "PRIMER.BAS"

RUN "ZADACHA",R
```

STOP – оператор, прерывающий выполнение программы и возвращающий управление на уровень команд

```
STOP
```

Для прерывания выполнения программы оператор может быть расположен в любом месте программы. Если при выполнении программы встретился оператор STOP, то на экран выдается сообщение STOP in nnnn (выход в nnnn), где nnnn – номер строки, в которой встретился STOP. В отличие от END, оператор STOP не закрывает файлы. Выполнение программы может быть продолжено командой CONT. Например:

```
10 INPUT A,B,C
20 K=A^2*5.3:L=B^3/.26
30 STOP
40 M=C*K100:PRINT M
RUN
?1,2,3
ВЫХОД В 30
PRINT L
30.76923
Ok
CONT
115.9
Ok
```

TRACE и UNTRACE – команды, включающие и выключающие трассировку программы

TRACE

UNTRACE

В режиме трассировки печатаются пронумерованные строки, заключенные в квадратные скобки, и результат выполнения каждого оператора (имя и значение переменной). Например:

```
10 TRACE
20 PRINT 2; "IS PRIM"
30 N=1
40 N=N2
50 FOR I=3 TO N/2 STEP 2
60 IF N MOD I = 0 THEN 40
70 NEXT I
80 PRINT N;"IS PRIM"
90 GOTO 40
RUN
[20 PRINT 2;"IS PRIM"] 2 IS PRIM
[30 N=1] N=1
[40 N=N2] N=3
[50 FOR I=3 TO N/2 STEP 2] I=3
[80 PRINT N; "IS PRIM"] 3 IS PRIM
[90 GOTO 40]
```

и т. д.

TRAP и UNTRAP – команды включения и выключения режима фиксации ошибок

TRAP

UNTRAP

Иногда необходимо продолжить выполнение программы после того, как XYBASIC обнаружит ошибку, например для того, чтобы программа работала без останова. Это достигается использованием команды UNTRAP, при выполнении которой режим UNTRAP XYBASIC пытается продолжить работу программы после вывода на экран сообщения об ошибке.

Некоторые ошибки всегда приводят к прерыванию выполнения программы без возможности продолжения работы. Но для других ошибок XYBASIC применяет особую восстановительную процедуру и продолжает выполнение программы.

Для задания режима UNTRAP необходимо лишь набрать команду UNTRAP, а для выхода – TRAP. В самом начале работы XYBASIC находится в режиме TRAP, а также возвращается в режим TRAP после выполения команды NEW. Например:

```
NEW
20 FOR I=1 TO 3
30 PRINNT I
40 NEXT I
RUN
SN ERROR: 30 PRINNT I
OK
```

При обнаружении синтаксической ошибки в 30 строке XYBASIC возвращается в командный режим.

Другой пример демонстрирует работу режима UNTRAP:

```
10 UNTRAP
RUN
SN ERROR: 30 PRINNT I
SN ERROR: 30 PRINNT I
SN ERROR: 30 PRINNT I
OK
```

Ошибочная строка игнорируется и выполнение программы продолжается.

TRON и TROFF – включение и выключение трассировки

TRON

TROFF

По команде TRON номера строк по мере выполнения программы печатаются на экране в квадратных скобках; команда TROFF выключает трассировку. Например:

```
10 K=10
20 FOR I=1 TO 2
30 L=K10
40 PRINT I;K;L
50 K=K10
60 NEXT
70 END
TRON
RUN
[10][20][30][40]1 10 11[50] и т.д.
```

Глава 6

Работа с графическими операторами и операторами звука

6.1. Графический экран

Графический экран в современных ПЭВМ, как правило, растрового типа. В графических видеоконтрольных устройствах (ГВКУ) такого типа электронный луч под управлением дисплейного процессора вычерчивает регулярный растр горизонтальных строк развертки. Изображение при этом получается изменением интенсивности точек растра. В случае цветного дисплея производится также управление интенсивностями трех лучей – для красного, зеленого и синего цветов (RGB-сигнал).

С экраном связаны используемые в интерпретаторах языка Бейсик следующие понятия: координаты точки, цвет точки, режимы работы экрана (рис. 6.1).

Координаты точки. Точка – минимальный программируемый элемент графического изображения. Точка в этом смысле отличается от понятия собственно точки экрана, "электронной точки", рисуемой электронным лучом.

Поэтому в иностранной литературе, а часто и в отечественной, употребляется слово "пиксел" (от англ. PICture ELEment), точно соответствующий указанному выше понятию графической точки. Графическая точка может состоять из нескольких "электронных точек" и воспринимается глазом как единый элемент изображения в виде точки того или иного цвета.

В графических дисплеях, как правило, применяется от 256 до 2048 строк развертки. Чем больше строк, тем выше разрешающая способность дисплея и качество изображения. В ПЭВМ может применяться в качестве цветного видеоконтрольного устройства (ЦВКУ) бытовой телевизионный приемник (телевизионный стандарт 625 строк).

В профессиональных ПЭВМ применяются специальные ЦВКУ с высокой разрешающей способностью. Обычно в каждой ПЭВМ разрешающая способность ЦВКУ имеет несколько фиксированных значений и может задаваться дисплейным процессором.

В соответствии со стандартом на языке Бейсик вершина координат находится в верхнем левом углу экрана, т. е. выглядит так, как показано на рис. 6.2.

В некоторых ПЭВМ существует иная система координат, так, в ПЭВМ "Спектрум" начало координат находится в нижнем левом углу экрана, так что точка (175,0) – это крайняя нижняя точка в правом углу экрана.

В случае разных систем координат переносимость программ на Бейсике сильно затрудняется. Принятие в стандарте указанной выше системы координат объясняется тем, что нарисованную таким образом картинку можно воспроизводить на экране любого размера (пусть и не целиком). В ряде версий языка существуют операторы, позволяющие работать в системе координат, выходящей за пределы физических координат растра.

В каждой системе координат существуют два основных способа задания координаты точки – абсолютный и относительный. В обоих случаях задаются две координаты X и Y, однозначно определяющие место точки на экране. При задании координат абсолютным способом значения X и Y (или выражений, определяющих эти значения) отсчитываются от начала координат (например, X=100, Y=50 означает, что задаваемая точка удалена от начала координат на 100 графических точек по оси X и на 50 графических точек по оси Y).

При задании координат относительным способом значения X и Y отсчитываются от координаты точки, выведенной на экран последней (или выведенной на экран последней определенным оператором, в Бейсик-АГАТ таким является оператор PLOT). Для указания на

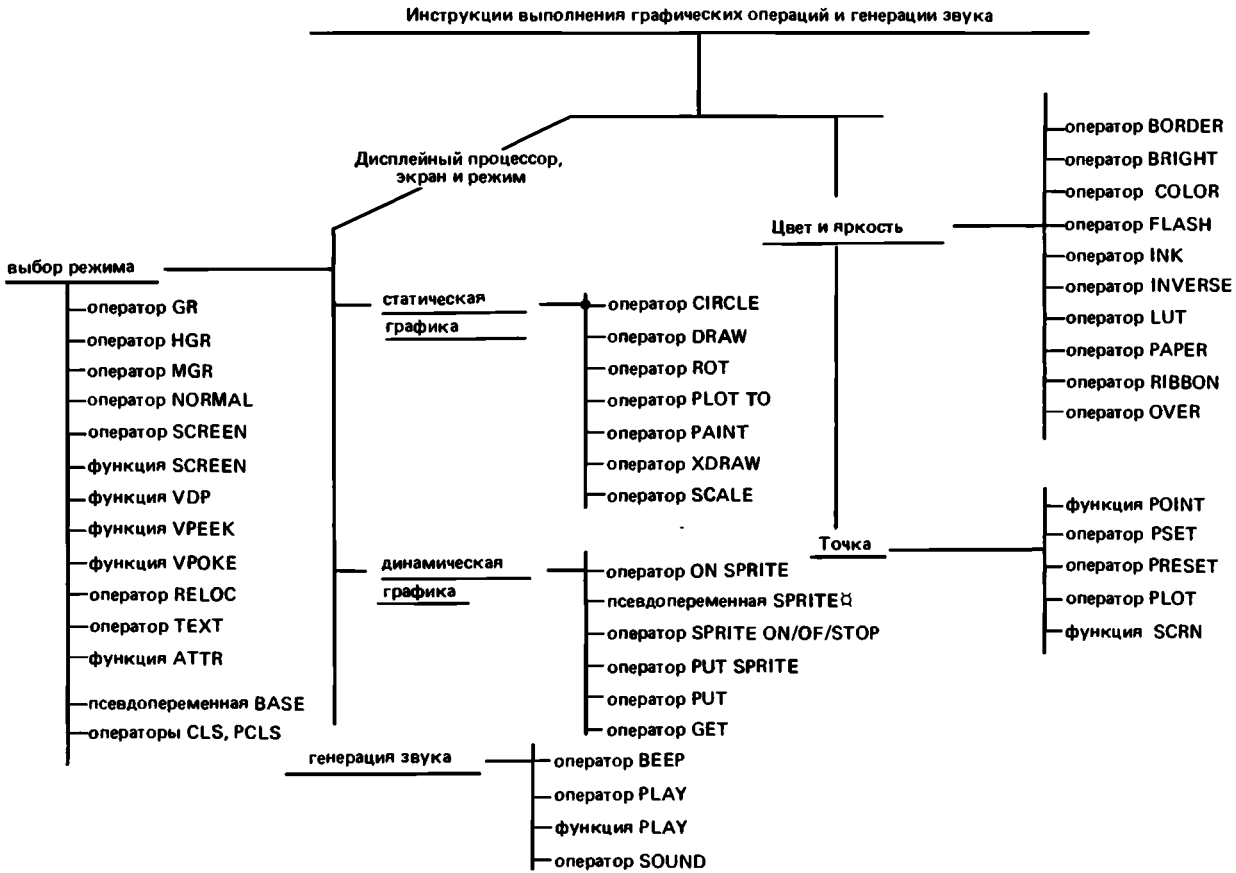

Инструкции выполнения графических операций и генерации звука
Дисплейный процессор, экран и режим
выбор режима
оператор GR
оператор HGR
оператор MGR
оператор NORMAL
оператор SCREEN
функция SCREEN
функция VDP
функция VPEEK
функция VPOKE
оператор RELOC
оператор TEXT
функция ATTR
псевдопеременная BASE
операторы CLS, PCLS
статическая графика
оператор CIRCLE
оператор DRAW
оператор ROT
оператор PLOT TO
оператор PAINT
оператор XDRAW
оператор SCALE
динамическая графика
оператор ON SPRITE
псевдопеременная SPRITE¤
оператор SPRITE ON/OF/STOP
оператор PUT SPRITE
оператор PUT
оператор GET
генерация звука
оператор BEEP
оператор PLAY
функция PLAY
оператор SOUND
Цвет и яркость
оператор BORDER
оператор BRIGHT
оператор COLOR
оператор FLASH
оператор INK
оператор INVERSE
оператор LUT
оператор PAPER
оператор RIBBON
оператор OVER
Точка
функция POINT
оператор PSET
оператор PRESET
оператор PLOT
функция SCRN

Рис. 6.1

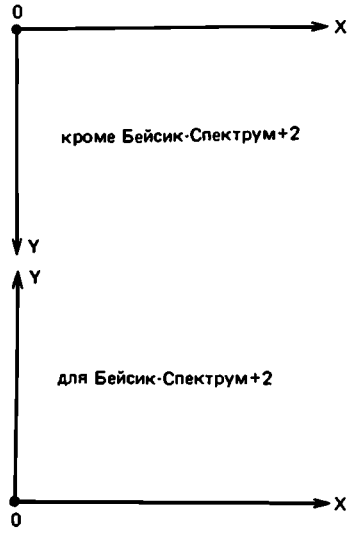


Рис. 6.2

этот способ определения координат используется служебное слово STEP. Так, если последняя выведенная точка находится в точке экрана с координатами 100, 50, то использование STEP (−50, 50) в любом из графических операторов для задания координат означает, что мы хотим неявно указать на точку экрана с координатами X=50, Y=100.

Относительная форма задания координат дает возможность продолжить начатое ранее построение на экране.

Цвет точки. Управление цветом точки производится с помощью дисплейного процессора индивидуально для каждой точки.

Емкость буфера регенерации растрового ЦВКУ соответствует размеру экрана. Каждый элемент буфера содержит значения интенсивности или цвета для соответствующей графической точки на экране. В самом простом случае требуется по одному биту на каждую графическую точку (черно-белое изображение). Если нужно управлять яркостью каждого из лучей, то необходимо увеличивать количество битов (2 бита — 4 значения яркости, 3 бита — 8 значений и т. д).

Если, например, элемент буфера содержит 6 битов и на управление каждым из основных цветов выделяется 2 бита, то максимальное количество цветовых оттенков, которое можно получить на экране, составит 64 (2^6).

При увеличении количества цветов не экономично прямое увеличение памяти буфера регенерации, поэтому часто используются такие средства, как палитра и таблица цветов. Эти средства позволяют управлять заданием цвета, используя содержимое элемента буфера не для управления интенсивностью цвета, а как индекс в палитре или таблице цветов, подлежащий дальнейшей расшифровке дисплейным процессором (см. операторы COLOR, LUT). В Бейсике цвета обычно пронумерованы, причем эти номера раз и навсегда зафиксированы.

В табл. 6.1 приведены номера цветов в основных версиях языка.

Таблица 6.1

Цвет	КУВТ "КОРВЕТ"	MSX BASIC	PC IBM п1	п2	Спектрум+2	АГАТ
0	черный	прозрачный	фон	фон	черный	черный
1	синий	черный	зел	гол	синий	красный
2	зеленый	средне-зел	крас	фиол	красный	зеленый
3	голубой	свет-сер	кор	бел	ярко-крас	желтый
4	красный	тем-гол			зеленый	синий
5	фиолет	свет-гол			циановый	фиолет.
6	желтый	тем-крас			желтый	голубой
7	белый	голубой			белый	белый
8		средне-крас				черный
9		свет-крас				красный
10		тем-желт				зеленый

Окончание табл. 6.1

Цвет	КУВТ "КОРВЕТ"	MSX BASIC	PC n1	IBM n2	Спектрум+2	АГАТ
11		свет-желт				желтый
12		тем-зел				синий
13		васильковый				фиолет.
14		серый				голубой
15		белый				белый

При построении графических изображений важное значение имеют понятия фон (задний фон), передний фон (план) и граница (бордюр).

В некоторых версиях языка (например, Бейсик-Спектрум+2) задний фон именуется "бумага" (paper), а передний фон – "чернила" (ink). Это образно поясняет смысл переднего и заднего фонов. На заднем фоне происходит появление точек другого цвета, которые и составляют изображение, в простейшем случае – это черные точки на белом фоне.

Когда цвет точки переднего фона становится одинаковым с цветом точки заднего фона, она "исчезает". Цвета точек заднего и переднего фонов задаются отдельно. Цвет границы (бордюра) обычно совпадает с цветом фона и служит для заполнения нерабочей области экрана, улучшая картинку.

Чаще всего управление цветом осуществляет оператор COLOR (HCOLOR). Установленные в этом операторе цвета переднего и заднего фонов действуют до следующего аналогичного оператора. Это не исключает того, что каждый графический оператор может "локально" устанавливать цвета переднего и заднего фонов только на время своей работы.

6.2. Дисплейный процессор

Дисплейный процессор служит для управления работой ГВКУ и, в частности, для задания режимов его работы. Основными режимами работы ГВКУ являются текстовой и графический. В текстовом режиме выводится только текстовая информация, в графическом – и текстовая и графическая. Некоторые типы ПЭВМ (например, ПК8020) не имеют разграничений этих двух режимов и позволяют всегда выводить и текстовую и графическую информацию.

Дисплейный процессор устанавливает также разрешающую способность ГВКУ. Переключение режимов работы ГВКУ осуществляется в Бейсике обычно с помощью оператора SCREEN или с помощью специального оператора для каждого режима (см. HR, HGR). Установленный режим действует до следующего аналогичного оператора.

В случае развитого дисплейного процессора одним оператором SCREEN задаются режимы работы ГВКУ (текстовый или графический) и их основные характеристики:

разрешающая способность;

включение или отключение цветности;

номер активной и отображаемой страницы (для текстового и/или графического режима);

способ работы с отдельными УВВ (печать, магнитофон);

способ работы с динамическими изображениями.

Для ПЭВМ, у которых дисплейный процессор не поддерживает управления всеми указанными выше характеристиками, оператор SCREEN может иметь усеченный вид (см. SCREEN).

В некоторых версиях языка Бейсик возможно непосредственное управление дисплейным процессором, что ускоряет работу с графикой, но требует знания аппаратных возможностей процессора (см. VDP, VPEEK, VPOKE).

Обычно каждому режиму соответствует определенный характер работ, которые можно выполнять только в нем.

Так, в BASICA в режиме высокого разрешения изображение монохромное и число точек на экране будет равно 640×200. Кроме того, вывод текстовой информации в этом режиме производится в формате 25×80 (25 строк по 80 символов каждая).

В режиме среднего разрешения, где только и возможны цветные изображения, число точек на экране будет равно 320×200 и текстовая информация выводится в формате 25×40.

В MSX-BASIC имеются четыре режима работы, из которых два — текстовые (0,1) и два — графические (2,3). При этом вывод текста в графических режимах (например, оператором PRINT) ограничен и должен задаваться специальным образом с применением устройства с именем "GRP:".

В текстовом режиме существуют понятия активной и отображаемой страниц. Отображаемая страница — это та часть буферной памяти, из которой выводится информация на экран в настоящий момент. Активная страница — это область буферной памяти, которая служит для накопления информации для последующего вывода. Оператор SCREEN в версии языка BASICA позволяет управлять переключением номера активной и отображаемой страниц. Оператор SCREEN в Бейсик-КОРВЕТ позволяет переключать активные и отображаемые области графической памяти.

6.3. Построение изображения (статическая и динамическая графика)

В настоящее время вся графика ПЭВМ делится на два класса: изобразительную (коммерческую) и анимационную. В первом случае речь идет о построении на экране некоторых статических изображений (диаграмм, схем, гистограмм, кривых и т. п.). Во втором случае, особенно в игровых программах, требуется быстро перемещать картинки в определенную область экрана, создавать столкновения разных картинок, видоизменять картинки в зависимости от их местоположения и связи с другими картинками и т. п. Этот процесс позволяет "оживлять" изображение на экране, подобно тому, как это делается в мультипликационном (анимационном) фильме.

Язык Бейсик в более поздних версиях поддерживает оба класса графики. Для этого разработаны специальные операторы (см. PUT, GET, SPRITE, PUT SPRITE, ON SPRITE GOSUB). Понятие спрайт (от англ. sprite) — "дух", "эльф", стало профессиональным термином для определения картинки, перемещающейся по экрану без разрушения фонового изображения. Как правило, для "оживления", "анимации" статического изображения необходимо провести три операции:

1. Определить некоторый шаблон, обычно фиксированного размера (8×8, 16×16 или более точек) в определенной области памяти. Возможна специальная область памяти, предназначенная только для этой цели. В картинке-шаблоне в "темные" места (передний фон самой картинки) помещаются единицы, а в остальные места (прозрачные для фона изображения) — нули. Для больших изображений логично создать несколько таких шаблонов (см. GET SPRITE$).

2. Собрать все шаблоны и поместить их в нужное место памяти (см. PUT SPRITE). При этом устанавливается цвет "темных" мест в шаблоне. "Светлые" места шаблонов будут сохранять цвет изображения, на котором они находятся. Выполняя оператор PUT SPRITE в цикле и меняя каждый раз координаты точек экрана, в которые помещается спрайт, имитируется движение картинки по экрану. Оператор PUT SPRITE позволяет легко проводить эту операцию, использование оператора PUT требует дополнительных действий.

3. Определить момент, когда движущиеся изображения будут находиться в одной области экрана, т. е. когда их темные места пересекутся. В этом случае вызывается прерывание, которое обрабатывается программным путем (см. ON SPRITE GOSUB, SPRITE ON/OFF/STOP).

6.4. Операторы звука и музыки

Звуковое сопровождение программ на языке Бейсик может выполнять как вспомогательные функции, так и быть самоцелью. В первом случае звук используется для следующих целей:

фиксации определенных действий с аппаратурой (нажатие клавиши, сигнал задания режима, аварии и т. п.);

имитации реальных шумов, особенно в игровых программах (например, выстрел, столкновение машин, движение автомашины и т. п.);

музыкального сопровождения выполнения игровой программы для создания игровой обстановки.

Во втором случае программа на языке Бейсик пишется только в целях воспроизведения какого-либо музыкального произведения и реализуется на ПЭВМ, которые могут подключаться к электромузыкальным инструментам. Это направление в последнее время быстро развивается, причем уже разработан специальный интерфейс, получивший наименование MIDI (Musical Interface for Digital Instruments).

В простейшем случае для воспроизведения звука в ПЭВМ используется встроенный динамик. Управляя частотой генератора звука, можно получать простые звуковые сигналы или одноголосные музыкальные произведения. Длительность и частота сигнала задаются программируемым таймером.

В более развитых ПЭВМ имеется специальный процессор звука, который позволяет управлять несколькими голосами, звучащими одновременно, задавать шумовые эффекты, управлять громкостью каждого голоса, запоминать музыкальные фразы и т. д.

Для управления звуком бо́льшая часть версий языка Бейсик использует три оператора: BEEP, SOUND и PLAY.

Оператор BEEP генерирует звуковой сигнал, обычно с частотой 800 Гц в течение 0,25 с.

Оператор SOUND дает возможность управлять частотой и длительностью звучания. Используя фиксированную таблицу нот, можно воспроизводить простые музыкальные фразы. Предоставляется также возможность создавать с помощью оператора SOUND шумовые эффекты.

Оператор PLAY позволяет переводить на язык компьютера все основные музыкальные термины – от номера октавы и ноты в ней до темповых (легато, модерато, стаккато, ларго и т. п.) и интонационных характеристик обычной музыкальной записи.

Для этого оператор PLAY имеет внутренний макроязык, позволяющий осуществлять перевод нотной записи в компьютерную.

6.5. Описание инструкций

Таблица 6.2

Инструкция	Номер версии									
	1	2	3	4	5	6	7	8	9	10
ATTR			+							
BASE									+	

Окончание табл. 6.2

Инструкция	Номер версии									
	1	2	3	4	5	6	7	8	9	10
BEEP		+	+	+	+	+	+	+	+	+
BORDER			+							
BRIGHT			+							
CIRCLE	+		+				+	+	+	+
CLS		+			+	+	+	+	+	+
COLOR	+			+			+	+	+	+
DRAW	+			+				+	+	+
FLASH			+	+						
GET										+
GR				+						
HGR				+						
INK				+						
INVERSE			+	+						
LINE	+						+	+	+	+
LUT							+	+		
MGR				+						
NORMAL				+						
ON SPRITE									+	
OVER			+							
PAINT	+						+	+	+	+
PAPER			+							
PCLS							+	+		
PLAY			+					+	+	+
PLOT			+	+						
PLOT TO			+	+						
POINT	+		+				+	+	+	+
PRESET							+	+	+	+
PSET							+	+	+	+
PUT										+
PUT SPRITE									+	
RELOC							+	+		
RIBBON				+						
ROT			+							
SCALE			+							
SCREEN	+						+	+	+	+
SCREEN$			+							
SCRN				+						
SOUND								+	+	+
SPRITE									+	
SPRITE$									+	
TEXT				+						
VDP									+	
VPEEK									+	
VPOKE									+	
XDRAW			+							

6.5.1. Описание графических инструкций

ATTR – функция, которая сообщает значения атрибутов

ATTR (строка, столбец)

Двумя аргументами этой функции являются номер строки и номер столбца, а результатом работы функции является некоторое число, которое после расшифровки сообщает о цвете "бумаги" и "чернил", а также о других атрибутах соответствующей позиции на экране.

Результирующее число является суммой четырех чисел:

128 – если атрибут мерцания установлен и 0 – если равен нулю;

64 – если атрибут яркости установлен и 0 – если не установлен;

8×(цвет "бумаги");

1×(цвет "чернил").

Например, если в позиции экрана установлено мерцание, нормальная яркость, желтая бумага и синие чернила, то четыре числа, которые должны складываться при получении результата работы функции следующие: 128,0,8·6=48 и 1, т. е. результат равен 177.

Например

```
PRINT AT 0,0;FLASH 1;PAPER 6;INK 1;"    ";ATTR(0,0)
```

BASE – используется как псевдопеременная

BASE (таблица входов) = указатель видеопамяти

X=BASE (таблица входов)

таблица входов – любое числовое выражение от 0 до 19;

указатель видеопамяти – любое числовое выражение от 0 до 14336, значение которого является допустимым адресом видеопамяти.

Использование инструкции BASE требует знания дисплейного процессора ПЭВМ типа MSX (типа TMS–9918/28).

Инструкция BASE применяется для запроса и определения местоположения таблиц видеопамяти (VDP RAM). Двадцать возможных параметров соответствуют двадцати системным переменным (каждая по 2 байта). Эти параметры разбиты на четыре группы. Каждая из этих групп, содержащая по пять параметров, определяет базовые адреса видеопамяти соответственно для режимов SCREEN 0, SCREEN 1, SCREEN 2, SCREEN 3.

Если BASE находится в левой части оператора присваивания, а параметры соответствуют текущему режиму SCREEN, то регистры VDP корректируются немедленно. В остальных случаях с помощью псевдопеременной BASE можно прочитать значения системных переменных.

При задании оператора SCREEN эти переменные задают значения в регистрах VDP. Если BASE находится в левой части оператора присваивания и значение, вычисляемое по выражению, лишено смысла для регистра VDP, заданного параметром, то фиксируется ошибка Illegal function (Неверная функция).

Для каждого режима SCREEN имеется пять параметров:

0 – базовый адрес таблицы имен;

1 – базовый адрес таблицы цветов (не используется в SCREEN 0 и 3);

2 – базовый адрес таблицы генератора шаблонов;

3 – базовый адрес таблицы атрибутов спрайтов (не используется в SCREEN 0);

4 – базовый адрес таблицы шаблонов спрайтов (не используется в SCREEN 0).

Таким образом, параметр для BASE определяется так:

BASE (режим SCREEN*5 + смещение относительного адреса базы)

Инструкция BASE в отличие от SCREEN не выполняет инициализацию режима экрана. Она избавляет от вычислений, необходимых для преобразования нужных адресов BASE( ) в требуемые значения и помещения их в регистры VDP.

Эта псевдопеременная позволяет одновременно вводить в видеопамять содержимое нескольких различных кадров дисплея, возможно даже в нескольких режимах, по желанию переходя от одного режима к другому.

Если BASE используется для установки адресов таблиц, то они не меняют положение курсора, оставляя его в положении, заданном в последнем операторе SCREEN. Например:

```
10 COLOR 1,15
20 SCREEN 0
30 FOR N=0 TO 19
40 PRINT N"=";BASE(N),
50 NEXT N
RUN
0       0       2048      0       0
6144    8192    0         6912    14336
6114    8192    0         6912    14336
2048    0       0         6912    14336
```

Пять чисел в первой строке для режима SCREEN 0:

0 — адрес таблицы символов равен 0;

0 — для данного режима не используется (база таблицы цветов);

2048 — адрес таблицы начертаний символов (побайтно);

0 — для данного режима не используется (база атрибутов спрайтов);

0 — для данного режима не используется (база шаблонов спрайтов);

Пять чисел во второй строке для режима SCREEN 1:

6144 — адрес таблицы символов;

8192 — адрес таблицы цветов соответствующих символов;

0 — для данного режима не используется;

6912 — адрес базы атрибутов спрайтов;

14336 — адрес базы шаблонов спрайтов

и так далее для режимов SCREEN 2 и SCREEN 3.

BORDER — оператор, устанавливающий цвет границы бордюра экрана

BORDER n

n — от 0 до 7.

BRIGHT — оператор, устанавливающий яркость изображения символа

BRIGHT n

n — равно 0 или 1.

Оператор BRIGHT устанавливает яркость символа, которая может быть нормальной (n=0) или повышенной (n=1). Оператор BRIGHT называется атрибутом символа (см. ATTR, INK).

CIRCLE — оператор, который рисует на экране дисплея эллипс с центром в точке (X,Y) заданным радиусом

CIRCLE [STEP] (X,Y),радиус[,цвет[,начало[,конец[,аспект]]]]

X — координата по горизонтали, целое число со знаком;

Y — координата по вертикали, целое число со знаком;

радиус – расстояние от точки (X,Y), целое число без знака (в допустимом диапазоне см. 6.1);

цвет – номер цвета 0 < [допустимого номера] (см. 6.1);

начало, конец – начало и конец дуги;

аспект – форма эллипса, безразмерная величина.

Координаты центра могут быть заданы абсолютным или относительным способом. Оператор CIRCLE не отображает те точки, координаты которых выходят за границы экрана. После выполнения оператора CIRCLE точка (X,Y) считается последней выполненной.

Параметр цвет задается явно или по умолчанию. По умолчанию он совпадает с цветом переднего фона, установленным в параметре COLOR.

Начало и конец – угловые параметры, задаются в радианах и могут изменяться от -2π (-6.21318) до 2π ($+6.21318$), где $\pi = 3.141595$. Они позволяют задать начало и конец рисунка эллипса. Если значения углов отрицательны (–0 не допускается), крайние точки дуги будут соединяться линиями с центром, а значения углов будут обрабатываться так, как если бы они были положительными (это не то же самое, что прибавление $2*\pi$). Параметр начало может быть меньше параметра конец.

Параметр аспект – это отношение X-радиуса к Y-радиусу. Он позволяет управлять формой эллипса. Если аспект меньше 1, то заданный радиус является Y-радиусом, т. е. эллипс вытянут по оси Y. Если аспект больше 1, то определяется X-радиус, т. е. эллипс вытянут по оси X. Обычно значение параметра аспект изменяется от 1/260 до 260.

В разных ПЭВМ существует стандартное (по умолчанию) значение аспекта, соответствующее окружности (для ПК8020/10 это значение равно 3/2, для PC IBM оно составляет в среднем разрешении 5/16, а в высоком разрешении 5/12, для MSX – 4/3).

Например:

```
10 REM "Установите соответствующий режим"
20 CIRCLE(100,100),60,1
30 CIRCLE(100,100),40,1
40 CIRCLE(100,100),20,
```

Рисуются три концентрические окружности.

```
10 COLOR 4
20 CIRCLE(128,96),50,,-1,-3
30 GOTO 30
```

Рисуется сектор.

```
10 COLOR 4,15
20 SCREEN 2
30 CIRCLE(128,96),50,,-1,-3
40 GOTO 40
```

Рисуется сектор для MSX-BASIC.

В Бейсик-Спектрум+2 используется упрощенная форма записи

```
CIRCLE X,Y,радиус
```

для изображения окружности.

CLS – оператор очистки экрана

```
CLS
```

Обычно CLS устанавливает весь экран в цвет фона. В BASICA, MSX-BASIC и аналогичных версиях CLS действует по-разному в текстовом и графических режимах. В текстовом режиме

очищается активная страница (красится в цвет фона) и курсор возвращается в верхнее левое положение (0,0).

В графическом режиме видеопамять красится в цвет фона и курсор устанавливается в центр экрана, в точку с координатами (160,100) для среднего разрешения и в точку (320,100) для высокого разрешения.

В ПК8020/10 оператор CLS стирает всю алфавитно-цифровую информацию с экрана. Для чистки графической видеопамяти применяется другой оператор – PCLS (см. ниже).

COLOR – оператор, который используется для установки цветов

Формат 1 (только для BASICA).

COLOR[передний фон][,[задний фон]][,[бордюр]]

Формат 2 (только для BASICA).

COLOR[задний фон][,[палитра]]

Формат 3 (только для MSX-BASIC).

COLOR[цвет][,[цвет]][,цвет]]

Формат 4 (только для Бейсик-ПК8010, Бейсик-ПК8020).

COLOR[передний фон][,задний фон]

Формат 5 (только для Бейсик-АГАТ).

COLOR=арифметическое выражение

Формат 1 (текстовый режим):

передний фон – цвет символа (числовое выражение от 0 до 31);

задний фон – цвет фона (числовое выражение от 0 до 7);

бордюр – цвет границы экрана (числовое выражение в расширенном диапазоне).

Оператор распространяет свое действие на символы, которые будут выводиться на экран в дальнейшем, цвет символов, уже находящихся на экране, не меняется.

Можно не указывать любой из параметров, но хотя бы один должен быть задан. Если параметры не указаны, то сохраняется действие предыдущего оператора COLOR.

Чтобы получить мерцающее изображение символа, необходимо задать в качестве цвета переднего плана номер цвета, увеличенный на 16.

Для работы с черно-белым экраном используются следующие значения оператора COLOR:

1. Для переднего фона: 0 – черный; 1 – подчеркнутый символ с белым передним фоном; 7 – белый; 15 – интенсивно-белый.

2. Мерцание символа получается указанным выше способом (16 – это черный мерцающий).

3. Для заднего фона можно выбрать два цвета: 0 – черный или 7 – белый.

Если оператор COLOR заканчивается запятой (,), то выдается сообщение Missing operand (Неверный операнд), но цвет изменяется. Любые значения вне разрешенного диапазона вызывают ошибку Illegal function call (Неверный функциональный вызов) и остаются предыдущие значения цветов.

Данный формат используется во всех версиях языка Бейсик для PC IBM (BASICA, GWBASIC).

Формат 2 (графический режим):

задний фон – цвет фона, числовое выражение от 0 до 15;

палитра – числовое выражение.

Если значение выражения – четное число, устанавливается палитра 0, если нечетное – палитра 1 (см. 6.1).

Формат 3 (графический и текстовый режимы):

цвет – цвет изображения, фона или бордюра в разрешенном диапазоне.

Оператор устанавливает цвета переднего и заднего фонов, бордюра, которые затем действуют по умолчанию до следующего оператора COLOR.

Любой не указанный операнд сохраняет свое предыдущее значение. Если числовое значение выходит за разрешенный диапазон, вызывается сообщение Illegal function call (Неверный функциональный вызов). Отсутствие параметров приведет к сообщению Missing operand (Неверный операнд).

В данном формате оператор COLOR описан в ГОСТ 27787–88 на язык программирования Бейсик и применяется в MSX-BASIC.

Оператор используется в режиме среднего разрешения, в режиме высокого разрешения использование неправомочно. Цвет бордюра всегда совпадает с цветом заднего фона. Сообщения об ошибках те же, что и в *формате 1*.

Например:

```
10 REM"Установите режим"
20 COLOR 1,4
30 CIRCLE(100,100),20
40 CIRCLE(100,100),40,7
50 CIRCLE(100,100),60
60 GOTO 60
```

Действие оператора COLOR распространяется на строку 30 и 50. В строке 40 локально определен новый цвет, но только на время действия этого конкретного оператора. Строка 60 только для MSX-BASIC.

Формат 4.:

передний фон – значение целочисленного выражения от 0 до 7;

задний фон – значение целочисленного выражения от 0 до 7.

Устанавливает цвета, используемые в графических операторах. Задний фон используется в качестве неявного параметра в операторах PCLS и PRESET при удалении изображения или точки.

Задаваемые оператором COLOR цвета с помощью таблицы LUT (см. LUT) преобразуются в физические цвета. При выходе значений параметров за пределы диапазонов, выводится сообщение НЕВЕРЕН ВЫЗОВ ФУНКЦИИ. Данный формат оператора COLOR применяется в Бейсик-ПК8010, Бейсик-ПК8020.

Формат 5 (графический режим). Значение арифметического выражения изменяется от 0 до 15 и используется для установки номера цвета отображаемых точек и линий. Данный формат оператора COLOR применяется в Бейсик-АГАТ.

DRAW – оператор, позволяющий рисовать произвольное изображение, определенное строкой

Формат 1 (кроме Бейсик-АГАТ и Бейсик-Спектрум+2).

```
DRAW строка
```

Формат 2 (только для Бейсик-АГАТ).

```
DRAW арифметическое выражение

[AT X,Y]
```

Формат 3 (только для Бейсик-Спектрум+2).

DRAW X,Y [,число]

Формат 1:

строка – строковое выражение, состоящее из команд графического макроязыка

Оператор DRAW позволяет рисовать сложные фигуры отрезками прямых линий. Каждая команда макроязыка представлена одним символом. Графический макроязык, используемый в операторе DRAW – это как бы язык в языке; он существует внутри языка Бейсик исключительно для выполнения операции рисования произвольных фигур. С помощью графического макроязыка можно управлять движением курсора, рисующего фигуру, что похоже на "черепашью графику", используемую в языке ЛОГО.

"Программа" на макроязыке состоит из строки длиной до 255 символов. Каждая команда представлена одним символом.

В одних версиях языка (например, Бейсик-КОРВЕТ) команды пишутся подряд, в других допускается разделение команд пробелами для удобства чтения.

Команды перемещения позволяют рисовать отрезки прямых по вертикали, горизонтали и диагоналям:

Un – вверх	**Fn** – вправо вниз
Dn – вниз	**En** – вправо вверх
Rn – вправо	**Gn** – влево вниз
Ln – влево	**Hn** – влево вверх

где n – число точек экрана, на которые нужно передвинуть курсор. В случае диагональных команд n=n(x)=n(y).

Все команды перемещения производят движение из точки, определенной предыдущей координатой. В начальный момент положение курсора перед командой DRAW определено предыдущим графическим оператором. Рисунок 6.3 иллюстрирует действие команд перемещения.

Команда M(X,Y) пересылает курсор от текущей точки до точки (X,Y). Координаты (X,Y) могут быть относительными и абсолютными. Если перед X стоит знак "+" или "–", пересылка относительная, если нет – абсолютная.

Любой команде перемещения могут предшествовать следующие команды:

B – переслать, но не рисовать;

N – переслать, но возвратить в начальную точку;

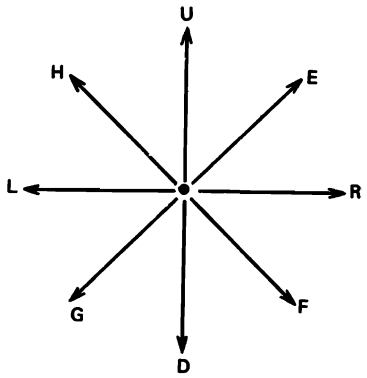


Рис. 6.3

A n – установить угол n, угол задается одним и значений: 0, 1, 2, 3, где 0–0 градусов, 1–9(градусов, 2–180 градусов, 3–270 градусов; команда вращает оси координат экрана дисплея против часовой стрелки на определенное число градусов и соответственно изменяет направление перемещения;

A0 – установить оси координат в начальное положение;

Sn – установить масштаб, где n изменяется от 0 до 255; команда устанавливает число точек экрана для всех последующих команд в соотношении n/4. Начальное значение обычно S4 или S0;

Cn – установить цвет рисуемой линии, где n – номер цвета; ввести строковую переменную,

которая используется как подстрока; перед именем переменной ставится X, а после имени — точка с запятой (;).

В любой из перечисленных выше команд числовой аргумент может быть заменен переменной. Для этого после команды ставится знак "=" (равно) и после него пишется имя переменной. Точка с запятой (;) ставится после имени, обозначая его конец. В команде M точка с запятой не заменяет запятую.

Как указывалось в описании оператора CIRCLE (см. выше), при рисовании изображений необходимо учитывать искажение экрана (аспект экрана). Например:

```
10 BA$="UR0R60D40L60"
15 REM"Аспект ИК8020 равен 3/2"
20 DRAW "BM100,100,S0XBA$"
30 DRAW "S1XBA$"
40 DRAW "S2XBA$"
```

Данная программа рисует три квадрата, определенных в строке 10. Комментарий (строка 15) объясняет количество точек в командах перемещения.

Например:

```
10 COLOR 1,4
20 SCREEN 2
30 FOR N=4 TO 100 STEP 4
40 DRAW "BM 128,96 S=N; R4 U4 L4 D4"
50 NEXT N
60 DRAW "S4"
70 GOTO 70
```

Установка режима в примере проведена для MSX-BASIC. Обратите внимание на использование команды S=N; и S4.

Формат 2. Оператор DRAW в этом формате воспроизводит на экране в режиме высокого разрешения шаблон или, как еще его называют, "маску", определенную последовательностью команд, расположенных в специальной таблице. (Иначе этот шаблон называется "описателем формы".) Таблица состоит из последовательности закодированных векторов, определяющих форму объекта. Эта таблица создается до выполнения Бейсик-программы. Каждый байт шаблона разделен на три секции, каждая из которых описывает вектор, указывающий направление движения воображаемого "пера", рисующего объект. По команде DRAW интерпретатор просматривает секцию за секцией шаблона. Когда встречается байт, все биты которого нули, просмотр заканчивается.

Для использования таблицы в Бейсик-программе необходимо загрузить ее в память средствами Ассемблера или с помощью оператора POKE.

Если в команде введен параметр **AT**, то начало рисунка совпадает с точкой с координатами (X,Y). Если параметр AT не вводится, рисование начинается с точки, определенной операторами PLOT, DRAW или XDRAW. Значения выражения изменяются от 0 до 255. Диапазоны изменения X и Y определены координатами экрана.

Формат 3. Служит для рисования прямой линии или дуги:

X,Y — целые числа в разрешенном диапазоне;

число — целое число (в радианах).

Рисование с помощью оператора DRAW начинается либо в той точке, где закончилось последнее предложение PLOT, DRAW или CIRCLE, либо в левом нижнем углу (после операторов RUN,CLEAR CLS или NEW).

Конечная точка линии — это точка, отстоящая на X графических точек вправо и на Y влево от исходной. Команда DRAW сама не определяет начальную точку. Числа X и Y могут быть отрицательными.

При задании параметра **число** можно рисовать дуги, причем **число** — это число радиан, на которое производится поворот при построении линии. Если **число** > 0, то поворот происходит влево, если **число** < 0, то вправо. При этом надо учесть, что полный поворот происходит при **число**=2π, при **число**=π будет нарисована полуокружность, при **число**=0,5π — ее четверть и т. д. Например:

```
10 PLOT 100,100 : DRAW 50,50,PI
```

FLASH — оператор, устанавливающий атрибут мерцания символа

Формат 1 (только для Бейсик-Спектрум+2).

```
FLASH n
```

Формат 2 (только для Бейсик-АГАТ).

```
FLASH
```

Формат 1:

n равно 0 или 1.

Оператор FLASH устанавливает атрибут мерцания для символа (непрерывная замена цветов "чернил" и "бумаги"). Мерцание вводится при n=1 и отменяется при n=0.

Формат 2. Используется для установки мерцающего режима вывода текста.

GET — оператор, позволяющий считать информацию об изображении всех точек заданной области экрана и поместить ее в числовой массив

```
GET(X1,Y1)-(X2,Y2), имя массива
```

X1, Y1, X2, Y2 — координаты, заданные в абсолютной или относительной форме;

имя массива — имя области, в которой информация будет запоминаться.

Оператор GET является парным с оператором PUT. С помощью этих двух операторов можно создавать динамическую графику. Процесс основан на управлении передвижением прямоугольных фрагментов, состоящих из "светлых" и "темных" пятен. Впечатление движения создается последовательной сменой "кадров" с помощью операторов PUT и GET.

Операнд (X1,Y1)–(X2,Y2) определяет прямоугольную область, задавая координаты вершин прямоугольника.

Имя массива определяет область памяти, в которую будет помещаться информация, полученная оператором GET. Информация запоминается в массиве последовательно. Необходимо, чтобы размер массива был равен емкости области памяти в буфере.

Необходимая величина области памяти в байтах составляет 4 + INT((X*A+7)/8)*Y, где X,Y — длины соответственно горизонтальной и вертикальной сторон прямоугольника (в графических точках);

A — равно 2 при среднем разрешении и 1 при высоком.

Информация в массиве имеет следующую структуру:

два байта, содержащие размер X (бит);

два байта, содержащие размер Y (бит);

данные.

В среднем разрешении прямоугольник 10×12 графических точек будет эквивалентен 40 байтам массива.

GR — оператор установки графического режима

```
GR=n
```

n — номер страницы памяти.

Оператор GR устанавливает графический режим с низким разрешением (64×64 точки), **n** — номер страницы памяти размером 2 Кбайта, при этом в ПЭВМ АГАТ **n** изменяется от 2 до 31. Оператор очищает экран.

HGR — оператор установки графического режима

HGR=n

n — номер страницы памяти.

Оператор HGR устанавливает графический режим с высоким разрешением (256×256 точек), **n** — номер страницы памяти размером 8 Кбайт. В ПЭВМ АГАТ **n** изменяется от 1 до 7. Оператор очищает экран.

INK — устанавливает цвет переднего фона

INK число

число — числовое выражение от 0 до 9.

В соответствии с образами, введенными в Бейсик-Спектрум, для операции графики всегда "чернилами" рисуют на "бумаге". "Чернила" (INK) — это передний фон символа. Оператор INK может задавать восемь (0 — 7) цветов или цвет 9, определяемый как "контрастный". При задании цвета 9 автоматически установится цвет "чернил", противоположный тому, в который установлена "бумага" (PAPER).

Цвет 8 определяется как "прозрачный" в том смысле, что сохраняется старый цвет. Оператор INK относится всегда к определенному элементу экрана и называется атрибутом (см. PAPER, ATTR). Например:

```
INK 9 : FOR C=0 TO 7 : PAPER C : PRINT C : NEXT C
```

INVERSE — оператор инвертирующий цвет точки

Формат 1 (только для Бейсик-Спектрум+2).

INVERSE n

Формат 2 (только для Бейсик-АГАТ).

INVERSE

Формат 1:

n равно 0 или 1.

Оператор INVERSE инвертирует цвет точки. При использовании оператора INVERSE в контексте работы с символом операнд n=1 изменит образ каждой точки в позиции экрана таким образом, что цвет "бумаги" будет заменен на цвет "чернил", и наоборот.

При использовании INVERSE в графических операторах его действие распространяется только на одну точку.

Формат 2. Инвертирует цвет точки в режиме вывода текста.

LINE — оператор изображения линий и прямоугольников

```
LINE[STEP[(X0,Y0)]]-[STEP](X1,Y1)[,цвет][,B[F]]
```

STEP — указывает на возможность задания координат точки в относительной форме;

X0, Y0 — координаты начальной точки;

X1, Y1 — координаты конечной точки;

цвет — номер цвета (используются только цвета переднего фона);

B — указывает, что рисуется прямоугольник заданным цветом;

F — указывает, что прямоугольник заполняется этим цветом.

При задании координат необходимо учитывать аспект экрана.

Оператор LINE рисует линию от точки X0,Y0 до точки X1,Y1 указанным цветом или цветом переднего фона. Точки, имеющие координаты за пределами экрана, не изображаются. Если в операторе задан параметр B, то рисуется прямоугольник с диагональными вершинами в точках (X0,Y0) и (X1,Y1).

Простейшей формой LINE является

```
LINE-(X1,Y1)
LINE STEP (X1,Y1)
```

При этом рисуется линия до точки (X1,Y1) цветом переднего фона. Последней точкой оператора LINE всегда является точка с координатами (X1,Y1).

Например:

```
10 LINE(0,200)-(511,100),7,B
20 LINE(0,100)-(511,200),2
30 LINE(511,100)-(0,200),4
```

Например (для MSX BASIC):

```
10 COLOR 1,15
20 SCREEN 2
30 DRAW "BM128,96"
40 FOR N=1 TO 10
50 LINE STEP(0,0)-STEP(10,10),,B
60 NEXT N
70 GOTO 70
```

LUT — оператор, определяющий физические цвета. (Только для Бейсик-КОРВЕТ.)

```
LUT X(1)
```

X(i) — имя первого элемента целочисленного массива, содержащего элементы X(i), X(i+1), ..., X(i+15).

Программирует таблицу цветов. Элементы массива с индексами от i до i+15 содержат номера физических цветов, а их порядковый номер соответствует логическим номерам цветов.

При начальной инициализации логические номера цветов от 0 до 7 соответствуют цветам графики (без алфавитно-цифровых символов), а от 8 до 15 — цветам графики с алфавитно-цифровой информацией.

Соответствие номеров в таблице физическим цветам при начальной инициализации представлено в операторе COLOR.

Например:

```
10 DIM XX(15)
20 FOR I=0 TO 7
30 LINE(100,100)-(200,200),4,BF
40 XX(I)=1 : NEXT
50 FOR I=8 TO 15
60 XX(I)=15 : NEXT
70 LUT XX(0)
```

MGR — оператор установки графического режима

```
MGR=n
```

n — номер страницы памяти, изменяется от 1 до 7.

Устанавливает графический режим со средним разрешением (128×128 точек); n — номер страницы памяти размером 8 Кбайт.

NORMAL – оператор, устанавливающий режим вывода текстовой информации

```
NORMAL
```

Отменяет действие операторов INVERSE и FLASH и устанавливает режим вывода текстовой информации в прямом контрасте и без мерцания.

ON SPRITE GOSUB – обрабатывает прерывания при совмещении спрайтов

```
ON SPRITE GOSUB номер строки
```

Оператор ON SPRITE GOSUB связан с аппаратной возможностью фиксировать прерывания в случае, когда на двух или более плоскостях спрайты находятся в одних и тех же точках экрана. Такая аппаратная возможность существует в ПЭВМ класса MSX.

Когда спрайты (вернее, их "темные пятна") накладываются друг на друга в одной и той же точке экрана, оператор ON SPRITE фиксирует такое событие и обрабатывает его программным путем, передавая управление подпрограмме обработки прерывания.

OVER – задает наложение символов или точек

```
OVER n
```

n – имеет значение 0 или 1

Оператор OVER активизирует операцию наложения. В текстовом режиме при записи в позицию на экране символа старое значение удаляется. При использовании OVER 1 новый символ накладывается на старое изображение.

При использовании OVER в графических операторах цвет точки изменится. Например:

```
PLOT 0,0:DRAW OVER 1;250,175
```

PAINT – закрашивает область экрана, ограниченную замкнутым контуром

```
PAINT [STEP](X,Y)[,цвет закраски[,цвет границы]]
```

STEP – указывает координаты относительно последней точки;

X,Y – координаты точки в пределах области, которая будет закрашиваться; могут быть заданы в абсолютной или относительной форме;

цвет закраски – номер цвета закраски;

цвет границы – номер цвета изображения границы области.

Закраска контура начинается с точки (X,Y), которая может находиться в произвольном месте внутри контура, но не на его границе. Закраска производится заданным цветом до достижения границы области. Если координаты точки находятся вне контура, будет закрашиваться весь экран, за исключением замкнутого контура.

Как правило, PAINT работает совместно с другими операторами, позволяющими строить контур (см. CIRCLE, DRAW, LINE).

Если операнды задания цвета не указаны, то по умолчанию действуют цвета, установленные оператором COLOR. Оператор PAINT присутствует во всех графических версиях языка Бейсик, кроме Бейсик-АГАТ и Бейсик-Спектрум+2. Например (Бейсик-КОРВЕТ):

```
10 COLOR 1,7
20 FOR N=1 TO 3
30 CIRCLE(150,100),N*10,N
40 NEXT N
50 PAINT(150,100),3,1
```

Например (MSX-BASIC):

```
10 COLOR 1,15
20 SCREEN 3
30 FOR N=3 TO 6
```

```
40 CIRCLE(129,96),N*10,N
50 NEXT N
60 PAINT(128,96),3,3
70 GOTO 70
```

PAPER – оператор, устанавливающий цвет фона

PAPER число

число – числовое выражение от 0 до 9.

В соответствии с образами, введенными в Бейсик-Спектрум+2, рисуют всегда на "бумаге" "чернилами". "Бумага" (PAPER) – это задний фон. Весь экран разделяется на позиции (клетки), в которых рисуют символы. Таких клеток может быть 24 × 32 = 768.

Оператор PAPER устанавливает цвет фона одним из 8 цветов (0 – 7), цвет 8 – прозрачный, цвет 9 – контрастный. Например:

```
INK9:PAPER8:PRINT AT(0,0);:FOR n=1 TO 1000:PRINT
n:NEXT n
```

PCLS – очищает видеопамять

PCLS

Действие оператора аналогично действию оператора CLS, но производится оно над графической информацией. Видеопамять очищается цветом заднего фона, графическая информация на экране исчезает.

PLOT – рисует точку на экране

PLOT X,Y

X,Y – координаты точки в разрешенном диапазоне.

Рисует точку на экране с координатами (X,Y). Цвет точки устанавливается отдельными операторами (см. COLOR, PAPER, INK, RIBBON).

В случае выхода координат точки за заданные пределы выводится сообщение INTEGER OUT OF RANGE (Аргумент вне диапазона). Имеется в Бейсик-АГАТ и Бейсик-Спектрум+2.

Например:

```
10 FOR N=0 TO 255
20 PLOT N,88+80*SIN(N/128*PI)
30 NEXT N
```

PLOT TO – строит отрезок прямой или ломаной линии

PLOT TO X,Y

PLOT X1Y1 TO X2Y2

PLOT X1Y1 TO X2Y2 TO X3Y3...

XnYn – координаты точки.

Строится отрезок прямой с началом в текущей точке (PLOT TO X,Y) или в точке, заданной координатами (Xi,Yi), и с концом в точке, заданной координатами (Xn,Yn).

Возможно построение ломаной линии из m отрезков. Имеется только в Бейсик-АГАТ.

POINT – функция, с помощью которой считывается цвет точки

```
Y=POINT(X,Y)
```

X,Y – координаты точки в абсолютной форме.

Функция позволяет считать цвет точки и передать номер цвета вызывающему оператору. Если точка лежит вне экрана, то передается значение –1. В остальных случаях передается номер цвета из установленного диапазона (0 ... 7 в Бейсик-КОРВЕТ, 0 ... 15 в MSX-BASIC, в BASICA 0 ... 2 в среднем разрешении и 0 или 1 в высоком).

В Бейсик-Спектрум+2 функция POINT передает в программу только 0 или 1; 0 – в случае, если графическая точка цвета "бумаги", и 1, если графическая точка цвета "чернил" (см. PAPER, INK).

Выводится сообщение об ошибке INTEGER OUT OF RANGE (B), если 0 > X > 255 и 0 > Y > 175.

Аналогом POINT в Бейсик-АГАТ является функция SCRN(X,Y). Функция обычно используется для проверки достижения определенной области на экране или для проверки наличия точки на экране.

Например:

```
10 IF POINT(I,1)<>0 THEN PRESET(I,1) ELSE
PSET(I,1)
```

PRESET – удаляет точку на экране дисплея

```
PRESET [STEP](X,Y)[,цвет]
```

X,Y – координаты точки на экране;

цвет – числовое значение в заданном диапазоне.

Что означает удалить точку на графическом экране? Для этого существует две возможности: дать ей цвет заднего фона или сменить в данной точке цвет заднего фона на другой. Обе эти возможности реализует оператор PRESET.

Если операнд цвета не указан, применяется по умолчанию цвет заднего фона. Если значения цвета выходят за пределы разрешенного диапазона, выводится сообщение Illegal function call (Неверный функциональный вызов).

Если координаты точки заданы вне области экрана, то не производится никаких действий и не выводится сообщение об ошибке.

PSET – рисует точку в заданном месте экрана заданным цветом

```
PSET [STEP](X,Y)[,цвет]
```

X,Y – координаты точки экрана, в которой необходимо нарисовать графическую точку, могут задаваться в абсолютной или относительной форме;

цвет – числовое значение в заданном диапазоне.

Оператор позволяет нарисовать точку в произвольном месте экрана. Если операнд цвет не указан, принимается по умолчанию цвет переднего фона.

Если значение цвета выходит за пределы разрешенного диапазона, выводится сообщение Illegal function call (Неверный функциональный вызов). Если координаты точки заданы вне области экрана, то не производится никаких действий и не выводится сообщение об ошибке.

Например:

```
10 REM "Установите режим"
20 FOR N=100 TO 120
30 FOR M=80 TO 100
40 PSET(M,N)
50 NEXT M.N
```

```
60 FOR N=100 TO 120
70 FOR M=80 TO 100
80 PRESET(X,Y)
90 NEXT M
100 NEXT N
110 GOTO 110
```

PUT – оператор воспроизводит на экране изображение, которое хранится в числовом массиве

PUT(X,Y),имя массива[,действие]

X,Y – координаты левого верхнего угла передаваемого на экран изображения. Задается в абсолютной или относительной форме;

имя массива – имя области памяти, которая содержит передаваемую информацию;

действие – одно служебное слово из следующих: PSET, PRESET, XOR, OR, AND.

Оператор PUT является парным оператору GET. С помощью этих двух операторов можно создавать динамическую графику. Процесс основан на управлении передвижением прямоугольных фрагментов, состоящих из "светлых" и "темных" пятен. При этом впечатление движения создается последовательной сменой операторов PUT и GET.

Оператор PUT воспроизводит на экране изображение, которое хранится в числовом массиве, созданном оператором GET. Эта картина, представленная числовым массивом, может быть передана в любое место экрана. Каким способом будет воспроизведена эта картинка на экране? Это определяет операнд действие.

Самый простой способ – использование служебного слова PSET. В этом случае картинка будет воспроизведена точно в том виде, в котором она сохранилась оператором GET.

При использовании служебного слова PRESET цвета всех точек картинки будут заменены на противоположные. Это означает, что цвет 0 заменяется на 3, и наоборот, а цвет 1 заменяется на 2, и наоборот (в среднем разрешении).

В обоих указанных выше действиях старый цвет точек экрана, на который выводится картинка, игнорируется.

При использовании служебных слов AND, OR и XOR производятся соответствующие логические операции над точками "старой" картинки (уже существующей на экране) и новой (из массива, созданного оператором GET).

Правила этих логических операций следующие (в среднем разрешении, поэтому номер цвета 0 – 3):

AND

Номер цвета	Значение точек массива			
	0	1	2	3
0	0	0	0	0
1	0	1	0	1
2	0	0	2	2
3	0	1	2	3

OR

Номер цвета	Значение точек массива			
	0	1	2	3
0	0	1	2	3
1	1	1	3	3
2	2	3	2	3
3	3	3	3	3

XOR

Номер цвета	Значение точек массива			
	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

Как видно из этих таблиц, AND используется только для пересылки картинки, если эта картинка уже существует "под" передаваемым изображением; OR используется для суперпозиции передаваемой картинки, существующие точки сохраняются. Особое значение имеет действие XOR:

если картинка накладывается на такое же изображение, то это изображение удаляется с экрана (восстанавливается цвет фона);

если проделать предыдущую операцию дважды, то прежнее изображение восстанавливается;

при изменении координат (X,Y) и повторении действия XOR создается иллюзия движения картинки по экрану;

если никакое действие в операторе не указано, по умолчанию выполняется действие XOR.

Если передаваемая картинка слишком велика, появляется сообщение Illegal function call (Неверный функциональный вызов), конечная точка не устанавливается.

PUT SPRITE – управляет размещением графического объекта

```
PUT SPRITE Nспрайта[,[STEP](X,Y)[,цвет][спробраз]
```

Nспрайта – арифметическое выражение, задающее один из 32 спрайтов движущихся изображений, с которыми будет производиться работа;

STEP – обеспечивает возможность относительной адресации;

X – арифметическое выражение, значение которого определяет X-координату (столбец), $-37768 <= X <= 32767$;

Y – арифметическое выражение, значение которого определяет Y-координату (строку), $-37768 <= Y <= 32767$;

цвет – арифметическое выражение в диапазоне 0 – 15 или 0 – 255 (в зависимости от значения параметров оператора SCREEN), значение которого определяет, какой из шаблонов (образов), созданных оператором SPRITE$, связывается со спрайтом;

спробраз – описание шаблона графического образа.

Оператор PUT SPRITE действует со спрайтами (движущимися изображениями), а также с образами, созданными с помощью оператора SPRITE$.

Оператор PUT SPRITE устанавливает связь образа со спрайтом. Его цвет и определяет местоположение на экране. Допускается 32 видимых спрайта с ограничением не более 4 спрайтов в одной строке.

Координаты X и Y всегда вычисляются по модулю размера экрана, причем точка (X,Y) задает верхний левый угол спрайта.

Значение координат может быть отрицательным, при этом спрайт размещается так, что он будет виден на экране частично.

Например, при Y=–1 спрайт поместится в самую верхнюю часть экрана, еще меньшее значение совсем выведет спрайт за пределы экрана. Спрайт двойного размера (32×32) полностью выйдет за пределы экрана при Y=–33.

Операнд цвет задает цвет "темных пятен" спрайта. По умолчанию присваивается цвет, совпадающий с цветом образа, при этом надо иметь в виду, что спрайт всегда одноцветный. Размерами спрайтов управляет оператор SCREEN. Сам спрайт можно представить в виде картинок на прозрачном носителе (типа слайда), которые накладываются на экран с графическим изображением и текстом. При этом как бы существует столько плоскостей наложения, сколько имеется спрайтов. Так, спрайт с номером 31 будет наложен прямо на изображение на экране, спрайт с номером 30 – на спрайт 31, поэтому спрайт с более низким номером имеет более высокий приоритет и накладывается всегда на спрайт с более высоким номером. Таким образом, удаляться с экрана в качестве лишнего спрайта будет спрайт с более высоким номером.

Разноцветные картинки могут быть получены двумя способами:

1. Создать картинку из нескольких спрайтов (используя оператор SPRITE$, в котором с помощью конкатенации объединяются несколько шаблонов разного цвета, например, разные

цвета рубашки и брюк, а также разные цвета рук и ног бегущего человека) и рассматривать этот спрайт как целое или как несколько спрайтов, находящихся всегда рядом.

2. Накладывать один спрайт на другой в разных плоскостях. Однако в этом случае нельзя использовать оператор ON SPRITE GOSUB.

Для ускорения работы со спрайтами часто используется оператор VDP. Использование таблиц шаблонов спрайтов и таблиц цветов спрайтов может быть ускорено с помощью оператора VPOKE.

Оператор PUT SPRITE имеется только в версии MSX-BASIC. Например:

```
10 SCREEN 2,0
20 COLOR 15,1,1
30 CLS
40 SPRITE$(0)=CHR$(126)+STRING$(6,CHR$(255)+
   CHR$(126))
50 SPRITE$1=STRING$(3,CHR$0))+CHR$(24)+CHR$(24)+
   STRING$(3,CHR$(0))
60 FOR I=0 TO 6.28 STEP 0.2
70 X=X+1.5
80 Y=Y+1
90 X1=30*COS(I)
100 Y1=30*SIN(I)
110 X2=15*COS(I)
120 Y2=15*SIN(I)
130 PUT SPRITE 0,(X,Y),11,0
140 PUT SPRITE 1,(X1+X,Y1+Y),9,1
150 PUT SPRITE 2,(X2+X,Y2+Y),15,1
160 NEXT
170 GOTO 60
```

Данный пример демонстрирует движение планет вокруг Солнца. Строка 10 устанавливает черный задний фон и белый передний. Строка 40 (спрайт 0) – это Солнце. Строка 70 и 80 – Солнце помещается на экран. Строки 90 – 120 – установка координат X и Y для двух планет. Строка 130 – перевод Солнца на новую позицию. Строки 140 и 150 – перевод планет 1 и 2 на новые позиции.

RELOC – оператор, устанавливающий начало координат при работе с графическими операторами

```
RELOC (X,Y)
```

X,Y – координаты начальной точки.

Оператор устанавливает "локальную" систему координат, в которой в дальнейшем будут действовать все графические операторы. Это позволяет выделить области внутри графического экрана, как бы графические "окна", в которых происходят локальные действия, не затрагивающие других частей графического экрана.

Например:

```
10 RELOC (100,100)
20 LINE (0,0)-(200,200),4,BF
```

Линия будет рисоваться в абсолютных координатах (0,0), (200,200), но начальная точка отсчета будет находиться в точке 100,100.

RIBBON – устанавливает код цветности

RIBBON=n

n – выражение от 0 до 7.

Оперотор RIBBON устанавливает код цветности выводимых в текстовом режиме символов в соответствии с таблицей цветов.

ROT – оператор, устанавливающий угол поворота

ROT=X

Оператор ROT устанавливает угол поворота графических изображений по часовой стрелке. Образы должны быть заданы в таблице шаблонов. Значение берется по модулю 64. Нуль – отсутствие угла поворота. При =16; 32; 48 угол поворота составляет 90°; 180°; 270° соответственно. Значение ROT связано со значением оператора SCALE.

При SCALE=1 все четыре значения ROT сохраняются. При SCALE-2 возможны восемь значений ROT – 0, 8, 16, 24, 32, 40, 48, 56.

SCALE – установка масштаба

SCALE=n

n – изменяется от 1 до 255.

Устанавливает масштаб воспроизведения графических изображений, которые должны быть заданы оператором DRAW или XDRAW. При SCALE=1 воспроизводится имеющееся изображение, при SCALE=2 точка увеличивается вдвое и т. д.

SCREEN – оператор установки текстового или графического режима (в форматах 1 – 3) и функция передачи в программу значения символа знакоместа (в форматах 4 и 5)

Формат 1 (только для BASICA).

SCREEN [режим][,[цветность][[,активная страница]

[,отображаемая страница]]]

Формат 2 (только для MSX-BASIC).

SCREEN [,режим][,размер спрайта][,звук клавиши]

[,скорость ленты][,печать]

Формат 3 (только для Бейсик-КОРВЕТ).

SCREEN [N][,M][,R]

Формат 4 (только для BASICA).

Z=SCREEN ((Y,X)[,Z])

Формат 5 (только для Бейсик-Спектрум+2).

Z=SCREEN$ (X,Y)

Формат 1:

режим – целочисленное выражение, равное 0 – 3;

цветность – числовое выражение;

активная страница – целочисленное выражение со значениями 0 – 3 при ширине экранной строки 80 символов и 0 – 7 при ширине экранной строки 40 символов;

отображаемая страница – целочисленное выражение с теми же значениями, что и активная страница.

Параметр **режим** задает режим работы дисплея:

0 — символьный режим;

1 — графический режим среднего разрешения (320 × 200 точек);

2 — графический режим высокого разрешения (640 × 200 точек).

Параметр **цветность** включает или отключает цветное изображение и может принимать два значения: 0 ("нуль") и $\overline{0}$ ("не нуль"). Значение "нуль" в символьном режиме отменяет цветность, а в графическом режиме среднего разрешения означает поддержку цветности. Соответственно значение "не нуль" в графическом режиме означает отмену цветности, а в символьном — включение цветности. В режиме высокого разрешения цветность не имеет значения, изображение всегда черно-белое.

Параметр **активная страница** задает страницу видеопамяти, на которую будет выводиться информация, а параметр **отображаемая страница** — страницу видеопамяти, содержимое которой выводится на экран. Оба параметра допустимы только в символьном режиме.

При выполнении оператора устанавливается новый режим дисплейного процессора, экран чистится и устанавливается передний фон белый, а задний фон — черный.

Если параметр **режим** не указан, изменяется только изображение в соответствии с содержимым отображаемой страницы.

В операторе может отсутствовать любой параметр. Для всех параметров, кроме отображаемой страницы, устанавливаются старые значения. В случае ошибочной установки диапазонов параметров выводится сообщение Illegal function call (Неверный функциональный вызов).

Например:

```
10 SCREEN 0,1,0,0
```

Установлен символьный режим без цветности, активная и отображаемая страницы имеют один и тот же номер 0.

Например:

```
10 SCREEN 1,1
```

Установлен графический режим среднего разрешения; изображение черно-белое.

Например:

```
10 SCREEN 2,,0,0
```

Установлен режим высокого разрешения.

Например:

```
10 SCREEN 1
20 COLOR 2,0
```

Установлен режим среднего разрешения, зеленый фон и палитра 0.

Формат 2. Установка режима графической системы и ряда устройств ввода-вывода:

режим — арифметическое выражение от 0 до 3;

размер спрайта — арифметическое выражение от 0 до 3;

звук клавиши — арифметическое выражение от 0 до 255;

скорость ленты — арифметическое выражение от 0 до 255;

параметр печати — арифметическое выражение от 0 до 255.

Любой параметр может отсутствовать, но вместо него ставится запятая.

Оператор SCREEN устанавливает текущий режим экрана и некоторые важные параметры ввода-вывода. Если какой-либо параметр выходит за допустимые пределы, выводится сообщение Illegal function call (Неверный функциональный вызов), в противном случае устанавливается соответствующий параметр. Если параметр не указан, то он не изменится, принимается старое значение.

Параметр **режим.** Устанавливает режим работы экрана. Если параметр задан, то происходит очистка экрана и производится сброс всех регистров.

Существует четыре режима работы экрана:

1. SCREEN 0 — только текст, бордюр отсутствует, 40 строк по 24 символа, спрайты недопустимы, монохромный режим;

2. SCREEN 1 — действует по умолчанию; 32 строки по 24 символа, спрайты допустимы, группы по восемь символов на экране могут иметь цвет переднего или заднего фона;

3. SCREEN 2 — графический режим; текст выводится на экран в графическом изображении; графический экран размером 256 × 192 графических точек; собственный цвет (один из двух) могут иметь только группы из восьми графических точек; спрайты допустимы; в этом режиме наиболее высокое разрешение.

4. SCREEN 3 — графический режим, текст выводится на экран в графическом изображении, размер графического экрана 64 × 40 графических точек, каждая графическая точка имеет свой собственный цвет (от 0 до 15); спрайты допустимы.

Только при работе экрана в текстовом режиме (SCREEN 0 и SCREEN 1) допустимо применение оператора PRINT и INPUT. Только в графическом режиме (SCREEN 2 или SCREEN 3) экран воспринимает графические команды и операторы; допустим также вывод текста в графическом изображении с помощью специального устройства вывода "GRP:". (Необходимо также иметь в виду, что оба графических экрана и все операции со спрайтами используют одну и ту же координатную сетку 256 × 192.)

Следует отметить, что только SCREEN 0 поддерживает вывод символов размером 6 × 8 графических точек, режимы SCREEN 1 — SCREEN 3 используют размер 8 × 8. Только режим SCREEN 0 не допускает спрайтов, цветных границ или какого-либо другого индивидуального цветового управления (SCREEN 1 имеет управление цветом для группы символов, для чего необходимо использовать оператор VPOKE). Для режимов SCREEN 0 и SCREEN 1 можно использовать оператор WIDTH.

Параметр **размер спрайта.** Режим экрана SCREEN 0 не допускает спрайтов. Для остальных режимов параметр размер спрайта принимает следующие значения:

0 — все спрайты 8 × 8 графических точек, каждый занимает 8 байт данных, допустимо 256 спрайтов;

1 — аналогично 0, за исключением того, что спрайты выводятся на экран вдвое большего размера;

2 — все спрайты размером 16 × 16 графических точек, каждый занимает 32 байта данных, допустимо 64 спрайта;

3 — аналогично 2, за исключением того, что спрайты выводятся вдвое большего размера.

Все спрайты должны быть одного и того же размера по числу байтов и масштабу. Эти значения определяют размеры массива, выбираемого из таблицы, находящейся в видеопамяти при задании псевдопеременной SPRITE$( ).

Параметр **звук клавиши.** Если это значение было установлено в 0, то нажатие клавиш будет происходить без звукового сопровождения; любое другое значение (например, 1) вызовет звуковое сопровождение нажатия клавиш. По умолчанию принимается значение, равное 0.

Параметр **скорость ленты.** Определяет скорость вывода информации (в бодах). Если значение параметра равно 1, то устанавливается наиболее надежная скорость передачи 1200 бод (по умолчанию), если 2, то скорость становится 2400 бод, т. е. в два раза быстрее. Можно изменить скорость вывода командой CSAVE.

Параметр **печати.** Этот параметр устанавливает тип печатающего устройства. Если параметр равен 0, то это печатающее устройство (принтер) совместимо со стандартом MSX и будет распечатывать каждый графический символ из набора MSX. Если параметр не равен 0, то это устройство не совместимо с системой MSX и при печати графические символы преобразуются в пробелы. По умолчанию этот параметр равен 0.

При обращении к оператору SCREEN могут быть заданы все параметры или любая их комбинация; если параметры не указаны (заменены запятыми), старые значения остаются неизменными.

Важное назначение оператора SCREEN – это определение типа и размера спрайта. Можно изменить размер спрайта в любой момент, но это удалит из памяти шаблон спрайта. Точно так же можно использовать SCREEN для выбора режима и это не повлияет на спрайты, но это удалит текущее содержимое экрана.

Надо иметь в виду, что в режимах SCREEN 0,1 начальный цвет экрана устанавливается оператором COLOR, ширина текста – оператором WIDTH, положение курсора устанавливается с помощью POS( ), CSRLIN и LOCATE. Действует оператор PRINT. Графические операторы в этих режимах не действуют.

В режимах SCREEN 2,3 оператор PRINT не действует. Возможно использование всех графических операторов (LINE, DRAW, CIRCLE и т. п.). Цвета экрана устанавливаются как оператором COLOR, так и некоторыми графическими операторами. При возникновении ошибки в операторе SCREEN появляется сообщение Illegal function call (Неверный функциональный вызов). Такая же ошибка появляется при использовании графических операторов в режимах SCREEN 0 и SCREEN 1 или при использовании SPRITE в режиме SCREEN 0.
Данный формат используется в MSX-BASIC. Например:

```
10 COLOR 1,15
20 SCREEN 1
30 FOR N=192 TO 255
40 PRINT CHR$(N);
50 NEXT
```

Высвечиваются на экране символы русских букв.

Формат 3. Переключает графические страницы памяти и устанавливает текстовый режим расширенных символов:

N – номер графической страницы отображения, целочисленное выражение от 0 до 3;

M – номер активной графической страницы, целочисленное выражение от 0 до 3;

R – целочисленное выражение, принимающее значение 0 или 1.

Особенностью ПЭВМ ПК8020/10 является аппаратно реализованная возможность совмещения вывода текста и графики на экран. Поэтому оператор SCREEN Бейсик-КОРВЕТ, реализованный на ПЭВМ ПК8020/10, не нуждается в средствах переключения режимов. Другой особенностью ПК8020/10 является возможность работы с четырьмя графическими страницами, каждая размером 48 Кбайт (три слоя по 16 Кбайт). Для переключения их используются параметры N и M.

Параметр R используется для расширения символов, выводимых на экран (32 символа вместо 64).

В случае, если значения параметров заданы неверно, выводится сообщение НЕВЕРЕН ВЫЗОВ ФУНКЦИИ.

Например:

```
10 SCREEN 0,0
20 COLOR 4,3      ;рисуется и отображается линия в
                   нулевой странице
```

```
30 LINE-(100,200),5
40 SCREEN 0,1
50 CIRCLE(100,200),100 ; рисуется окружность в
                         первой странице
60 SCREEN 1,1    ; отображается первая страница
70 SCREEN ,,1    ; задается режим расширенных
                   символов
```

Формат 4. Функция SCREEN передает в программу код символа, находящегося в указанном знакоместе:

Y – числовое выражение от 1 до 25;

X – числовое выражение от 1 до 40 или от 1 до 80 в зависимости от ширины строки экрана, заданной оператором WIDTH;

Z – числовое выражение, по умолчанию Z=0.

Функция SCREEN действует как в текстовом, так и в графическом режимах. В текстовом режиме имеются два варианта работы функции, в зависимости от значения Z. Если Z=0, то результатом работы функции будет код символа (в ASCII или КОИ-8), находящегося в знакоместе с координатами (X,Y). Если Z<>0, результатом работы функции является атрибут цвета (АЦ) – целое число в диапазоне от 0 до 255. Оно расшифровывается следующим образом:

(АЦ MOD 16) – цвет переднего фона;

((АЦ – цвет переднего фона)/16) MOD 128 – цвет заднего фона;

АЦ > 127 – ИСТИНА (–1), если символ мерцает, и ЛОЖЬ (0), если не мерцает.

В графическом режиме функция SCREEN действует так же, как и в символьном, если в точке (X,Y) находится символ. Если в этой точке находится графическое изображение (точка, линия и т. п.), то в программу передается нуль.

Функция SCREEN имеется во всех версиях языка Бейсик для PC IBM и ЕС 184X.

Например:

```
10 X=SCREEN(10,10)
20 X=SCREEN(1,1,1)
```

Если в знакоместе (10,10) стоит символ A, то X=65. В строке 20 передается атрибут цвета символа, находящегося в верхнем левом углу экрана.

Формат 5. Функция передает значение символа, считанного в заданных координатах:

X – номер строки;

Y – номер столбца.

Функция SCREEN$ предназначена для чтения символа в заданном месте экрана. Функция не может считывать знаки, определенные пользователем, графические символы, а также строки, нарисованные с помощью PLOT,DRAW или CIRCLE. Попытка считать вышеуказанные символы приведет к выводу нуля.

Например:

```
10 PRINT AT 0,0 ; SCREEN$(11,16);
```

Будет считан символ в центре экрана и он же будет выведен в верхнем левом углу экрана.

SCRN – функция, с помощью которой считывается цвет точки (Бейсик-АГАТ)

```
Z=SCRN(X,Y)
```

X,Y – координаты точки в абсолютной форме.

Функция позволяет считать цвет точки и передать номер цвета вызывающему оператору. Номер цвета находится в разрешенном диапазоне. В режиме низкого разрешения координаты

точки должны быть в диапазоне 0...47. Для разных версий Бейсик-АГАТ существуют особенности задания X и Y координат в функции SCRN.

SPRITE$ – позволяет установить и проверить шаблон спрайта (псевдопеременная)

```
SPRITE$(AB)=строка

L$=SPRITE$(AB)
```

AB – арифметическое выражение в диапазоне 0...255 для SCREEN 0 или SCREEN 1 и в диапазоне 0...63 для SCREEN 2 или SCREEN 3;

строка – любое строковое выражение.

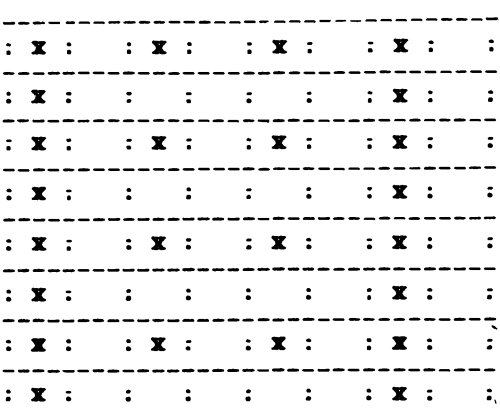
Псевдопеременная SPRITE$ предоставляет доступ к шаблонам спрайтов, находящимся в видеопамяти. Допустимо в этой области видеопамяти 64 32-байтных спрайта или 256 8-байтных. Число и размер спрайтов устанавливается оператором SCREEN (см. SCREEN формат 2). Размер области видеопамяти, отводимой под спрайты составляет 2 Кбайта.

Работает переменная SPRITE$ следующим образом.

Сначала вычисляется строковое выражение из параметра строка и его байты пересылаются в соответствующий значению AB (значение, равное номеру спрайта) образ спрайта. Если строка слишком велика, то она укорачивается, если коротка – дополняется справа байтами нулей до длины спрайта.

Если SPRITE$ (AB) используется в каком-либо выражении, то в программу передается 8 или 32 байта содержимого видеопамяти, соответствующего значению арифметического выражения (номеру спрайта).

Использование SPRITE$ приведем на примере. Представим себе, что мы хотим получить следующую картинку на экране:



Картинка состоит из 8 строк следующего вида: 01000001, 01010101 и т. д. Единицами (X на картинке) обозначены темные пятна, нулями – светлые пятна (см. раздел 6.4). Заметим, что нулевая строка – это двоичный код символа A, вторая строка – символа U и т. д. Например:

```
10 COLOR 1,15
20 SCREEN 1
30 FOR N=1 TO 8
40 READ A$
50 A$="&B"+A$
60 C$=C$+CHR$(VAL(A$))
70 NEXT N
80 SPRITE$(1)=C$ : PRINT SPRITE$(1)
90 DATA 01000001
100 DATA 01010101
110 DATA 01000001
120 DATA 01010101
130 DATA 01000001
140 DATA 01010101
150 DATA 01000001
160 DATA 01010101
```

В примере все картинки записаны в операторах DATA (см. DATA, READ), строки 90 – 160.

В строках 30 – 70 происходит чтение восьми строк спрайта, конкатенация с двоичным признаком – &B и числа преобразуются из строкового выражения в числовой массив спрайта 1 (строка 60). Можно было написать следующее:

```
SPRITE$(1)="AUAUAUAU"
```

Однако такое написание требует развитого образного мышления, поэтому картинка спрайта обычно рисуется на бумаге. Использование массивов, созданных с помощью SPRITE$, см. на примере оператора PUT SPRITE.

SPRITE ON/OFF/STOP – оператор управления процессом прерывания при совмещении ("столкновении") спрайтов

```
     ON
SPRITE  OFF
     STOP
```

Оператор SPRITE ON/OFF/STOP управляет процессом прерывания при совмещении спрайтов. Когда задан SPRITE ON, любое совмещение спрайтов (см. ON SPRITE GOSUB) вызывает прерывание (по окончании выполнения текущего оператора) и затем переход к соответствующей подпрограмме. Автоматически устанавливается SPRITE STOP и, когда происходит возврат из подпрограммы, вновь устанавливается SPRITE ON. Оператор SPRITE STOP временно отключает обработку прерываний. Если "столкновение" происходит, то этот факт фиксируется, но никаких действий не происходит, однако, если позже будет выполняться SPRITE ON, то будет выполняться ранее зафиксированное прерывание.

Оператор SPRITE OFF задает режим игнорирования любого "столкновения" спрайтов, действует до ближайшего ON SPRITE. Определение номеров спрайтов, участвующих в "столкновении", производится программно.

Например:

```
10 ON SPRITE GOSUB 110
20 SCREEN 2,0
30 SPRITE$(0)=STRING$(8,CHR$(255))
40 SPRITE$(1)=STRING$(8,CHR$(255))
50 SPRITE ON
60 FOR I=10 TO 240
```

```
70 PUT SPRITE 0,(I,100),11,0
80 PUT SPRITE 0,(250-I,100),15,0
90 NEXT I
100 GOTO 50
105 REM "Подпрограмма обработки столкновения"
110 SPRITE OFF
120 BEEP
130 RETURN
```

В примере два спрайта: желтый и белый. Когда они сталкиваются, работает подпрограмма (строка 110). Оператор SPRITE OFF позволяет избежать зацикливание подпрограммы, поскольку во время ее выполнения будут продолжаться "столкновения" спрайтов.

VDP – функция, обеспечивающая операции чтения и записи в восемь регистров БИС дисплейного процессора (VDP)

X=VDP(номер регистра)

VDP(номер регистра)=числовое выражение

номер регистра – числовое выражение от 0 до 7;

числовое выражение – числовое выражение от 0 до 255..

Дисплейный процессор ПЭВМ типа MSX имеет 8 регистров только для записи и 1 регистр только для чтения. Доступ к этим регистрам осуществляется с помощью функции VDP.

Регистры 0 – 7 позволяют записывать информацию, например VDP(1)=2 (в регистр 1 заносится число 2). Считать эту информацию непосредственно нельзя. Однако копия этой информации находится в системной области ОЗУ. Регистр 8 можно только прочитать, например PRINT VDP(8). Ошибки при записи в VDP приводят к зависанию компьютера, поэтому нужна особая тщательность при использовании этой функции.

Назначение регистров VDP:

Регистр 0 – два значащих бита 0 и 1:

бит 1 – предназначен для выбора режима экрана и работает с регистром 1 (обозначается M3).

Регистр 1 – семь значащих битов 0, 1, 3, 4, 5, 6, 7:

бит 0 – управление величиной графической точки (1×1 или 2×2);

бит 1 – 0 – спрайт 8×8 графических точек и 1 – спрайт 16×16 графических точек;

бит 3 – обозначается M2, используется для выбора режима экрана;

бит 4 – обозначается M1, в комбинации с M2 и M3 управляет выбором режима экрана:

M1	M2	M3	
0	0	0	– текстовый режим (1)
0	0	1	– высокое разрешение (2)
0	1	0	– многоцветная графика (3)
1	0	0	– текстовый режим (0)

бит 5 – 0 – запрет прерываний VDP и 1 – разрешение прерываний VDP;

бит 6 – 0 – запрет высвечивания экрана и 1 – разрешение высвечивания экрана;

бит 7 – выбор размера видеопамяти.

Регистр 2 – четыре значащих бита (0 – 3). Эти биты определяют старшие 4 бита из 14-битового адреса таблицы имен.

Регистр 3 – восемь значащих битов. Определяет адрес таблицы цветности (старшие 8 бит из 14-битового адреса).

Регистр 4 — три значащих бита (0 — 2). Определяет 3 старших бита (из 14-битового адреса) для таблицы шаблонов, символов или таблицы генератора мультицветности (в зависимости от режима).
Регистр 5 — семь значащих битов (кроме 8). Определяет адрес таблицы атрибутов спрайтов (7 старших битов из 14-битового адреса).
Регистр 6 — три значащих бита (0 — 2). Определяет три старших бита для 14-битового адреса таблицы шаблонов спрайтов.
Регистр 7 — состоит из двух частей. Первая часть (биты 0 — 3) определяет цвет заднего фона, а вторая часть (биты 4 — 7) — переднего фона в текстовом режиме.
Регистр 8:

биты 0 — 4 — содержат номер пятого спрайта на линии;

бит 5 — флаг устанавливается при совмещении спрайтов;

бит 6 — флаг устанавливается в конце растра;

бит 7 — флаг прерывания при совмещении спрайтов или наличии пятого (лишнего) спрайта на линии.

VPEEK — функция для чтения информации из видеопамяти

```
X=VPEEK(адрес)
```

адрес — числовое выражение от 0 до 16383.

Функция позволяет читать содержимое видеопамяти. Содержимое любой ячейки от 0 до 16383 может быть прочитано только с помощью VPEEK, так как эта память недоступна для других инструкций. Обычно VPEEK используется совместно с BASE(0) и VPOKE. Если адрес находится вне области видеопамяти, выдается сообщение Illegal function call (Неверный функциональный вызов).

Например:

```
10 COLOR 1,15
20 SCREEN 0
30 FOR N=1 TO 8
40 PRINT VPEEK(2048+65*8+N-1)
50 NEXT N
```

VPOKE — служит для записи информации в видеопамять

```
VPOKE адрес,данные
```

адрес — числовое выражение от 0 до 16383;

данные — числовое выражение от 0 до 255.

Оператор VPOKE позволяет поместить информацию в любой байт видеопамяти. Если этот байт находится в таблице экрана, то сразу высветится результат. Оператор VPOKE — единственный способ поместить данные в видеопамять. Если параметры выходят за пределы диапазона, то выводится сообщение Illegal function call (Неверный функциональный вызов).

Например:

```
10 SCREEN 0
20 COLOR 1,15
30 FOR N=0 TO 7 STEP 2
40 VPOKE 2048+32*8+N,170
50 VPOKE 2048+32*8+N+1,85
60 NEXT N
```

XDRAW – оператор, позволяющий рисовать произвольные изображения

```
XDRAW арифметическое выражение [AT X,Y]
```

Действие оператора аналогично действию оператора DRAW в формате 2 с той разницей, что изображение воспроизводится цветом, дополнительным к цвету экрана.

6.5.2. Описание звуковых инструкций

BEEP – оператор генерации звукового сигнала

Формат 1 (кроме Бейсик-Спектрум+2).

```
BEEP
```

Формат 2 (только для Бейсик-Спектрум+2).

```
BEEP t,f
```

Формат 1. Оператор вызывает звучание динамика. Как правило, частота звучания 800 Гц, длительность звучания – 0,25 с.

Формат 2. Генерирует звуковой сигнал:

t – длительность (в секундах);

f – высота (в полутонах).

Оператор BEEP вызывает звучание динамика. Высота задается от точки отсчета ноты C (до) первой октавы, высота которой принята за 0. Прочие ноты в зависимости от этого расположены ниже или выше этой ноты и будут иметь соответственно отрицательные или положительные значения. Каждой ноте октавы ставится в соответствие некоторое число с интервалом в полтона между этими числами. Например:

```
10 FOR f=1 TO 8
20 READ note
30 BEEP 0.5,note
40 NEXT f
50 DAT 0,2,4,5,7,9,11,12
```

PLAY – оператор генерации музыкальных фраз

Формат 1 (кроме Бейсик-Спектрум+2).

```
PLAY команда
```

Формат 2 (только для Бейсик-Спектрум+2).

```
PLAY строка
```

Формат 1:

команда – строковое выражение, состоящее из команд музыкального макроязыка.

Оператор PLAY позволяет создавать музыкальные фразы, записанные в строковом выражении. Как и в операторе DRAW, здесь использован "язык в языке" – музыкальный макроязык. Использование этого языка позволяет создавать музыкальные фразы на ПЭВМ, держа перед глазами общепринятую нотную запись.

Программа на музыкальном макроязыке состоит из строки длиной 255 символов. Каждая команда представлена одним или двумя символами. Команды пишутся подряд в Бейсик-КОРВЕТ и Бейсик-Спектрум+2, в других версиях они могут быть разделены пробелами. Ниже приведены команды музыкального макроязыка и их содержание:

A–G – звучит заданная нота (C – до, D – ре, E – ми, F – фа, G – соль, A – ля, B – си);

или **+** – после ноты обозначает диез;

– – после ноты обозначает бемоль;

Ln – устанавливает длительность каждой ноты: L1 – целая нота, L2 – половина, L4 – четверть и т. д.; т. е. действительная длительность ноты – это 1/n, где n изменяется от

1 до 64; длительность может быть указана за нотой, если необходимо изменить длительность только одной ноты, например, запись A16 эквивалентна L16A;

MF — музыкальный передний фон; музыка (запрограммированная оператором PLAY или SOUND) будет звучать на переднем фоне, т. е. каждая последовательная нота или звук не зазвучит, пока не закончится предыдущая нота или звук; музыкальный передний фон — состояние по умолчанию;

MB — фоновая музыка; музыка будет звучать на заднем фоне, т. е. значение каждой ноты или звука будут размещены в буфере, позволяющем программе продолжить выполнение, пока музыка звучит на заднем фоне; одновременно может звучать до 32 нот или пауз;

MN — обычный режим воспроизведения звука, каждая нота звучит 7/8 времени, заданного длительностью L;

ML — легато (legato), каждая нота звучит полный период, установленный длительностью L;

MS — стаккато (staccato), каждая нота звучит 3/4 времени, заданного длительностью L;

Nn — звучит нота n; аргумент n изменяется от 0 до 84; в семи возможных октавах существуют 84 ноты; n=0 означает паузу;

On — октава; Установить текущую октаву; существует 7 октав: от 0 до 6; каждая октава от ноты C до ноты B;

Pn — пауза; аргумент n изменяется от 1 до 64 и определяет длительность паузы тем же способом, что и L;

Tn — темп (tempo); устанавливает число четвертей ноты в секунду; аргумент n изменяется от 32 до 255, по умолчанию 120;

Xстрока — выполняет заданную строку;

. — точка, указанная после ноты, заставляет ноту звучать как удлиненную, т. е. ее длительность умножается на 3/2.

После ноты можно указать больше одной точки и длительность будет увеличиваться соответственно. Например, "A..." будет звучать 9/4 заданной длительности L. Точки также можно указывать после паузы P. Во всех этих командах аргумент n может быть константой, например "12", или переменной, записанной как "=переменная", где "переменная" — имя переменной. Для звучания связанных нот можно соединять несколько записей.

Можно использовать команду X, чтобы запомнить мелодию в единой строке и вызывать ее повторно с различными темпами или октавами из другой строки.

Например:

```
10 A$="BB-C"
20 B$="O4XA$;"
30 C$="L1CT50N3N4N5N6"
40 RLAY "P2XA$;XB$;XC$;"
```

В MSX-BASIC команды музыкального макроязыка несколько отличаются от ранее описанных. В частности,

Rn — пауза (то же, что и Pn), по умолчанию n=4;

Vn — громкость (volume); аргумент n изменяется от 0 до 15, по умолчанию 8;

Mn — модуляция, устанавливает период огибающей; аргумент n изменяется от 1 до 65535, по умолчанию 255;

Sn — форма огибающей; аргумент n изменяется от 1 до 15, по умолчанию 1 (формы огибающей, установленные этой командой, приведены на рис. 6.4).

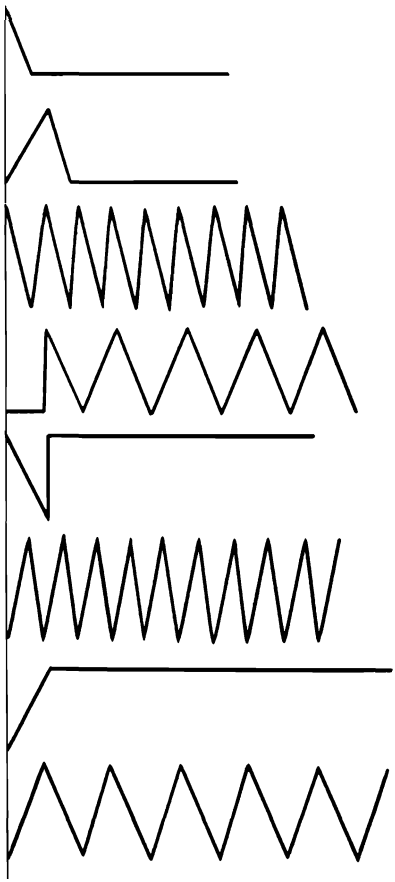


Рис. 6.4

Пример смотри в гл. 7, программа "Танец маленьких лебедей".

Формат 2. Генерирует музыкальные фразы:

строка – строковое выражение, состоящее из параметров.

В Бейсик-Спектрум+2 оператор PLAY использует следующие параметры, задаваемые в строке:

a–g(A–G) – высота ноты в пределах текущей октавы;

$ – после параметра знак строчной переменной перед нотой как бемоль;

– перед нотой как диез;

O – номер используемой октавы, определяемой номером октавы (0–8);

1–12 – длительность используемых нот;

& – пауза;

_ – (подчеркивание) знак легато;

N – разделяет два числа;

V – уровень звука, определяемый числом 0–15;

W – задание формы огибающей;

U – задание шума;

X – длительность шума, определяемая числом 0–65535;

T – темп музыки, определяемый числом 60–240;

() – (скобки) указывает, что включенная в скобки фраза должна повторяться;

!! – комментарии в строке;

H – останов оператора PLAY:

M – номер канала интерфейса MIDI, определяемый числом 1–63

Y – включение каналов MIDI, затем следует код 1–16

Z – задание кода программирования MIDI, затем следует номер кода;

, – (запятая) разделитель голосов.

Например:

оператор PLAY

```
10 LET t$="T120"
20 LET a$=t$+"o4cCcCgGgG"
30 LET b$="o6CaCc$bd$bD"
40 PLAY a$,b4
```

PLAY – функция для опроса состояния звукового канала

Y=PLAY (n)

n – числовое выражение от 0 до 3.

Образует информацию о состоянии звукового канала. Поскольку звуковой канал может состоять из трех голосов, n может изменяться от 0 до 3. Если n=0, то сообщается, есть ли еще в

буфере информация о несыгранных звуках. Если n=1, 2 или 3, то образуется –1, если очередь еще в обработке, т. е. звучит соответствующий голос, иначе – 0.

SOUND – оператор генерации звука

Формат 1 (только для BASICA).

SOUND частота, продолжительность

Формат 2 (только для Бейсик-КОРВЕТ и MSX-BASIC).

SOUND регистр, значение

Формат 1:

частота - частота в герцах; числовое выражение от 37 до 32767;

продолжительность – числовое выражение от 0 до 65535, задается аппаратно во внутренних квантах времени (1000/18,2 мс).

Оператор позволяет генерировать любой звук, однако желательно использовать темперированную гамму. Когда с помощью оператора SOUND генерируется звук, выполнение программы продолжается до тех пор, пока не появится другой оператор SOUND или END. Если продолжительность нового оператора SOUND равна 0, то текущий оператор SOUND выключается. Иначе программа ожидает до тех пор, пока не закончится действие старого оператора SOUND. Информация о вызываемых звуках будет находиться в специальном буфере, при этом выполнение старого оператора SOUND не останавливается при встрече нового оператора SOUND. Если оператор SOUND не выполняется, то новый SOUND X, 0 не действует.

Для создания периодов пауз рекомендуется использовать оператор **SOUND 32767, продолжительность.**

Значения частот, соответствующих принятой нотной записи, приведены в табл. 6.3.

Таблица 6.3

Нота	Частота	Нота	Частота
C(до)	130.810	C*(до)_	523.250
D(ре)	146.830	D(ре)	587.330
E(ми)	164.810	E(ми)	659.260
F(фа)	174.610	F(фа)	698.460
G(соль)	196.000	G(соль)	783.990
A(ля)	220.000	A(ля)	880.000
B(си)	246.940	B(си)	987.770
C(до)	261.630	C(до)	1046.500
D(ре)	293.660	D(ре)	1174.700
E(ми)	329.630	E(ми)	1318.500
F(фа)	349.230	F(фа)	1396.900

Окончание табл. 6.3

Нота	Частота	Нота	Частота
G(соль)	392.000	G(соль)	1568.000
A(ля)	440.000	A(ля)	1760.000
B(си)	493.880	B(си)	1975.500

Например:

```
10 FOR I=440 TO 1000 STEP 5
20 SOUND I,0.5
30 NEXT
40 FOR I=1000 TO 440 STEP -5
50 SOUND I,0.5
60 NEXT
```

Формат 2:

регистр – арифметическое выражение от 0 до 10, определяющее номер регистра;

значение – арифметическое выражение, задающее значение регистра.

Команда позволяет пользователю добиться различных звуковых эффектов управлением программируемым синтезатором звука. Управление осуществляется заданием значений 11 регистров (для ПЭВМ типа MSX – 14 регистров).

В табл. 6.4. описывается назначение этих регистров.

Таблица 6.4

Номер регистра	Назначение регистра в Бейсик-КОРВЕТ	в MSX-BASIC
0	Задает коэффициент деления звука для первого голоса	Младшие 8 бит частоты первого голоса
1	То же для второго голоса	Старшие 4 бита частоты первого голоса
2	То же для третьего голоса	Младшие 8 бит частоты второго голоса
3	То же для четвертого голоса	Старшие 4 бита частоты второго голоса
4	Задает коэффициент деления для шума	Младшие 8 бит частоты третьего голоса

Окончание табл. 6.4

Номер регистра	Назначение регистра в Бейсик-КОРВЕТ	Назначение регистра в MSX-BASIC
5	Управляет звуком или шумом по голосам	Старшие 4 бита частоты третьего голоса
6	Задает громкость, форму огибающей для первого голоса	5 битов для задания шума
7	То же для второго голоса	Управляющий регистр смешивания. Возможность управления тоном и шумом
8	То же для третьего голоса	Управление амплитудой и режимом канала первого голоса
9	То же для четвертого голоса	То же для второго голоса
10	Задает период огибающей	То же для третьего голоса
11	-	Младшие 8 бит управления длительностью шума
12	-	Старшие 8 бит управления длительностью шума
13	-	Задает форму огибающей

Коэффициент деления определяется по следующей формуле: $N_{зв}=f_{вх}/f_{зв}$, где $f_{вх}$ – входная частота задается аппаратно; $f_{зв}$ – частота звука, которую необходимо получить на выходе синтезатора звука.

При задании огибающей формула приобретает вид: $N_{ог}=f_{вх}/f_{ог}\cdot 256$. Формы огибающих, которые можно получить, приведены на рис. 6.4.

Оператор SOUND позволяет воспроизводить звуки, дополняющие музыкальные фразы, воспроизводимые с помощью оператора PLAY.

Глава 7

Примеры построения программ на языке Бейсик

Наибольшее распространение получили программы для решения математических и инженерных задач, для обучения, игровые программы и программы для решения бытовых задач.

Программы для решения математических задач выполняют такие функции, как преобразование чисел, различные действия над числами, тригонометрические вычисления, формирование последовательности случайных чисел, задачи математической логики, определение площадей фигур и т. п.

Постановка инженерных задач производится в прикладной форме (расчет конструкций, линейных фильтров, экономические расчеты и т. п.). Они более сложны и сводятся к различным математическим описаниям:

системам линейных и нелинейных алгебраических уравнений;

трансцендентным уравнениям;

системам обыкновенных дифференциальных уравнений;

системам уравнений и уравнениям с частными производными.

Часто эти задачи ставятся как задачи отыскания оптимальных вариантов, как задачи исследования объектов и обработки экспериментальной информации. Методы решения задач этого плана подробно описаны в обширной литературе по методам цифровых и аналоговых вычислений. Применение языка Бейсик к решению инженерных задач в силу простоты овладения языком расширяет круг пользователей.

Обучающие программы классифицируются по уровню учебных заведений, для которых они предназначены; особое место занимают программы для самообучения. Для каждого уровня учебного заведения программы классифицируются по предметному признаку (химия, физика, программирование и т. п.). Особый интерес у школьников приобретают программы проверки знаний, содержащие элемент самообучения.

В последние годы ПЭВМ значительно расширили свои возможности в части представления графических образов, цвета и звука. Это, как уже было сказано в гл. 6, нашло свое естественное отражение в составе операторов языка Бейсик. Применение графики и цвета в обучающих, математических и инженерных задачах резко увеличивает иллюстративные возможности ПЭВМ и упрощает восприятие выходной информации, а также организацию диалога с пользователем. Особое значение графика, цвет и звук приобретают в игровых программах.

Игровые программы предназначены для решения двух основных задач — разумного использования свободного времени и приобретения навыков к работе с ПЭВМ. Все игровые программы можно условно разделить на статические игровые программы и динамические игровые программы.

К статическим программам относятся программы, предполагающие деление игры на последовательные этапы, каждый из которых делится на два цикла. В первом цикле игрок делает "ход"; во втором цикле "ход" делает машина, а игрок ждет ответа машины.

В динамических играх игрок и машина действуют "параллельно" — любое действие игрока или машины вызывает немедленную реакцию машины или игрока соответственно.

Бытовые программы предлагают пользователям возможность записи и поиска адресов и телефонов (электронная картотека), построения памяток дел, которые нужно выполнить в ближайшие дни (электронный секретарь), ведения и контроля расходов в семье и ряд других возможностей, использование которых может облегчить быт семьи.

Ниже приведены примеры, демонстрирующие работу различных операторов языка Бейсик для математических расчетов, построения графических картин, создания музыкальных программ.

"Арифметика" – программа выполнения четырех арифметических действий

```
100 INPUT "Количество цифр в числе ";D
110 D=INT(D) : IF D<=0 OR D>=3 THEN 100
120 DEF FNR(X)=INT(RND(1)*10^X)
130 OP=INT(RND(1)*4)+1
140 ON OP GOSUB 190,220,260,290
150 PRINT "Чему равно ";N1;OP$;N2;
160 INPUT A : PRINT
170 IF A<>EX THEN PRINT "Ответ неверен.
    Попытайтесь еще раз." : GOTO 150
180 PRINT "Правильно."
190 OP$="+"
200 N1=FNR(D) : N2=FNR(D)
210 EX=N1+N2 : RETURN
220 OP$="-"
230 N1=FNR(D) : NN=FNR(D) : IF NN<=N1 THEN N2=NN :
    GOTO 250
240 N2=N1 : N1=NN
250 EX=N1-N2 : RETURN
260 OP$="*"
270 N1=FNR(D) : N2=FNR(D)
280 EX=N1*N2 : RETURN
290 OP$="/"
300 N2=FNR(D) : IF N2=0 THEN 300
310 N1=FNR(D)
320 EX=N1/N2 : RETURN
```

Строки 100–180 являются основной программой, а строки 190–210, 220–250, 260–280, 290–320 содержат четыре подпрограммы.

В строках 100–120 переменной D присваивается количество цифр в числах для этой задачи и определяется функция FNR, которая задает числа, используемые в этой задаче.

В строках 130–140 выбирается одна из четырех арифметических операций (сложение, вычитание, умножение и деление) и вызывается нужная подпрограмма.

В строках 150–180 формируется ответ.

"Перевод числа" – программа перевода десятичного числа в число с основанием больше десяти

```
 50 DIM A(15)
 90 B$="0123456789ABCDEFHIJKLMN"
100 PRINT "Новое основание";
110 INPUT B
120 PRINT "Первое и последнее число, которые надо
      перевести, ";
130 INPUT F,L
140 FOR I=F TO L
150 PRINT
160 GOSUB 1480
170 REM Печать результирующей таблицы
```

```
180 PRINT I;TAB(6);
190 FOR D=J TO 1 STEP-1
200 PRINT MID$(B$,A(D)+1,1);
210 NEXT D
220 NEXT I
230 END
1480 REM Программа перевода строки
1500 I1=I
1510 J=1
1520 Q=INT(I1/B)
1530 R=I1-Q*B
1535 I1=Q
1540 A(J)=R
1545 J=J+1
1550 IF Q>=B THEN GOTO 1520
1560 A(J)=Q
1570 RETURN
```

"Шахматные фигуры" – программа на языке Бейсик-Спектрум+2 вводит графические символы, определенные пользователем для следующих клавиш, которые должны изображать шахматные фигуры: B – для слона (bishop); K – для короля (king); R – для ладьи (rook); Q – для ферзя (gueen); P – для пешки (pawn); N – для коня (knight).

```
5 b=BIN01111100 : c=BIN00111000 : d=BIN00010000
10 FOR n=1 TO 6 : READ p$ : REM 6 фигур
20 FOR f=0 TO 7 : REM Считать фигуры в 8 байт
30 READ a : POKE USR p$+f,a
40 NEXT f
50 NEXT n
55 REM Шахматные фигуры
60 REM Слон
70 DATA"b",0,d,BIN00101000,BIN01000100,
      BIN01101100,c,b,0
80 REM Король
90 DATA "k",0,d,c,d,c,BIN01000100,c,0
100 REM Ладья
110 DATA "r",0,BIN01010100,b,c,c,b,b,0
120 REM Ферзь
130 DATA "q",0,BIN01010100,BIN00101000,
       BIN011011,BIN00101000,b,b,0
140 REM Пешка
150 DATA "p",0,0,d,c,c,d,b,0
160 REM Конь
170 DATA "n",0,d,c,BIN01111000,BIN00011000,
       c,b,0
```

```
B - СЛОН      K - КОРОЛЬ     R - ЛАДЬЯ

   1             1           1 1 1
  1 1           111          11111
 1   1           1            111
11 11          111            1
 111          1   1           11111
11111          111           11111
```

```
Q - ФЕРЗЬ     P - ПЕШКА     N - КОНЬ

 1 1 1                          1
  1 1            1            111
 11 11          111          1111
  1 1           111            11
 11111            1           111
 11111         11111         11111
```

Каждый графический символ, определенный пользователем, размещается в сетке из 8×8 точек, каждая из которых может быть включена или выключена. Графический символ имеет свой образ, который хранится в виде восьми чисел, каждое из которых соответствует одной строке сетки. Числа в программе можно записать в виде ключевого слова BIN, за которым будут следовать 8 нулей или единиц (0 — для "бумаги"; 1 — для "чернил").

Эти восемь чисел запоминаются в восьми байтах. Адрес первого байта определяет USR"X" (где X — символ, при нажатии которого будет появляться на экране графический символ, определенный пользователем). Адрес второго байта USR"X" + 1 и т. д. до 8-го байта, адрес которого USR"X"+7. Строковый аргумент функции USR должен быть единственным символом.

"Пятнашки" — игровая программа

```
100 DIM A(16),B(16)
110 CLS : PCLS : REM БЕЙСИК-КОРВЕТ
115 REM НАРИСОВАТЬ И ЗАКРАСИТЬ ПРЯМОУГОЛЬНИК
120 LINE (152,48)-(362,190),1,B
130 PAINT (153,49),4,1
135 REM РАЗДЕЛИТЬ ПРЯМОУГОЛЬНИК НА 16 РАВНЫХ
      ЧАСТЕЙ
140 FOR YI=59 TO 180 STEP 32
150 FOR XI=160 TO 340 STEP 48
160 LINE (XI,YI)-(XI+42,YI+26),1,B
170 NEXT XI : NEXT YI
180 FOR I=1 TO 16
190 B(I)=I : NEXT I
200 FOR J=1 TO 16
210 K=INT(RND(1)*16+1)
220 IF B(K)=0 THEN 210
230 A(J)=K : B(K)=0 : NEXT J
240 I=1
245 REM РАССТАНОВКА ЦИФР В КВАДРАТАХ
250 FOR YI=5 TO 12 STEP 2
260 FOR XI=22 TO 44 STEP 6
270 LOCATE XI,YI
280 IF A(I)<>16 THEN PRINT A(I) : GOTO 300
290 PRINT "" : X=XI : Y=YI : II=I :
    PAINT (X*8+1,Y*16+1),5,1
300 I=I+1
310 NEXT XI : NEXT YI
320 K$=INKEY$
330 DX=0 : DY=0
335 REM ВНИЗ
340 IF K$="N" AND Y>6 THEN DY=-2 : J=-4 :
    GOTO 390
```

```
345 REM ВВЕРХ
350 IF K$="W" AND Y<11 THEN DY=2 : J=4 :
    GOTO 390
355 REM ВЛЕВО
360 IF K$="L" AND X<38 THEN DX=6 : J=1 :
    GOTO 390
365 REM ВПРАВО
370 IF K$="P" AND X>22 THEN DX=-6 : J=-1 :
    GOTO 390
380 GOTO 320
385 REM СДВИГ МАЛЕНЬКИХ КВАДРАТОВ
390 LOCATE X,Y : PAINT(X*8+1,Y*16+1),4,1
400 PRINT A(II+J) : A(II)=A(II+J)
410 II=II+J : A(II)=0
420 X=X+DX : Y=Y+DY
430 LOCATE X,Y : PRINT "       " :
    PAINT(X*8+1,Y*16+1),5,1
440 FOR I=1 TO 15
450 IF A(I)<>I THEN 320
460 NEXT I
470 END
```

"Узор" – программа построения цветного узора с помощью датчика случайных чисел

```
10 REM Цветной узор
20 CLS
30 PRINT "Введите максимальные вертикальные и
          горизонтальные "
40 PRINT "значения для этого режима разрешения"
50 INPUT XM,YM
60 XC=XM/2
70 YC=YM/2
80 CLS
90 REM Установка графического режима
100 FOR X=0 TO XM
110 C=INT((RND(1)*8)) : REM Номер цвета случаен
120 COLOR C,0
130 LINE (XC,YC)-(X,0)
140 NEXT X
150 FOR Y=0 TO YM
160 C=INT((RND(1)*8)) : REM Номер цвета случаен
170 COLOR C,0
180 LINE (XC,YC)-(XM,Y)
190 NEXT Y
200 FOR X=XM TO 0 STEP -1
210 C=INT((RND(1)*8)) : REM Номер цвета случаен
220 COLOR C,0
230 LINE (XC,YC)-(X,YM)
240 NEXT X
250 FOR Y=YM TO 0 STEP -1
260 C=INT((RND(1)*8)) : REM Номер цвета случаен
270 COLOR C,0
280 LINE (XC,YC)-(0,Y)
290 NEXT Y
300 END
```

"Построение подобных треугольников" – программа предназначена для обучения школьников визуальной оценке подобия треугольников

```
10 PRINT "Укажите координаты первого угла
          основного треугольника"
20 INPUT "X0=";X0
30 INPUT "Y0=";Y0
40 PRINT "Укажите длину стороны основного
          треугольника"
50 INPUT "L1=";L1
60 PRINT "Укажите значение прилежащих углов
          (в градусах)"
70 INPUT "F1=";F1 : INPUT "F2=";F2
80 F1=F1*3.141593/180 : F2=F2*3.141593/180
90 PRINT "Для построения основного треугольника";
100 PRINT " нажмите одну из клавиш из ряда 1-6"
110 INPUT A
120 IF A=1 OR A=2 OR A=3 OR A=4 OR A=5 OR A=6 THEN
    GOSUB 1000
130 PRINT "Хотите ли вы изменить параметры
          основного треугольника?"
140 PRINT "Если да - нажмите клавишу 1; если нет -
          клавишу 0"
150 INPUT B
160 IF B=1 THEN GOTO 10
170 PRINT "Задайте коэффициент подобия
          треугольников (от .1 до 2)"
180 PRINT "Задайте координаты первого угла
          подобного треугольника"
190 INPUT "K=";K : INPUT "X3=";X3 : INPUT "Y3=";Y3
200 L2=K*L1
210 PRINT "Для построения подобного треугольника";
220 PRINT " нажмите одну из клавиш из ряда 1-6"
230 INPUT A
240 IF A=1 OR A=2 OR A=3 OR A=4 OR A=5 OR A=6 THEN
    GOSUB 1100
250 PRINT "Хотите вы изменить параметры подобного
          треугольника?"
260 PRINT "Если да - нажмите клавишу 1; если нет -
клавишу 0"
270 INPUT B
280 IF B=1 THEN GOTO 180
290 IF B=0 THEN PRINT "Хотите вы изменить
                параметры основного треугольника?";
300 PRINT "Если да - нажмите кнопку 1; если нет -
          0"
310 INPUT B
320 IF B=1 THEN GOTO 10 ELSE END
1000 X1=X0+L1*COS(F1) : Y1=Y0-L1*SIN(F1)
1010 X2=X1-ABS(Y0-Y1)*1/TAN(F1+F2)
1020 LINE(X0,Y0)-(X1,Y1),A
1030 LINE-(X2,Y0),A
1040 LINE-(X0,Y0),A
```

```
1050 RETURN
1100 X4=X3+L2*COS(F1) : Y4=Y3-L2*SIN(F1)
1110 X5=X4-ABS(Y3-Y4)*1/TAN(F1+F2)
1120 LINE (X3,Y3)-(X4,Y4),A
1130 LINE-(X5,Y3),A
1140 LINE-(X3,Y3),A
1150 RETURN
```

"Мяч" – программа, демонстрирующая графические возможности интерпретатора BASICA для PC IBM

```
10 REM Программа BALL
20 PLAY "mf132t067"
30 DEFINT A-Z : DIM B(280) : DIM A(150)
40 CLS : SCREEN 1,0 : COLOR 8,0 : KEY OFF
50 CIRCLE (160,100),10,2 : REM Рисование мяча
60 PAINT (160,100),2,2
70 GET (150,90)-(170,110),A : REM Запомнить
   рисунок мяча в массиве A
80 CLS : LINE (19,0)-(299,177),,B : REM Рамка
90 LINE (20,1)-(300,178),,B
100 LOCATE 24,11 : PRINT "Для выхода нажмите ESC";
110 FOR X=20 TO 280 STEP 4
115 REM Затухающая синусоида
120 B(X)=159-CINT(ABS(SIN(X*.0785398)*X)/2)
130 NEXT : L=7 : Y2=150
140 LINE (223,3)-(253,175),3,BF
150 LINE (191,3)-(222,175),2,BF
160 LINE (160,3)-(190,175),1,BF
170 FOR D=0 TO 1
180 S=20+D*260 : E=280-D-260
190 FOR X=S TO E STEP 4-8*D
200 Y=B(X) : N=(170-Y2)/5 : PLAY "L=L;T255;N=N;"
210 IF F THEN PUT (X2,Y2),A : REM Рисунок на экран
220 PUT (X,Y),A : X2=X : Y2=Y : F=-1 : NEXT
230 L=(L+7) MOD 64 : NEXT
```

"Танец маленьких лебедей" – программа демонстрирует работу оператора PLAY

```
10 PRINT "Танец маленьких лебедей"
20 A$="R1R8A8A8A8A8.G#32A32B8A8G#8B8B8B8B8.A32B32
      O5C#8O4B8A8O5C#8F#8E#8C#8O4G#8O5C#16O4B16A1
      6G#16A8O5C#8F#8E#8C#8O4G#8O5C#16O4B16A16G#1
      6"
30 B$="R1R8F#8F#8F#8F#8.R16R8F#8E#8R8G#8G#8G#8.R16
      R8G#8F#8R4G#8E#8RGR4R4R8G#8E#8R4R8"
40 C$="R1RRRR"
50 D$="O2F#8O3C#8O2F#8O3C#8O2F#8O3C#8O2F#8O3C#8
      O2F#8O3C#8O2F#8O3C#8O2F#8O3C#8O2F#8O3C#8
      O2F#8O3C#8O2F#8O3C#8O2F#8O3C#8O2F#8O3C#8
      O2F#8O3C#8O2F#8O3C#8O2F#8O3C#8O2F#8O3C#8
      O2F#8O3C#8O2F#8O3C#8O2F#8O3C#8O2F#8R8
```

```
60 E$="F#8O5C#8C#8C#8C#8C#8F#16E16D16C#16O4B8O5
       C#4D4O4A#B8B8B8B8B8B8B8O5E16D16C#16O4B16
       A8F#8G#16A16B16B#16O5C#8O4C#8D16D#16E16E#
       16F#8A8A8A8A8.G#32A32B8A8G#8B8B8B8.A32B32
       O5C#8O4B8"
70 F$="R1RRRR8F#8F#8F#8F#8.R18R8F#8"
80 G$="R4R8O4E8R8E8R8E8R4G-8R8F#8R8E8R8R4R8D8R8D8
       R8DBR1R"
90 H$="O3F#8O4C#8O3B8O4C#8O3A#8O4C#8O3F#8A#8B8R8O4
       E8R8D8R8C#8R8O3B8O4D8O3A8B8G#8B8E8G#8F#8A8G
       #8F#8E#8C#8O2B8G#8O2F#8O3C#8O2F#O3C#O2F#8O3
       C#8"
100 I$="A8O5C#8F#8E#8C#8O4G#8O5C#16O4B16A16G#16A8O
       5C#8F#8E#8C#8O4G#8O5C#16O4B16A16G#16F#8O5F
       #8E#8C#8F#8F#8E#8C#8F#8C#4O4B4A4G#8F#8O5E
       #8F#4R2"
110 J$="E#8R8G#8G#8G#8.R16R8G#8F#8R4G#8E#8R8R4R4R8
       G#8E#8R4R8R8R8G#8R8A8R8G#8R8A8A4G#4F#4E#8R
       8C8A4R2"
120 K$="R1RRR4O3C2C4"
130 L$="O2F#8O3C#8O2F#8O3C#8O2F#8O3C#8O2F#8O3C#8
       O2F#8O3C#8O2F#8O3C#8O2F#8O3C#8O2F#8O3C#8
       O2F#8O3C#8O2F#8O3C#8O2F#8O3C#8O2F#8O3C#8
       O2F#8R8F#2F#4F#4F#2F4R8O3C#8O2F#4R2"
140 PLAY "V1504L4T120XA$;XE$;XI$;","V15 T120O4L
         4XB$;XF$;XJ$;","T120O4L4V15XD$;XH$;XL$
         ;","T120O4L4V15XC$;XG$;XK$;"
```

"Шум прибоя" – программа, демонстрирующая работу оператора SOUND

```
10 PRINT "Шум морского прибоя" _
20 FOR I=0 TO 10 : SOUND I,O : NEXT
30 SOUND 5,239
40 SOUND 6,30
50 SOUND 10,15000
60 FOR I=1 TO 250
70 SOUND 4,100
80 NEXT I
90 FOR I=1 TO 20000
100 NEXT I
110 SOUND 5,255
120 END
```

"Камень" – человек бросает камень под углом V0 к горизонту с некоторой скоростью V0. Нужно подобрать угол V0 и скорость так, чтобы камень попал в заданную область.

На экране задается поверхность земли и отмечается область, куда следует попасть. Игрок по запросу машины должен выбрать нужную скорость и угол бросания. На экране изображается траектория полета камня. Если камень попал в заданную область, на экране выводится поздравление с успехом, а если нет – игрок задает новые значения скорости и угла бросания. Число попыток до успеха фиксируется.

```
10 A=INT(100*RND(5))+200
20 B=INT(20*RND(5))+10
30 XA=249 : YA=250
40 FOR I=1 TO (A+B+50)
50 PSET(I,YA),5
60 NEXT
70 FOR I=A TO (A+B)
80 PSET(I,XA),5
90 NEXT
100 L=1
110 PRINT "Введите нужную скорость (от 10 до 50
          м/сек)"
120 INPUT V0
130 PRINT "Введите угол бросния (от 5 до 60)"
140 INPUT Y0 : YP=Y0*3,14/180
150 VX=V0COSYP - VY=V0SINYP
160 X=0 : Y=250
170 FOR I=1 TO 800
180 VX=VX-0.05
190 VY=VY-0.1
200 X=X+0.01+VX
210 Y=Y-0.01*VY
215 PLAY "FD+FGA+A+BA+","D+DDGO2GAAA",
         "FFFCCCCC"
220 IF X>500 THEN 260
230 IF VY<0 AND Y>250 THEN 260
240 PSET(X,Y),5
250 NEXT
260 IF A<X AND A+B>X THEN PRINT "Молодец" :
    GOTO 280 ELSE PRINT "Повторите"
270 L=L+1 : GOTO 110
280 PRINT "Число попыток"; L
290 END
```

"Лобовой удар" — с помощью этой программы можно определить потерю энергии и скорость двух тел массой M1 и M2 после соударения

```
10 PRINT "Введите массы соударяющихся тел"
20 INPUT "M1=";M1 : INPUT "M2=";M2
30 PRINT "Введите скорости тел до соударения"
40 INPUT "V1=";V1 : INPUT "V2=";V2
50 PRINT "Введите коэффициент восстановления"
60 INPUT K
70 E=M1*V1+M2*V2:M=M1+M2:V0=(V1-V2)*K
80 V01=(E-V0*M1)/M:V02=(E+V0*M2)/M
90 PRINT "Скорость первого тела после
          соударения";V01
100 PRINT "Скорость второго тела после
           соударения";V02
110 W=M1*M2*(V1-V2)^2*(1-K^2)/2M
120 PRINT "Потеря энергии после удара";W
130 END
```

"Простые числа" – программа находит заданное количество простых чисел 1, 2, 3 и т. д. Так как все последующие простые числа нечетные, то в программе проводится определение следующего простого числа путем деления его на все предыдущие простые числа

```
10 PRINT "Сколько простых чисел Вы хотите найти?"
20 INPUT N
30 IF N<=3 THEN M=3 ELSE M=N
40 DIM P(M)
50 P(1)=1 : P(2)=2 : P(3)=3
60 IF N<=3 THEN 150
70 I=3 : REM I - количество уже найденных простых
                 чисел
80 K=3 : REM K - значение очередного проверяемого
                 числа
90 K=K+2
100 FOR J=2 TO I
110 IF INT(K/P(J))=K/P(J) THEN 90
120 NEXT J
130 I=I+1 : P(I)=K
140 IF I<N THEN 90
150 FOR J=1 TO N
160 PRINT P(J);
170 NEXT J
180 END
```

"Сложные проценты" – необходимость в определении сложных процентов возникает во всех тех случаях, когда сумма, на которую начисляются проценты, является результатом предыдущих расчетов с учетом процентных начислений. Примером таких расчетов является расчет стоимости оборудования с учетом амортизации или определение вклада в сберкассе

```
10 PRINT "Введите начальное значение суммы"
20 INPUT S
30 PRINT "Введите процент (со знаком)"
40 INPUT P
50 PRINT "Введите число параметров"
60 INPUT N
70 W=(1+P/100)^N
80 PRINT "Конечное значение суммы";W
90 END
```

"Таблица умножения" – достаточно простая программа для проверки знания таблицы умножения у школьников младших классов. Для ответа на каждый вопрос дается три попытки. В первой части программ (строки 10 – 60) формируется таблица умиожения, во второй части (строки 90 – 130) выдается запрос школьнику и контролируется его ответ, а в третьей части производится подсчет правильных и неправильных ответов (K – число ошибок, L – число правильных ответов, N – число попыток для каждого вопроса) и содержатся строки для продолжения проверки знаний.

```
10 DIM T(9,9)
20 FOR I=1 TO 9
30 FOR J=1 TO 9
40 T(I,J)=I*J
50 NEXT J
```

```
60 NEXT I
70 K=0:L=0
80 F=0
90 I=INT(9*RND(9))
100 J=INT(9*RND(9))
110 PRINT "Чему равно произведение";I;"X";J
120 INPUT P
130 IF P=T(I,J) THEN 250
140 PRINT "Ответ неверен"
150 F=F+1
160 IF F<3 THEN 110
170 K=K+F
180 PRINT "Будете продолжать?"
190 PRINT "Если да, нажмите клавишу 1, если нет -
      0"
200 INPUT D
210 IF D=1 THEN 80
220 PRINT "Число правильных ответов";L
230 PRINT "Число неверных ответов";K
240 END
250 L=L+1
260 GOTO 180
```

Приложение 1

Перечень ключевых слов

Таблица П. 1.1

Ключевое слово	Бейсик-система										Тип инструкции	Глава
	1	2	3	4	5	6	7	8	9	10		
ABS	+	+	+	+	+	+	+	+	+	+	функция	3
ACS			+								функция	3
AND	+	+	+	+	+	+	+	+	+	+	оператор	2
APPEND				+							команда	4
ASC	+	+		+	+	+	+	+	+	+	функция	3
ASN						+					функция	3
ASSIGN		+									команда	4
AT			+								оператор	3,6
ATN	+	+	+	+	+	+	+	+	+	+	функция	3
ATTR						+					функция	6
AUTO		+			+	+	+	+	+	+	оператор	5
BASE									+		оператор	6
BCD			+								функция	3
BEEP			+				+	+	+	+	оператор	6
BIN		+	+								функция	3
BIN$			+				+	+	+	+	функция	3

Ключевое слово	Бейсик-система										Тип инструкции	Глава
	1	2	3	4	5	6	7	8	9	10		
BLOAD									+	+	команда	4
BORDER			+								оператор	6
BRIGHT			+								оператор	6
BSAVE									+	+	команда	4
CALL		+		+		+			+	+	оператор	3
CAT			+								команда	4
CATALOG				+							команда	4
CDBL					+	+	+	+	+	+	функция	3
CHAIN				+		+				+	оператор	4
CHR$	+	+	+	+	+	+	+	+	+	+	функция	3
CINT					+	+	+	+	+	+	функция	3
CIRCLE	+		+				+	+	+	+	оператор	6
CLEAR		+	+	+	+	+	+	+	+	+	оператор	3,4
CLOAD	+					+			+		команда	4
CLOSE	+	+		+	+	+		+	+	+	оператор	4
CLOSE#			+								оператор	4
CLR				+							оператор	3
CLS			+		+		+	+	+	+	оператор	6
CODE			+								оператор	4
COLOR	+			+			+	+	+	+	оператор	6
COM								+	+	+	оператор	4
COMMON						+				+	оператор	4
CON#		+									оператор	4
CONT	+	+		+	+	+	+	+	+	+	оператор	5
CONTINUE			+								оператор	5
COPY			+								команда	4
COS	+	+	+	+	+	+	+	+	+	+	функция	3
CSAVE	+					+			+		команда	4
CSNG					+	+	+	+	+	+	функция	3
CSRLIN	+						+	+	+	+	функция	4
CVD	+				+	+			+	+	функция	4
CVI	+				+	+			+	+	функция	4
CVS	+				+	+			+	+	функция	4
DATA	+	+	+	+	+	+	+	+	+	+	оператор	3
DATE$										+	функция	4
DEF	+	+		+	+	+	+	+	+	+	оператор	2,3
DEFDBL	+				+	+	+	+	+	+	оператор	2
DEFFN			+								оператор	3
DEFINT	+				+	+	+	+	+	+	оператор	2
DEF SEG										+	оператор	3,4
DEFSNG	+				+	+	+	+	+	+	оператор	2
DEFSTR	+				+	+	+	+	+	+	оператор	2
DEL				+							оператор	5
DELAY		+									оператор	4
DELETE	+	+		+	+	+	+	+	+	+	оператор	4,5
DIM	+	+	+	+	+	+	+	+	+	+	оператор	2
DIR		+									команда	4
DISABLE		+									оператор	4

Ключевое слово	Бейсик-система										Тип инструкции	Глава
	1	2	3	4	5	6	7	8	9	10		
DRAW	+		+	+				+	+	+	оператор	6
DSKF									+		функция	4
EDIT		+			+	+	+	+		+	оператор	5
ELSE	+			+	+	+	+	+	+	+	оператор	2
ENABLE		+									оператор	4
END		+		+	+	+	+	+	+	+	оператор	3
EOF	+	+			+	+		+	+	+	функция	4
EQV	+					+	+	+	+	+	оператор	2
ERASE			+			+	+	+	+	+	оператор	2,4
ERL	+				+	+	+	+	+	+	оператор	5
ERR	+				+	+	+	+	+	+	оператор	5
ERROR	+				+	+	+	+	+	+	оператор	5
EXEC		+									оператор	3
EXP	+	+	+	+	+	+	+	+	+	+	функция	3
FIELD	+				+	+			+	+	оператор	4
FILES	+					+		+	+	+	команда	4
FIRST		+									функция	3
FIX	+				+	+	+	+	+	+	функция	3
FLASH			+	+							оператор	6
FN	+	+	+	+	+	+	+	+	+	+	оператор	3
FOR	+	+	+	+	+	+	+	+	+	+	оператор	3
FORMAT			+								оператор	4
FRE		+		+	+	+	+	+	+	+	функция	3
FRE$		+									функция	3
FROM		+									оператор	3
GET	+	+	+		+	+			+	+	функция	3,4,6
GET$		+									функция	3
GO SUB			+								оператор	3
GO TO			+								оператор	3
GOSUB	+	+		+	+	+	+	+	+	+	оператор	3
GOTO	+	+		+	+	+	+	+	+	+	оператор	3
GR				+							оператор	6
HEX$	+					+	+	+	+	+	функция	3
HGR				+							оператор	6
HIMEM				+							функция	3
HOME				+							оператор	4
HTAB				+							оператор	4
IF	+	+	+	+	+	+	+	+	+	+	оператор	3
IMP	+					+	+	+	+	+	оператор	3
IN						+					функция	3
IN#		+	+								оператор	4
INK			+								оператор	6
INKEY$	+		+		+	+	+	+	+	+	функция	3
INP					+	+			+	+	функция	3
INPUT	+	+	+	+	+	+	+	+	+	+	оператор	3,4
INSTR	+	+			+	+	+	+	+	+	функция	3
INT	+	+	+	+	+	+	+	+	+	+	функция	3

Ключевое слово	Бейсик-система										Тип инструкции	Глава
	1	2	3	4	5	6	7	8	9	10		
INTERVAL									+		оператор	4
INVERSE			+	+							оператор	6
IOBYTE		+									функция	4
JOIN		+									оператор	3
KEY	+								+	+	оператор	4
KILL					+	+		+	+	+	команда	4
LAST		+									функция	3
LEFT$	+	+		+	+	+	+	+	+	+	функция	3
LEN	+	+	+	+	+	+	+	+	+	+	функция	3
LET	+	+	+	+	+	+	+	+	+	+	оператор	3
LFILES					+	+		+	+	+	команда	4
LINE	+		+		+	+	+	+	+	+	оператор	6
LINPUT		+									оператор	3,4
LIST	+	+	+	+	+	+	+	+	+	+	оператор	5
LLIST			+		+	+		+	+	+	команда	5
LN			+								оператор	3
LOAD	+	+	+	+	+	+	+	+	+	+	команда	4
LOC					+	+		+	+	+	функция	4
LOCATE							+	+	+	+	оператор	3
LOF					+	+		+	+	+	функция	4
LOG	+	+		+	+	+	+	+	+	+	функция	3
LOMEM				+							функция	3
LPOS	+					+		+	+	+	функция	4
LPRINT	+		+		+	+		+	+	+	оператор	4
LSBYTE		+									функция	3
LSET	+				+	+			+	+	оператор	4
LSHIFT		+									функция	3
LST#		+									оператор	4
LUT							+	+			оператор	6
MARGIN		+									оператор	4
MAXFILES									+		оператор	4
MEM					+						функция	3
MERGE			+		+	+	+	+	+	+	команда	4
MGR				+							оператор	6
MID$	+	+		+	+	+	+	+	+	+	функция	3
MKD$	+				+	+			+	+	функция	4
MKI$	+				+	+			+	+	функция	4
MKS$	+				+	+			+	+	функция	4
MOD	+	+				+	+	+	+	+	оператор	3
MOVE		+	+								оператор	3
MOTOR	+					+	+	+	+		команда	4
MSBYTE		+									функция	3
NAME					+	+		+	+	+	команда	4
NEW	+	+	+	+	+	+	+	+	+	+	команда	5
NEXT	+	+	+	+	+	+	+	+	+	+	оператор	3
NORMAL				+							оператор	6
NOT	+	+	+	+	+	+	+	+	+	+	оператор	2

Ключевое слово	Бейсик-система 1	2	3	4	5	6	7	8	9	10	Тип инструкции	Глава
NOTRACE				+							оператор	5
NULL		+				+					оператор	3
OCT$		+				+	+	+	+	+	функция	3
OFF	+						+	+	+	+	оператор	4,6
ON	+	+		+	+	+	+	+	+	+	оператор	3,5,6
OPEN	+	+		+	+	+		+	+	+	оператор	4
OPEN#			+								оператор	4
OPTION	+					+				+	оператор	2
OR	+	+	+	+	+	+	+	+	+	+	оператор	2
OUT		+	+		+	+			+	+	оператор	3
OVER			+								оператор	6
PAD									+		функция	4
PAINT	+						+	+	+	+	оператор	6
PAPER			+								оператор	6
PAUSE			+								оператор	4
PCLS							+	+			оператор	6
PDL				+					+		функция	4
PEEK	+	+	+	+	+	+	+	+	+	+	функция	3
PEN										+	оператор	4
PI			+								оператор	3
PLAY			+					+	+	+	оператор	6
PLOT			+	+							оператор	6
POINT	+		+		+		+	+	+	+	функция	6
POKE	+	+	+	+	+	+	+	+	+	+	оператор	3
POP				+							оператор	3
POS	+	+		+	+	+	+	+	+	+	функция	3,4
POSITION				+							оператор	4
PR#				+							оператор	4
PRESET							+	+	+	+	оператор	6
PRINT	+	+	+	+	+	+	+	+	+	+	оператор	3,4
PSET	+						+	+	+	+	оператор	6
PUN#		+									оператор	4
PUT	+				+	+			+	+	оператор	4,6
RANDOM					+						оператор	3
RANDIMIZE	+	+	+			+		+		+	оператор	3
RDR#		+									оператор	4
READ	+	+	+	+	+	+	+	+	+	+	оператор	3,4
RELOC							+	+			оператор	6
REM	+	+	+	+	+	+	+	+	+	+	оператор	2
RENAME				+							команда	4
RENUM		+				+	+	+	+	+	оператор	2
RESET		+			+	+				+	команда	3,4
RESTORE	+	+	+	+	+	+	+	+	+	+	оператор	3
RESUME	+			+	+	+	+	+	+	+	оператор	5
RETURN	+	+	+	+	+	+	+	+	+	+	оператор	3
RIBBON				+							оператор	6
RIGHT$	+	+		+	+	+	+	+	+	+	функция	3

Ключевое слово	Бейсик-система 1	2	3	4	5	6	7	8	9	10	Тип инструкции	Глава
RND	+	+	+	+	+	+	+	+	+	+	функция	3
ROT				+							оператор	6
ROTATE		+									оператор	3
RSET	+				+	+			+	+	оператор	3, 4
RSHIFT		+									функция	3
RUN	+	+	+	+	+	+	+	+	+	+	команда	5
SAVE	+	+	+	+	+	+	+	+	+	+	команда	4
SCALE			+								оператор	6
SCALL		+									оператор	3
SCRATCH		+									команда	4
SCREEN	+						+	+	+	+	оператор	6
SCREEN$			+								функция	6
SCRN				+							оператор	6
SENSE		+									функция	3
SET		+			+						оператор	3
SGN	+	+	+	+	+	+	+	+	+	+	функция	3
SIN	+	++	+	+	+	+	+	+	+		функция	3
SNG		+									оператор	3
SOUND								+	+	+	оператор	6
SPACE$	+					+	+	+	+	+	функция	3
SPC(		+		+		+	+	+	+	+	оператор	3
SPEED				+			+	+	+		оператор	4
SPRITE$									+		функция	6
SQR	+	+	+	+	+	+	+	+	+	+	функция	3
STEP	+	+	+	+	+	+	+	+	+	+	оператор	3
STICK								+	+	+	функция	4
STOP	+	+	+	+	+	+	+	+	+	+	оператор	5
STR		+									оператор	3
STR$	+	+	+	+	+	+	+	+	+	+	функция	3
STRIG								+	+	+	функция	4
STRING$	+				+	+	+	+	+	+	функция	3
SWAP						+	+	+	+	+	оператор	3
SYSTEM					+	+		+		+	команда	4
_SYSTEM									+		команда	4
TAB(	+	+	+	+	+	+	+	+	+	+	оператор	3
TAN	+	+	+	+	+	+	+	+	+	+	функция	3
TEST		+									оператор	3
TEXT				+							оператор	6
THEN	+	+	+	+	+	+	+	+	+	+	оператор	3
TIME		+							+	+	оператор	4
TIME$					+					+	функция	4
TIMER										+	оператор	4
TO	+	+	+	+	+	+	+	+	+	+	оператор	3
TRACE		+		+							команда	5
TRAP		+									команда	5
TROFF	+				+	+	+	+	+	+	команда	5
TRON	+				+	+	+	+	+	+	команда	5

Окончание табл. П. 1.1

Ключевое слово	Бейсик-система 1	2	3	4	5	6	7	8	9	10	Тип инструкции	Глава
UNS			+								оператор	3
UNTRACE			+								команда	5
UNTRAP			+								команда	5
USING	+					+	+	+	+	+	оператор	3
USR				+	+	+	+	+	+	+	функция	3
VAL	+	+	+	+	+	+	+	+	+	+	функция	3
VAL$				+							функция	3
VARPTR						+	+	+	+	+	функция	3,4
VDP									+		оператор	6
VERIFY				+							команда	4
VPEEK									+		функция	6
VPOKE									+		оператор	6
VTAB					+						оператор	4
WAIT			+		+		+			+	оператор	3
WEND	+						+			+	оператор	3
WIDTH							+	+	+	+	оператор	3
WHILE	+						+			+	оператор	3
WRITE					+		+			+	оператор	3,4
XDRAW									+		оператор	6
XOR	+	+					+	+	+	+	оператор	2

Таблица П. 1.2

Составное ключевое слово	Глава	Составное ключевое слово	Глава
FOR ... NEXT	3	ON ERROR GOSUB	5
GOSUB ... RETURN	3	ON ... GOTO	3
IF ... GOTO	3	ON ... GOSUB	3
IF ... THEN	3	ON ... KEY	4
IF ... THEN ... ELSE ..	3	ON PEN GOTO	4
INTERVAL ON	4	ON SPRITE GOTO	6
INTERVAL OFF	4	ON STRIG GOTO	4
INTERVAL STOP	4	PEN ON	4
KEY LIST	4	PEN OFF	4
KEY ON	4	PEN STOP	4
KEY OFF	4	PEN (n) ON	4
KEY STOP	4	PEN (n) OFF	4
KEY (n) ON	4	PEN (n) STOP	4
KEY (n) OFF	4	PRINT USING	3
KEY (n) STOP	4	PRINT USING #	4
LINE INPUT	3	PUT SPRITE	6
LINE INPUT #	4	SPRITE ON	6

Окончание табл. 1.2

Составное ключевое слово	Глава	Составное ключевое слово	Глава
LPRINT USING	3	SPRITE OFF	6
MOTOR ON	4	SPRITE STOP	6
MOTOR OFF	4	STRIG ON	4
ON COM (n)	4	STRIG OFF	4
ON COM (n) GOSUB	4	STRIG STOP	4
ON ERROR GOTO	5	WHILE ... WEND	3

Приложение 2

Внутреннее представление программ, написанных на языке Бейсик

При внутреннем представлении программ* инструкции и знаки операций представлены определенными 16-ричными кодами или наборами кодов, перечень которых для MBASIC приведен в табл. П.2.1 – П.2.3.

Программа на языке Бейсик записывается на НГМД с помощью команды SAVE. В зависимости от параметров в команде SAVE программа сохраняется на диске в трех различных форматах. Рассмотрим небольшую программу PRIMER.BAS и ее внутреннее представление**.

```
10 A=1:B%=&H1:C%=&O1
20 PRINT "TABL"
30 A=A+1
40 B%=B%+1
50 C%=C%+1
60 PRINT A;TAB(10);HEX$(B%);OCT(C%)
70 IF A=100 THEN END
80 GOTO 30
```

Если программа будет записана на НГМД с помощью команды SAVE "PRIMER.BAS",A, то каждый символ программы сохранится в коде ASCII (КОИ-8) и ее внутреннее представление будет следующим:

```
31 30 20 41 3D 31 20 3A 20 42 25 3D 26 48 31 20
3A 20 43 25 3D 26 4F 31 0D 0A 32 30 20 50 52 49
4E 54 20 22 54 41 42 4C 22 0D 0A 33 30 20 41 3D
41 2B 31 0D 0A 34 30 20 42 25 3D 42 25 2B 31 0D
0A 35 39 20 43 25 3D 43 25 2B 31 0D 0A 36 30 20
50 52 49 4F 54 20 41 3B 54 41 42 28 31 30 29 3B
48 45 58 24 28 42 25 29 2C 4F 43 54 24 28 43 25
29 0D 0A 37 30 20 49 46 20 41 3D 31 30 30 20 54
48 45 4E 20 45 4F 44 0D 0A 38 30 20 47 4F 54 4F
20 33 30 0D 0A
```

* Дано на примере Бейсик-системы MBASIC.

** Ниже в приведенных распечатках содержимого памяти используются 16-ричные числа.

В этой записи два байта с кодами 0D 0A указывают на конец программной строки. Файлы в коде ASCII (КОИ-8) будут занимать больше памяти, но могут использоваться как файлы данных последовательного доступа.

Если программа будет записана на НГМД с помощью команды SAVE "PRIMER.BAS,P, то программа сохранится в закодированном двоичном формате и ее представление будет следующим:

```
FE 24 1D 2B C1 EB 07 95 48 6E 4F 6C E6 DF E9 11
59 52 70 EB A1 F0 C5 B9 2B E0 87 88 C2 20 BE EE
4A C9 52 A2 4F 34 7A 70 12 AE 0E 17 2B 4B 6C 3D
1B 05 7A BA A8 2C 68 74 08 AA 6D 11 B5 F6 90 3D
DF D5 9E BC 11 FC 47 F0 02 E3 38 E6 F2 64 4F 3D
55 70 7D 36 8C C7 2B 4A 5B 1F 60 2A 8E 2A BF BE
43 8A 52 50 BC 88 C8 F0 87 F7 96 71 EC 8B 92 BA
38 89 88 6E 02 9D DE 4B 8C 69 7D 5A A7 AF 21 1A
```

Такую программу нельзя ни отредактировать, ни распечатать.

Если программа записана на НГМД с помощью команды SAVE "PRIMER.BAS", то программа будет сохраняться в упакованном формате и ее внутреннее представление будет следующим:

```
FF E4 61 0A 00 41 F0 12 20 3A 20 42 25 F0 0C 01
00 20 3A 20 43 25 F0 0B 01 00 00 F1 61 14 00 91
20 22 54 41 42 4C 22 00 FB 01 1E 00 41 F0 41 F2
12 00 07 62 28 00 42 25 F0 42 25 F2 12 00 13 62
32 00 43 25 F0 43 25 F2 12 00 2E 62 3C 00 91 20
41 3B D0 0F 0A 29 3B FF 9A 28 42 25 29 2C FF 99
28 43 25 29 00 3D 62 46 00 8B 20 41 F0 0F 64 20
CF 20 81 00 47 62 50 00 89 20 0E 1E 00 00 00 00
```

Программа сохраняется в формате, в котором все синтаксические элементы хранятся в упакованном виде. Ключевые слова представляются однобайтовыми 16-ричными кодами. Эти коды имеют значения больше 80. Код FF является признаком встроенной функции. Ниже приводится перечень ключевых слов и их кодов.

Коды команд и операторов

Таблица П. 2.1

Команда/ оператор	Код	Команда/ оператор	Код	Команда/ оператор	Код	Команда/ оператор	Код
AUTO	AB	ERR	D7	MOD	FC	RESET	CC
AND	F7	EQV	FA	NEXT	83	RANDOMIZE	BB
CLOSE	C3	FOR	82	NULL	96	STOP	90
CONT	9A	FIELD	C0	NAME	C7	SWAP	A5
CLEAR	92	FILES	C6	NEW	94	SAVE	CB
CALL	B6	FN	D3	NOT	D5	SPC	D4
COMMON	B8	GOTO	89	OPEN	BF	STEP	D1
CHAIN	B9	GOSUB	8D	OUT	9D	STRING$	D8
DELETE	AA	GET	C1	ON	95	SYSTEM	BD
DATA	84	INPUT	85	OR	F8	THEN	CF
DIM	86	IF	8B	OPTION	BA	TRON	A3
DEFSTR	AD	INSTR	DA	PRINT	91	TROFF	A4
DEFINT	AE	IMP	FB	PUT	C2	TAB(	D0

Окончание табл. П. 2.1

Команда/ оператор	Код	Команда/ оператор	Код	Команда/ оператор	Код	Команда/ оператор	Код
DEFSNG	AF	INKEY$	DD	POKE	99	TO	CE
DEFDBL	B0	KILL	C8	RETURN	8E	USING	D9
DEF	98	LPRINT	9E	READ	87	USR	D2
ELSE	A2	LLIST	9F	RUN	8A	VARPTR	DC
END	81	LET	88	RESTORE	8C	WIDTH	A1
ERASE	A6	LOAD	C4	REM	8F	WHILE	B4
EDIT	A7	LSET	C9	RESUME	A9	WEND	B5
ERROR	A8	LIST	93	RSET	CA	WRITE	B7
ERL	D6	MERGE	C5	RENUM	AC	XOR	F9

Коды функций

Таблица П. 2.2

Функция	Код	Функция	Код	Функция	Код	Функция	Код
ABS	FF 86	CHR$	FF 96	LOC	FF B0	PEEK	FF 97
ATN	FF 8E	EXP	FF 8B	LEN	FF 92	RIGHT$	FF 82
ASC	FF 95	EOF	FF AF	LEFT$	FF 81	RND	FF 88
CINT	FF 9C	FRE	FF 8F	LOF	FF B1	SGN	FF 84
CSNG	FF 9D	FIX	FF 9F	MKI$	FF B2	SQR	FF 87
CDBL	FF 9E	HEX$	FF 9A	MKS$	FF B3	SIN	FF 89
CVI	FF AB	INT	FF 85	MKD$	FF B4	STR$	FF 93
CVS	FF AC	INP	FF 90	MID$	FF 83	SPACE$	FF 98
CVD	FF AD	LPOS	FF 9B	OCT$	FF 99	TAN	FF 8D
COS	FF 8C	LOG	FF 8A	POS	FF 91	VAL	FF 94

Коды операций

Таблица П, 2.3

Операция	Код	Операция	Код	Операция	Код
>	EF	=	F0	<	F1
+	F2	-	F3	*	F4
/	F5	^	F6	\	FD

При записи на диск в разных форматах, примеры которых представлены выше, имена переменных могут состоять из букв и цифр, т. е. их коды расположены между кодами 30 и 5A включительно. В коде ASCII (КОИ-8) коды русского алфавита расположены между кодами C0 и FE включительно, поэтому их нельзя использовать в именах переменных; строки, содержащие буквы русского алфавита, в операторах REM и DATA должны заключаться в кавычки. Иначе, когда программа будет выводиться на экран по команде LIST, вместо букв русского алфавита будут печататься инструкции языка Бейсик. Например, вместо прописной буквы "Э" появится MOD (код FC), а вместо строчной буквы "б" – PUT (код C2) и т. д.

При записи по команде SAVE (без параметра A или P) коды 0E, 1C, 0C и 0B указывают, что в следующих двух байтах находятся соответственно номер строки (после кодов, соответст-

вующих GOTO и GOSUB), целое число, 16-ричное число и 8-ричное число. Код 1D указывает, что в следующих четырех байтах находится число одинарной точности. Код 1F указывает, что в следующих восьми байтах находится число двойной точности.

Десятичное число представляется следующим образом:

если положительное число меньше или равно 9, то оно задается одним байтом, значение которого лежит в диапазоне 11 — 1A, т. е. 0 = 11, 1 = 12, ..., 9 = 1A;

если число меньше или равно 255, но больше или равно 10, то оно задается двумя байтами, код первого байта 0F, а второй байт — число, т. е. число 10 представляется как 0F0A.

Байт со значением 00 является концом программной строки.

Интерпретатор BASIC размещает программу на языке Бейсик в ОЗУ с определенного адреса (для MBASIC — 61C9). В ячейке 61C9 расположен байт со значением 00. При записи программы на НГМД значение этого байта меняется на FF. В ячейках памяти ОЗУ 603B и 603C находится адрес, максимально допустимый для программы на языке Бейсик.

Каждая программная строка занимает N байтов и находится в памяти в формате, представленном в табл. П.2.4.

Таблица П. 2.4

Номер байта	Содержимое памяти
1-2	Адрес памяти первого байта следующей строки программы
3-4	Номер строки
5-(N-1)	Упакованная строка
N	Конец строки (00H)

В табл. П.2.5 и П.2.6 даны коды и соответствующие им ключевые слова для рассматриваемых в справочнике версий языка Бейсик. Число версий в табл. П.2.6 ограничено пятью, так как в соответствии с построением интерпретаторов в остальных версиях языка функции включены в состав табл. П.2.5.

Таблица П. 2.5

Код	Ключевые слова в версиях				
	XYBASIC	Бейсик СПЕКТРУМ+2	Бейсик АГАТ	Бейсик TRS-80	MBASIC
80			END	END	
81			FOR	FOR	END
82			NEXT	RESET	FOR
83			DATA	SET	NEXT
84			INPUT	CLS	DATA
85			DEL	CMD	INPUT

Код	Ключевые слова в версиях				
	XYBASIC	Бейсик СПЕКТРУМ+2	Бейсик АГАТ	Бейсик TRS-80	MBASIC
86			DIM	RANDOM	DIM
87			READ	NEXT	READ
88	LET		GR=	DATA	LET
89	IF		TEXT=	INPUT	GOTO
8A	FOR		PR#	DIM	RUN
8B	NEXT		IN#	READ	IF
8C	GOTO		CALL	LET	RESTORE
8D	GOSUB		PLOT	GOTO	GOSUB
8E	PRINT		!	RUN	RETURN
8F	INPUT		&	IF	REM
90	READ	(a)	MGR=	RESTORE	STOP
91	RETURN	(b)	HGR=	GOSUB	PRINT
92	DEF	(c)	RIBBON=	RETURN	CLEAR
93	DIM	(d)	&	REM	LIST
94	DATA	(e)	DRAW	STOP	NEW
95	RESTORE	(f)	XDRAW	ELSE	ON
96	STOP	(g)	HTAB	TRON	NULL
97	END	(h)	HOME	TROFF	WAIT
98	NULL	(i)	ROT=	DEFSTR	DEF
99	CALL	(j)	SCALE	DEFINT	POKE
9A	SCALL	(k)	SHLOAD	DEFSNG	CONT
9B	OUT	(l)	TRACE	DEFDBL	
9C	POKE	(m)	NOTRACE	LINE	
9D	WAIT	(n)	NORMAL	EDIT	OUT
9E	UNTRACE	(o)	INVERSE	ERROR	LPRINT
9F	TRACE	(p)	FLASH	RESUME	LLIST
A0	UNTRAP	(q)	COLOR=	OUT	
A1	TRAP	(r)	POP	ON	WIDTH
A2	BREAK	(s)	VTAB	OPEN	ELSE
A3	UNBREAK	SPECTRUM	HIMEM:	FIELD	TRON
A4	ENABLE	PLAY (u)	LOMEM:	GET	TROFF
A5	DISABLE	RND	ONERR	PUT	SWAP
A6	RANDOMIZE	INKEY$	RESUME	CLOSE	ERASE
A7	REM	PI	RECALL	LOAD	EDIT
A8	DELAY	FN	STORE	MERGE	ERROR
A9	SAVE	POINT	SPEED=	NAME	RESUME
AA	LOAD	SCREEN$	LET	KILL	DELETE
AB	LIST	ATTR	GOTO	LSET	AUTO
AC	CLEAR	AT	RUN	RSET	RENUM
AD	NEW	TAB	IF	SAVE	DEFSTR
AE	RUN	VAL$	RESTORE	SYSTEM	DEFINT
AF	CONT	CODE	&	LPRINT	DEFSNG
B0	AUTO	VAL	GOSUB	DEF	DEFDBL
B1	DELETE	LEN	RETURN	POKE	LINE
B2	EDIT	SIN	REM	PRINT	
B3	RENUM	COS	STOP	CONT	

Продолжение табл. П. 2.5

Код	Ключевые слова в версиях				
	XYBASIC	Бейсик СПЕКТРУМ+2	Бейсик АГАТ	Бейсик TRS-80	MBASIC
B4	MOVE	TAN	ON	LIST	WHILE
B5	EXEC	ASN	WAIT	LLIST	WEND
B6	OPEN	ACS	LOAD	DELETE	CALL
B7	CLOSE	ATN	SAVE	AUTO	WRITE
B8	LINPUT	LN	DEF	CLEAR	COMMON
B9	MARGIN	EXP	POKE	CLOAD	CHAIN
BA	DIR	INT	PRINT	BSAVE	OPTION
BB	SCRATCH	SQR	CONT	NEW	RANDOMIZ
BC	ASSIGN	SGN	LIST	TAB(	
BD	TIME	ABS	CLEAR	TO	SYSTEM
BE	ON	PEEK	GET	FN	
BF	FN	IN	NEW	USING	OPEN
C0		USR	TAB(	VARPTR	FIELD
C1	+	STR$	TO	USR	GET
C2	-	CHR$	FN	ERL	PUT
C3	*	NOT	SPC(	ERR	CLOSE
C4	/	BIN	THEN	STRING	LOAD
C5	\	OR	AT	INSTR	MERGE
C6	^	AND	NOT	POINT	FILES
C7	<=	<=	STEP	TIME$	NAME
C8		>=		MEM	KILL
C9		<>		INKEY$	LSET
CA	>=	LINE	*	THEN	RSET
CB	<>	THEN	/	NOT	SAVE
CC	=	TO	^	STEP	RESET
CD	<	STEP	AND	+	
CE	>	DEF FN	OR	-	TO
CF	AND	CAT	>	*	THEN
D0	XOR	FORMAT	=	/	TAB(
D1	OR	MOVE	<	^	STEP
D2	MOD	ERASE	SGN	AND	USR
D3	JOIN	OPEN#	INT	OR	FN
D4	GET	CLOSE#	ABS	>	SPC(
D5	FRE	MERGE	USR	=	NOT
D6	RND	VERIFY	FRE	<	ERL
D7	POS	BEEP	SCRN(	SGN	ERR
D8	FIRST	CIRCLE	PDL	INT	STRING
D9	LAST	INK	POS	ABS	USING
DA	EOF	PAPER	SQR	FRE	INSTR
DB	UNS	FLASH	RND	INP	
DC	INT	BRIGHT	LOG	POS	VARPTR
DD	SQR	INVERSE	EXP	SQR	INKEY$
DE	EXP	OVER	COS	RND	
DF	LOG	OUT	SIN	LOG	
E0	SIN	LPRINT	TAN	EXP	
E1	COS	LLIST	ATN	COS	

Код	Ключевые слова в версиях				
	XYBASIC	Бейсик СПЕКТРУМ+2	Бейсик АГАТ	Бейсик TRS-80	MBASIC
E2	TAN	STOP	PEEK	SIN	
E3	ATN	READ	LEN	TAN	
E4	BIN$	DATA	STR$	ATN	
E5	HEX$	RESTORE	VAL	PEEK	
E6	OCT$	NEW	ASC	CVI	
E7	CHR$	BORDER	CHR$	CVS	
E8	STR$	CONTINUE	LEFT$	CVD	
E9	ASC	DIM	RIGHT$	EOF	
EA	LEN	REM	MID$	LOC	
EB	VAL	FOR		LOF	
EC	LEFT$	GOTO		MKI$	
ED	RIGHT$	GOSUB		MKS$	
EE	MID$	INPUT		MKD$	
EF	INSTR	LOAD		CINT	>
F0	IOBYTE	LIST		CSNG	=
F1	SGN	LET		CDBL	<
F2	ABS	PAUSE		FIX	+
F3	MSBYTE	NEXT		LEN	-
F4	LSBYTE	POKE		STR$	*
F5	BCD	PRINT		VAL	/
F6	BIN	PLOT		ASC	
F7	PEEK	RUN		CHR$	AND
F8	ROTATE	SAVE		LEFT$	OR
F9	TEST	RANDOMIZE		RIGHT$	XOR
FA	SENSE	IF		MID$	EQV
FB	RSHIFT	CLS			IMP
FC	LSHIFT	DRAW			MOD
FD	RESET	CLEAR			\
FE	SET	RETURN			
FF	IN	COPY			

Таблица П. 2.6

Код	Ключевые слова в версиях			
	Бейсик ПК8010	Бейсик ПК8020	MSX-BASIC	BASICA
80				
81	END	END	END	END
82	FOR	FOR	FOR	FOR
83	NEXT	NEXT	NEXT	NEXT
84	DATA	DATA	DATA	DATA
85	INPUT	INPUT	INPUT	INPUT
86	DIM	DIM	DIM	DIM
87	READ	READ	READ	READ
88	LET	LET	LET	LET
89	GOTO	GOTO	GOTO	GOTO
8A	RUN	RUN	RUN	RUN
8B	IF	IF	IF	IF
8C	RESTORE	RESTORE	RESTORE	RESTORE
8D	GOSUB	GOSUB	GOSUB	GOSUB
8E	RETURN	RETURN	RETURN	RETURN
8F	REM	REM	REM	REM
90	STOP	STOP	STOP	STOP
91	PRINT	PRINT	PRINT	PRINT
92	CLEAR	CLEAR	CLEAR	CLEAR
93	LIST	LIST	LIST	LIST
94	NEW	NEW	NEW	NEW
95	ON	ON	ON	ON
96			WAIT	WAIT
97	COM		DEF	DEF
98	DEF	DEF	POKE	POKE
99	POKE	POKE	CONT	CONT
9A	CONT	CONT	CSAVE	
9B			CLOAD	
9C			OUT	OUT
9D	MOTOR	MOTOR	LPRINT	LPRINT
9E		LPRINT	LLIST	LLIST
9F		LLIST	CLS	
A0			WIDTH	WIDTH
A1	WIDTH	WIDTH	ELSE	ELSE
A2	ELSE		TRON	TRON
A3	TRON	TRON	TROFF	TROFF
A4	TROFF	TROFF	SWAP	SWAP
A5	SWAP	SWAP	ERASE	ERASE
A6	ERASE	ERASE	ERROR	EDIT
A7	EDIT	EDIT	RESUME	ERROR
A8	ERROR	ERROR	DELETE	RESUME
A9	RESUME	RESUME	AUTO	DELETE
AA	DELETE	DELETE	RENUM	AUTO
AB	AUTO	AUTO	DEFSTR	RENUM

Продолжение табл. П. 2.6

Код	Ключевые слова в версиях			
	Бейсик ПК 8010	Бейсик ПК 8020	MSX-BASIC	BASICA
AC	RENUM	RENUM	DEFINT	DEFSTR
AD	DEFSTR	DEFSTR	DEFSNG	DEFINT
AE	DEFINT	DEFINT	DEFDBL	DEFSNG
AF	DEFSNG	DEFSNG	LINE	DEFDBL
B0	DEFDBL	DEFDBL	OPEN	LINE
B1	LINE	LINE	FIELD	WHILE
B2	SPEED	SPEED	GET	WEND
B3			PUT	CALL
B4			CLOSE	
B5			LOAD	
B6			MERGE	
B7			FILES	WRITE
B8			LSET	OPTION
B9			RSET	RANDOMIZE
BA			SAVE	OPEN
BB		RANDOMIZE	LFILES	CLOSE
BC	BEEP	BEEP	CIRCLE	LOAD
BD		SYSTEM	COLOR	MERGE
BE			DRAW	SAVE
BF		OPEN	PAINT	COLOR
C0			BEEP	CLS
C1			PLAY	MOTOR
C2			PSET	BSAVE
C3		CLOSE	PRESET	BLOAD
C4	LOAD	LOAD	SOUND	SOUND
C5	MERGE	MERGE	SCREEN	BEEP
C6		FILES	VPOKE	PSET
C7		NAME	SPRITE	PRESET
C8		KILL	VDP	SCREEN
C9			BASE	KEY
CA			CALL	LOCATE
CB	SAVE	SAVE	TIME	
CC			KEY	TO
CD		LFILES	MAX	THEN
CE	CLS	CLS	MOTOR	TAB(
CF	PCLS	PCLS	BLOAD	STEP
D0	COLOR	COLOR	BSAVE	USR
D1	CIRCLE	CIRCLE	DSKO	FN
D2		DRAW	SET	SPC(
D3	PAINT	PAINT	NAME	NOT
D4	PSET	PSET	KILL	ERL
D5	PRESET	PRESET	IPL	ERR
D6	LOCATE	LOCATE	COPY	STRING
D7	SCREEN	SCREEN	CMD	USING
D8			LOCATE	INSTR
D9	LUT	LUT	TO	

Код	Ключевые слова в версиях			
	Бейсик ПК 8010	Бейсик ПК 8020	MSX-BASIC	BASICA
DA	RELOC	RELOC	THEN	VARPTR
DB			TAB(	CSRLIN
DC	TO	TO	STEP	POINT
DD	THEN	THEN	USR	OFF
DE	TAB(	TAB(	FN	INKEY$
DF	STEP	STEP	SPC(	
E0	USR	USR	NOT	
E1	FN	FN	ERL	
E2	SPC(	SPC(	ERR	
E3	NOT	NOT	STRING$	
E4	ERL	ERL	USING	
E5	ERR	ERR	INSTR	
E6	STRING$	STRING$	'	>
E7	USING	USING	VARPTR	=
E8	INSTR	INSTR	CSRLIN	<
E9		'	ATTR$	+
EA	VARPTR	VARPTR	DSKI$	-
EB	CSRLIN	CSRLIN	OFF	*
EC	OFF	OFF	INKEY$	/
ED	INKEY$	INKEY$	POINT	^
EE	POINT	POINT	>	AND
EF	>	>	=	OR
F0	=	=	<	XOR
F1	<	<	+	EQV
F2	+	+	-	IMP
F3	-	-	*	MOD
F4	*	*	/	\
F5	/	/	^	
F6	^	^	AND	
F7	AND	AND	OR	
F8	OR	OR	XOR	
F9	XOR	XOR	EQV	
FA	EQV	EQV	IMP	
FB	IMP	IMP	MOD	
FC	MOD	MOD	\	
FD	\	\		
FE				
FF				

Таблица П. 2.7

Код	Функции в версиях				
	MBASIC	Бейсик ПК8010	Бейсик ПК8020	MSX-BASIC	BASICA
01	LEFT$	LEFT$	LEFT$	LEFT$	LEFT$
02	RIGHT$	RIGHT$	RIGHT$	RIGHT$	RIGHT$
03	MID$	MID$	MID$	MID$	MID$
04	SGN	SGN	SGN	SGN	SGN
05	INT	INT	INT	INT	INT
06	ABS	ABS	ABS	ABS	ABS
07	SQR	SQR	SQR	SQR	SQR
08	RND	RND	RND	RND	RND
09	SIN	SIN	SIN	SIN	SIN
0A	LOG	LOG	LOG	LOG	LOG
0B	EXP	EXP	EXP	EXP	EXP
0C	COS	COS	COS	COS	COS
0D	TAN	TAN	TAN	TAN	TAN
0E	ATN	ATN	ATN	ATN	ATN
0F	FRE	FRE	FRE	FRE	FRE
10	INP	BIN$	BIN$	INP	INP
11	POS	POS	POS	POS	POS
12	LEN	LEN	LEN	LEN	LEN
13	STR$	STR$	STR$	STR$	STR$
14	VAL	VAL	VAL	VAL	VAL
15	ASC	ASC	ASC	ASC	ASC
16	CHR$	CHR$	CHR$	CHR$	CHR$
17	PEEK	PEEK	PEEK	PEEK	PEEK
18	SPACE$	SPACE$	SPACE$	VPEEK	SPACE$
19	OCT$	OCT$	OCT$	SPACE$	OCT$
1A	HEX$	HEX$	HEX$	OCT$	HEX$
1B	LPOS		LPOS	HEX$	LPOS
1C	CINT	CINT	CINT	LPOS	CINT
1D	CSNG	CSNG	CSNG	BIN$	CSNG
1E	CDBL	CDBL	CDBL	CINT	CDBL
1F	FIX	FIX	FIX	CSNG	FIX
20				CDBL	PEN
21				FIX	STICK
22				STICK	STRIG
23				STRIG	EOF
24				PDL	LOC
25				PAD	LOF
26				DSKF	
27				FPOS	
28				CVI	
29				CVD	
2A				CVS	
2B	CVI			EOF	
2C	CVS			LOC	

Окончание табл. П. 2.7

Код	Функции в версиях				
	MBASIC	Бейсик ПК 8010	Бейсик ПК 8020	MSX-BASIC	BASICA
2D	CVD			LOF	
2E				MKI$	
2F	EOF		EOF	MKS$	
30	LOC		LOC	MKD$	
31	LOF		LOF		
32	MKI$				
33	MKS$				
34	MKD$				

Список литературы

1. ГОСТ 27787–88. Язык программирования Бейсик.

2. **Кетков Ю. Л.** Диалог на языке Бейсик для мини- и микроЭВМ. – М.: Наука, 1988. – 368 с.

3. **Морил Г.** Бейсик для ПК ИБМ:Пер. с англ./Под ред. С. В. Черемных. – М.: Финансы и статистика, 1987. – 208 с.

4. **Пул Л.** Работа на персональном компьютере. – М.: Мир, 1986. – 383 с.

5. **Корчак А. Е.** Язык программирования Бейсик для микроЭВМ. – М.: Изд. МЦНТИ, 1988. – 130 с.

6. **Башмакова Е. С., Либеров А. Б.** Язык программирования Бейсик. – М.: Изд. МЦНТИ, 1987. – 184 с.

7. **Кучура Н. А., Ходош М. В., Цагельский В. И.** Персональные ЭВМ единой системы. Бейсик. – М.: Финансы и статистика, 1988. – 149 с.

8. **Блэнд Г.** Основы программирования на языке Бейсик в стандарте MSX: Пер. с англ. – М.: Финансы и статистика, 1989. – 208 с.

9. **Дьяконов В. П.** Применение персональных ЭВМ и программирование на языке Бейсик. – М.: Радио и связь, 1989. – 304 с.

10. **Уолш Б.** Программирование на Бейсике: Пер. с англ. – М.: Радио и связь, 1988. – 336 с.

Оглавление

Справочное издание

Башмакова Елена Станиславовна, Витенберг Исаак Моисеевич, Либеров Александр Борисович, Пашков Александр Леонидович

ПРОГРАММИРОВАНИЕ МИКРОЭВМ НА ЯЗЫКЕ БЕЙСИК

Справочник

Заведующая редакцией Г. И. Козырева
Редактор В.И. Ченцова
Переплет художника Н. А. Пашуро
Художественный редактор А. В. Проценко
Технический редактор А. Н. Золотарева
Корректор А. К. Акименкова

ИБ № 1778

Подписано в печать с оригинал-макета 22. 02. 91 Формат 60×88 1/16 Бумага офсетная №
Гарнитура "пресс-роман" Печать офсетная Усл.печ.л. 14,70 Усл.кр.-отт. 14.
Уч.изд.л. 16,21 Тираж 50 000 экз. Изд. № 22259 Зак. № 6301 Цена 3 р.

Издательство "Радио и связь". 101000 Москва, Почтамт, а/я 693

Ордена Октябрьской Революции и ордена Трудового Красного Знамени МПО "Первая Образцовая типография" Государственного комитета СССР по печати 113054, Москва, Валовая, 28.

3р.