Утверждён
00008—01.35.04 ЛУ

ПЕРСОНАЛЬНАЯ ЭВМ «ЭЛЕКТРОНИКА МС 0513»
ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ
ЯЗЫК «БЕЙСИК»

ОПИСАНИЕ ЯЗЫКА
00008-01.35.04

Утверждён
00008—01.35.04 ЛУ

ПЕРСОНАЛЬНАЯ ЭВМ «ЭЛЕКТРОНИКА МС 0513»

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

ЯЗЫК «БЕЙСИК»

ОПИСАНИЕ ЯЗЫКА

00008-01.35.04

АННОТАЦИЯ

Данный документ представляет собой руководство по языку программирования Бейсик (версия 1987.12.25.) и предназначен для пользователей микро-ЭВМ «Электроника БК 0011».

Язык Бейсик создан для решения математических и инженерных задач в режиме диалога человек-ЭВМ. Он позволяет создавать программы для большого круга задач, сочетая в себе простоту использования и лёгкость изучения. Описываемая версия Бейсик-системы существенно расширена по сравнению с ядром языка Бейсик. Она включает операторы, которые позволяют создавать программы по обработке текстовой информации, использовать этот язык для программирования систем управления технологическими установками, а также игровых задач и т.д.

В настоящем руководстве приводятся сведения о синтаксисе и семантике операторов, функций языка и команд системы.

Перед изучением описания языка Бейсик необходимо ознакомиться с руководством по эксплуатации микро-ЭВМ «Электроника БК-0011» 3.057.004РЭ, руководством оператора 00008-01.34.04 и руководством системного программиста 00008-01.32.01.

ВВЕДЕНИЕ

1. ОБЩИЕ СВЕДЕНИЯ

Программа на языке Бейсик состоит из строк, которые могут содержать операторы и команды. Операторы и команды могут вводиться и выполняться в одном из двух режимов: косвенном и непосредственном.

2. КОСВЕННЫЙ РЕЖИМ

Чаще всего программа на языке Бейсик записывается в косвенном режиме. В этом случае каждая строка начинается с номера. За номером строки следует оператор. Завершается строка управляющим символом <ПС> (перевод строки), получаемым при нажатии на клавишу «ВВОД». Номер строки должен быть в диапазоне от 0 до 65535. Номер строки выполняет две функции: во-первых, он служит меткой оператора и может быть использован для ссылки на данный оператор; во-вторых, номер строки определяет порядок выполнения операторов. Кроме того, наличие номеров строк облегчает отладку программы, т.к. в сообщении об ошибке указывается номер строки, в которой встретилась ошибка. Любой из операторов языка Бейсик должен размещаться в одной строке, максимальная длина строки – 255 символов.

Номер строки используется и при редактировании программы. Ввод строки, номер которой уже существует, приводит к замене прежней строки. Ввод только номера строки (пустая строка) вызывает стирание соответствующей строки.

3. НЕПОСРЕДСТВЕННЫЙ РЕЖИМ

В зависимости от наличия или отсутствия номера строки Бейсик-система отличает строки, вводимые в косвенном режиме, от строк, вводимых для непосредственного выполнения. Операторы, которые начинаются с номера строки, запоминаются, операторы без номера строки выполняются непосредственно по мере их ввода в систему, т.е. осуществляется выполнение операторов в непосредственном режиме.

Следовательно, при вводе строки

```
10 PRINT «ЭЛЕКТРОНИКА БК 0010»
```

Строка заносится в текст программы, тогда как ввод строки

```
PRINT «ЭЛЕКТРОНИКА БК 0010»
```

Вызывает его непосредственное выполнение и в результате осуществляется немедленный вывод сообщения

```
ЭЛЕКТРОНИКА БК 0010
```

Непосредственный режим Бейсик-системы позволяет использовать вычислительную машину как очень мощный калькулятор.

4. СПОСОБ ОПИСАНИЯ ЯЗЫКА БЕЙСИК

Для формального описания языка используется ряд обозначений, которые необходимо понимать в следующем смысле:

— информация, записанная прописными буквами латинского алфавита, постоянна;

— информация, записанная буквами русского алфавита и заключённая в угловые скобки «< >», переменная;

— информация, заключённая в квадратные скобки «[ ]», при использовании может быть опущена;

— из параметров, размещённых друг под другом, должен быть выбран только один;

— параметр, заключённый в квадратные скобки [<параметр>...]N, сопровождаемый многоточием, используется для обозначения многократности, N указывает верхний предел возможных повторений параметра, если N не указано, то число повторений параметра не определено.

Примечание. В операторах с несколькими операндами в случае, если указаны последующие операнды, пропущенным операндам должны соответствовать запятые, например:

```
CIRCLE (100,100),50,,,,2
```

5. СООБЩЕНИЯ ОБ ОШИБКАХ

При обнаружении ошибки во время ввода, синтаксического анализа или выполнения программы Бейсик-система выдаёт сообщение об ошибке. Формат сообщения следующий:

```
<ОШИБКА> В СТРОКЕ NNNNN
```

Где <ОШИБКА> – короткое сообщение о виде ошибки, NNNNN – номер строки, содержащей ошибку.

В режиме непосредственного выполнения номер строки не указывается.

Пояснения ошибок приведены в приложении 1.

6. ОСНОВНЫЕ СИМВОЛЫ ЯЗЫКА

Все языковые конструкции могут быть выражены с помощью основных символов:

— прописные и строчные буквы латинского алфавита:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z;

— цифры: 0 1 2 3 4 5 6 7 8 9;

— специальные символы, приведённые в таблице 1.

Таблица 1

СПЕЦИАЛЬНЫЕ СИМВОЛЫ

Символ	Название символа
	Пробел
:	Двоеточие
=	Равно
+	Плюс
-	Минус
*	Звёздочка
/	Дробная черта
^	Знак отрицания
(	Левая круглая скобка
)	Правая круглая скобка
[	Левая квадратная скобка
]	Правая квадратная скобка
,	Запятая
;	Точка с запятой
.	Точка
« »	Кавычки
'	Апостроф
$	Знак денежной единицы
<	Меньше
>	Больше
?	Знак вопроса
@	Знак «относительно»
\	Обратная дробная черта
&	Амперсанд
!	Восклицательный знак
%	Процент
#	Знак номера
_	Знак подчёркивания

Бейсик-система использует ряд слов для обозначения команд, операторов, имён функций и т.п. Изменять значения этих слов пользователь не может, поэтому такие слова называются зарезервированными.

К ним в Бейсик-системе относятся:

— имена команд, например: LIST, RUN, RENUM;

— имена операторов и разделителей, например: LET, IF, THEN, GOTO, GOSUB, FOR, NEXT;

— имена арифметических и символьных функций, например: SIN, COS, TAN, EXP, LOG, SQR.

Зарезервированные слова разрешается укорачивать до трех символов, что облегчает ввод программ. Полный список зарезервированных слов

приводится в приложении 3.

Символ пробела не является значащим и поэтому может свободно использоваться для улучшения наглядности программы. Исключение составляет случай записи символа пробела в текстовой константе.

Примечание. Не допускается использование пробела внутри зарезервированных слов, чисел и имён переменных.

1. ЭЛЕМЕНТЫ ЯЗЫКА

1.1.Команды

Команды Бейсика служат для связи программиста с системой; в основном через них осуществляется начальный диалог человека с ЭВМ. Команды не имеют номера строки и выполняются только в непосредственном режиме. При попытке выполнить команду в косвенном режиме происходит ошибка 12.

1.2.Операторы

Операторы языка изменяют значения переменных, естественный порядок вычислений согласно алгоритму решения задачи, резервируют память под переменные и константы и т.д. С некоторыми исключениями все операторы могут выполняться как в косвенном, так и в непосредственном режимах.

1.3.Константы

В Бейсик-системе используются числовые и текстовые константы.

1.3.1. Числовые константы

В Бейсик-системе допустимы шесть форм записи числовых констант.

1. Целые константы имеют вид десятичных целых чисел со знаком или без него.

Примеры:

```
123
-2345
```

2. Вещественные константы, представленные в форме чисел с фиксированной запятой, – это положительные или отрицательные числа, имеющие десятичную точку.

Примеры:

```
2.45
-102.36
```

3. Вещественные константы, записываемые в экспоненциальной форме, состоят из мантиссы и порядка, отделённого от мантиссы буквой E.

Примеры:

```
235.988E-7=.0000235988
2359E6=2359000000
```

4. Шестнадцатеричные константы обозначаются &H. Основа счисления этих чисел – 16. Для обозначения цифр от 10 до 15 используются буквы от A до F.

Примеры:

```
&H76
&HA2F
```

5. Восьмеричные константы обозначаются &O (буква O). Основа счисления для этих чисел – 8, поэтому константы не могут содержать цифр 8 и 9.

Примеры:

```
&O347
&O177700
```

6. Двоичные константы – это двоичные числа, для обозначения которых используется &B. Эти числа содержат только цифры 1 и 0.

Примеры:

```
&B01110110
&B10101101
```

1.3.2. Типы числовых констант

По внутреннему представлению числовые константы подразделяются на целые и вещественные.

Для внутреннего представления констант целого типа используется одно слово памяти (16 бит). Это обеспечивает скорость обработки и экономию памяти для их хранения. Целый тип константы указывает знак % после десятичной константы, например:

```
156%
-3000%
```

Целый тип имеют также двоичные, восьмеричные и шестнадцатеричные константы.

Вещественные числовые константы в Бейсик-системе имеют одинарную точность. Для них хранятся 7 десятичных цифр и выделяются в памяти 2 машинных слова. По умолчанию в Бейсик-системе константы имеют вещественный тип.

Вещественная константа различается по признакам:

- любая запись числа без указания типа;
- для её записи используется экспоненциальная форма с буквой E;
- при её записи используется восклицательный знак <!>.

Примеры:

```
12343
-1.09E-09
23.567!
```

1.3.3. Текстовая константа

Формальное описание текстовой константы:

```
<ТЕКСТОВАЯ КОНСТАНТА>::=«[<СИМВОЛ>... ]255»
```

Текстовая константа – последовательность символов (цепочка знаков), заключённая в кавычки. Количество символов не должно превышать 255.

Символ пробела внутри цепочки знаков является значащим.

Примеры:

```
«ПРИВЕТ»
«ЭЛЕКТРОНИКА БК0011»
```

1.4.Переменные

Формальное определение переменной:

```
<ПЕРЕМЕННАЯ>::=<ПРОСТАЯ ПЕРЕМЕННАЯ>
               <ИНДЕКСИРОВАННАЯ ПЕРЕМЕННАЯ>

<ПРОСТАЯ ПЕРЕМЕННАЯ>::=<ИМЯ ПЕРЕМЕННОЙ>

<ИНДЕКСИРОВАННАЯ ПЕРЕМЕННАЯ>::= <ИМЯ
ПЕРЕМЕННОЙ>(<ИНДЕКС>[,<ИНДЕКС>...])
```

Переменные, идентифицирующие числовые данные, называются числовыми переменными; переменные, идентифицирующие текстовые данные, называются символьными переменными. Максимальная длина значения символьной переменной может быть 255 символов.

Переменные имеют символическое имя – идентификатор, состоящий из любого количества букв латинского алфавита и цифр; из них учитываются только две первые. Имя переменной обязательно должно начинаться буквой. Имя переменной в конце может иметь знак, определяющий тип переменной:

$	символьные;
%	целые;
!	вещественные.

При отсутствии знака считается, что переменная вещественная.

Целые переменные могут принимать значения из интервала [-32768,32767].

Примеры:

```
MINIMUM!
K%
```

```
ABC
N$
```

Различают простые и индексированные переменные. Простые переменные однозначно обозначают один элемент данных. Индексированные переменные идентифицируют элемент упорядоченного набора данных – массива. Каждый элемент массива идентифицируется именем массива и индексами, заключёнными в круглые скобки.

Если вместо круглых скобок используются квадратные, то индексированная переменная обозначает элемент виртуального массива. Память для элементов такого массива выделяется в свободных страницах ОЗУ микро-ЭВМ «Электроника БК0011» (подробнее см. оператор DIM, функцию VARPTR и руководство системного программиста 00008-01.32.01).

Индексы – это целые выражения, принимающие положительные значения. Число индексных выражений должно соответствовать числу измерений массива. Индексные выражения разделяются запятыми. Указание отрицательного индекса приводит к ошибке 5 (см. прил. 1). Для объявления массивов используется оператор DIM.

1.5.Выражения и операции

Выражение может состоять из текстовых или числовых констант и переменных или из комбинаций этих элементов, соединённых знаками операций, что приводит при выполнении к вычислению значения выражения.

Операции в Бейсик-системе делятся на:

- арифметические;
- отношения;
- логические;
- функции.

1.5.1. Арифметическое выражение

```
<АРИФМЕТИЧЕСКОЕ ::= [+]<ТЕРМ>[[+]<ТЕРМ>...]
 ВЫРАЖЕНИЕ>          [-]        [-]

                      [*]
<ТЕРМ>::=<МНОЖИТЕЛЬ>[[/]<МНОЖИТЕЛЬ>...]
                      [^]

                 <ПЕРЕМЕННАЯ>
<МНОЖИТЕЛЬ> ::= <ЧИСЛОВАЯ КОНСТАНТА>
                 (<АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ>)
                 <ФУНКЦИЯ>
```

Арифметическое выражение используется в операторах присваивания, цикла, условия, печати, определения функции. Элементами арифметического выражения могут быть переменные, функции, числовые константы или

комбинации этих элементов, соединённых круглыми скобками, знаками арифметических операций, знаками отношения и логическими операциями. В выражении могут использоваться произвольное число круглых скобок. При нарушении соответствия числа открывающих и закрывающих скобок выдаётся сообщение об ошибке 2.

Далее используется понятие целого выражения. Это арифметическое выражение, принимающее значения из интервала [-32768, 32767].

1.5.2. Арифметические операции

Для обозначения арифметических операций используются символы, приведённые в таблице 2.

Таблица 2

СИМВОЛЫ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ

Символ оператора	Обозначение	Пример
^	Возведение в степень (A в степени B)	A^B
*	Умножение (A умножить на B)	A*B
/	Деление (A делить на B)	A/B
\	Целочисленное деление (A делить на B)	A\B
MOD	Деление по модулю (остаток от деления A на B)	A MOD B
+	Сложение (A плюс B)	A+B
-	Вычитание (A минус B)	A-B
(	Открывающая скобка	
)	Закрывающая скобка	B/(A*(B+C))

При вычислении арифметических выражений соблюдается следующий приоритет выполнения арифметических операций и скобок:

1. Действия в скобках;
2. Возведение в степень;
3. Умножение и деление,
4. Целочисленное деление;
5. Деление по модулю;
6. Сложение и вычитание.

Примеры:

```
PRINT 10\4
2
OK
PRINT 10.4 MOD 4
2
OK
```

1.5.3. Операции отношения

Операции отношения используются для сравнения двух величин в операторах условного перехода. Можно считать, что результат сравнения

принимает два значения целого типа: «Истина» – во всех битах единицы, что соответствует -1 (минус 1), а «Ложь» – во всех битах нули, что соответствует 0. Как «Истина» трактуется также любое ненулевое значение.

Операции отношения приведены в таблице 3.

Таблица 3

ОПЕРАЦИИ ОТНОШЕНИЯ

Операция	Проверяемое отношение	Пример
=	Равенство	X=Y
<> ><	Неравенство	X<>Y или X><Y
<	Меньше	X<Y
>	Больше	X>Y
<=	Меньше или равно	X<=Y
>=	Больше или равно	X>=Y

Операции отношений являются менее приоритетными по отношению к арифметическим. Например, значение выражения

```
X+Y=X*Y
```

равно -1, когда X+Y равно X*Y.

1.5.4. Логические операции

Логические операции используются для работы с наборами битов, представленными в виде 16-битных целых чисел. Значение каждого бита результата зависит от значений соответствующих битов операндов.

Логические операции приведены в табл. 4.

Таблица 4

ЛОГИЧЕСКИЕ ОПЕРАЦИИ:

Операция	Аргумент		Результат
NOT	X		NOT X
	0		1
	1		0
AND	X	Y	X AND Y
	0	0	0
	0	1	0
	1	0	0
	1	1	1
OR	X	Y	X OR Y
	0	0	0
	0	1	1
	1	0	1
	1	1	1

XOR	X	Y	X XOR Y
	0	0	0
	0	1	1
	1	0	1
	1	1	0

EQV	X	Y	X EQV Y
	0	0	1
	0	1	0
	1	0	0
	1	1	1

IMP	X	Y	X IMP Y
	0	0	1
	0	1	1
	1	0	0
	1	1	1

Результат логических операций можно рассматривать как «Истину» (ненулевой результат) или «Ложь» (нуль), поэтому их удобно применять для объединения нескольких операций отношения в операторах условного перехода.

Примеры:

```
IF A=200 AND F<4 THEN 80
IF I=10 OR K>=0 THEN 50
IF NOT P THEN 100
```

В смешанных выражениях логические операции менее приоритетны, чем арифметические и операции отношений. Очерёдность выполнения логических операций осуществляется согласно следующему приоритету:

1 NOT;
2 AND;
3 OR;
4 XOR и EQV;
5 IMP.

Примеры работы логических операций:

В десятичном представлении	В двоичном представлении
63	111111
AND	
16	10000
=	
16	10000

	-1	1111111111111111
AND		
	8	1000
=		
	8	1000
	4	100
OR		
	2	10
=		
	6	110

1.5.5. Функции

Язык Бейсик позволяет использовать аппарат функций. Функции – это заранее определённые операции над данными. Различают стандартные функции (например SQR( ) – квадратный корень или SIN( ) – синус), которые заранее определены в системе, и внутренние, которые определяет сам пользователь (FN и USR). Стандартные в свою очередь подразделяются на числовые, предназначенные для работы с числовыми величинами, и символьные, предназначенные для действий над символьными строками. Более подробно стандартные функции описываются далее.

1.5.6. Преобразование типов

Преобразование типов в Бейсик-системе определяют несколько правил:

1 Если значение арифметического выражения одного типа присваивается переменной другого типа, то производится преобразование в тип переменной.

Пример:

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

2 Если во время вычисления значения арифметического выражения тип хотя бы одного операнда вещественный, то все операнды преобразуются в вещественный тип.

Пример:

```
10 D=6%*7.1
20 PRINT D
RUN
42.6
```

```
OK
```

(Арифметическое выражение было подсчитано как вещественное).

3 Логические операции преобразуют свои операнды в целый тип и дают целочисленный результат.

4 При преобразовании в целый тип отбрасывается дробная часть.

Пример:

```
10 C%=55.88
20 PRINT C%
RUN
55
OK
```

При преобразовании в целый тип значения, выходящего из пределов [-32768, 32767], происходит ошибка 6.

1.5.7. Операции над строками символов

Для работы с символьными переменными в Бейсик систему включён ряд функций (подробное описание – далее). Кроме того, символьные данные могут быть объединены операцией + (конкатенация строк).

Пример:

```
10 A$=«ИМЯ»
20 B$=« ФАЙЛА»
30 PRINT A$+B$
40 PRINT «НОВОЕ «+A$+B$
RUN
ИМЯ ФАЙЛА
НОВОЕ ИМЯ ФАЙЛА
OK
```

Строки можно сравнивать операциями отношений

```
=    <>    ><    <    >    <=    >=
```

Сравнение производится над кодами КОИ-8, взятыми из соответствующих строк. Если все коды совпадают, значит строки равны. Если коды КОИ-8 символов различаются, то меньше та строка, у которой код рассматриваемого символа меньше.

Примечание. Таблицу кодов КОИ-8 см. в руководстве системного программиста.

Если одна строка кончается, а другая – нет, то более короткая строка меньше (пробелы учитываются).

Примеры (результат всех выражений – «Истина»):

«АА» <> «АВ»

«А» = «А»

«Х&» > «Х»

«CL » > «CL»
B$ < «9.12.84.», если B$=«8.12.8»
«AAC» < «ABB»

2. РЕДАКТИРОВАНИЕ ТЕКСТОВ ПРОГРАММ

2.1.Команда LIST

Команда выводит текст программы на экран или печатающее устройство.

Формат:

```
[L]LIST [<АРГУМЕНТ1>][-[<АРГУМЕНТ2>]]
```

<АРГУМЕНТ1>::= номер первой строки выводимого фрагмента программы;
<АРГУМЕНТ2>::= номер последней строки;
L Указывает, что текст программы будет выводиться на печатающее устройство.

Команда LIST выводит текст программы на экран, а команда LLIST – на печатающее устройство. При выводе текста за номером строки идёт пробел, а далее непосредственно сам текст строки.

Если оба аргумента опущены, то выводится весь текст программы. Если задан, только один аргумент, то выводится строка с указанным номером. Тире перед номером строки (или после него) указывает, что надо выводить весь текст до (или после) строки с указанным номером, включая эту строку. Если заданы оба аргумента, то выводятся все строки с номерами из указанного интервала. Для указания номера «текущей» строки можно использовать символ «.».

Следует отметить, что эта команда не проверяет наличие в программе строк с указанными номерами, поэтому в таком случае сообщение об ошибке не выдаётся. Однако, если в программе существуют строки с номерами из заданного интервала, то они выводятся на указанное устройство. Если строк с указанными номерами нет в программе, то выполнение команды оканчивается и выводится приглашение «OK».

Для прерывания вывода используется клавиша «СТОП».

Пример:

```
200 REM ПОСЛЕДНЯЯ
150 REM СРЕДНЯЯ
100 REM ПЕРВАЯ
LIST.
100 REM ПЕРВАЯ
OK
LIST -175
100 REM ПЕРВАЯ
```

```
150 REM СРЕДНЯЯ
OK
```

2.2.Команда DELETE

Команда удаляет строки текста программы.
Формат:

```
DELETE [<АРГУМЕНТ1>][-[<АРГУМЕНТ2>]]
```

<АРГУМЕНТ1>::= номер первой строки удаляемого фрагмента программы;
<АРГУМЕНТ2>::= номер последней строки.

По этой команде удаляются строки, начиная с номера, указанного аргументом 1, до номера, указанного аргументом 2. Если аргумент 1 опущен, то вместо него берётся номер первой строки программы. Аналогично, если опущен аргумент 2, то вместо него берётся номер последней строки программы. Если в качестве аргумента 1 (или аргумента 2) задаётся «.», то берётся «текущая» строка программы. В отличие от команды LIST, если в тексте программы строка с указанным номером не существует, то выдаётся сообщение об ошибке 8.

Примечание. Эта команда может удалить всю программу или её часть, поэтому пользоваться ею надо очень осторожно.

Пример:

```
DELETE 100-200
OK
DELETE .
OK
DELETE -5000
OK
DELETE 50-.
OK
DELETE 135
OK
```

2.3.Команда RENUM

Команда перенумеровывает строки текста программы.
Формат:

```
RENUM [<АРГУМЕНТ1>][,< АРГУМЕНТ2>][,<АРГУМЕНТ3>]
```

<АРГУМЕНТ1>::= номер первой строки перенумерованного фрагмента,
<АРГУМЕНТ2>::= номер строки, с которой начинается перенумеровываемый фрагмент;
<АРГУМЕНТ3>::= шаг перенумерации, целая константа из

интервала [0,65535].

По умолчанию аргумент 1 и аргумент 3 равны 10, а аргумент 2 равен номеру первой строки программы. Вместо аргумента 1 и аргумента 2 можно использовать точку – она указывает на номер «текущей» строки.

Эта команда применяется для изменения номеров строк в программе. RENUM также проверяет, все ли строки, номера которых указаны в операторах, действительно существуют. Если обнаружено, что строки с указанным номером в программе нет, то выдаётся сообщение об ошибке 8.

Эта команда перенумеровывает строки программы, начиная со строки с номером, заданным аргументом 2. Номеру этой строки присваивается значение аргумента 1, а последующие строки нумеруются с шагом, заданным аргументом 3.

Пример:

```
RENUM 60,55
RENUM ,,5
```

2.4.Команда AUTO

Команда включает режим автоматической нумерации строк для ввода текста программы.

Формат:

```
AUTO [<АРГУМЕНТ1>][,<АРГУМЕНТ2>]
```

<АРГУМЕНТ1>::= начальный номер строки;
<АРГУМЕНТ2>::= приращение, число из интервала [0,65535].

Команда AUTO позволяет задавать автоматический ввод номеров строк при вводе текста программы, т.е. при нажатии клавиши «ВВОД» генерируется и выводится на экран следующий номер строки. Номера строк увеличиваются с шагом, равным значению второго аргумента. При отсутствии первого, второго или обоих аргументов, считается, что они равны 10.

Точка, указанная на месте первого аргумента, рассматривается как номер «текущей» строки.

Если окажется, что строка с очередным сгенерированным номером в тексте программы уже существует, этот текст выводится на экран. После этого он может быть отредактирован и введён нажатием клавиши «ВВОД». Если редакции строки не требуется, то, нажав клавишу «ВВОД», можно перейти к вводу следующей строки.

Для окончания режима AUTO достаточно нажать на клавиши «СТОП», что возвращает систему в непосредственный режим.

Пример:

```
100 PRINT «ПРИМЕР»
110 END
AUTO 100 (нажимаете клавишу «ВВОД»)
100 PRINT «ПРИМЕР» (нажимаете клавишу «ВВОД»).
```

```
110 END (нажимаете клавишу «ВВОД»)
120     (нажимаете клавишу «СТОП»)
СТОП
OK
LIST
100 PRINT «ПРИМЕР»
110 END
OK
```

2.5.Команда . (точка)

Команда используется при редактировании строк программы.

Формат:

```
. [<АРГУМЕНТ>]
```

<АРГУМЕНТ>::= номер строки программы.

Команда позволяет редактировать отдельные строки программы. Номер подлежащей редактированию строки задаётся аргументом.

После ввода программы на экран выводится заданная строка. После чего её можно редактировать при помощи клавиш редактирования (см. руководство оператора).

При ошибках, обнаруженных транслятором, «текущей» строкой становится строка, содержащая ошибку.

2.6.Клавиша (команда) «ВС»

Нажатием клавиши «ВС» можно вызвать последнюю введённую строку для дальнейшей корректировки и повторного ввода (см. руководство оператора).

2.7.Команда NEW

По этой команде удаляется программа пользователя.

Формат:

```
NEW
```

Команда используется для очистки всех переменных программ (оператор CLEAR) и удаления программы из памяти.

3. ЗАПУСК И ОТЛАДКА ПРОГРАММ

3.1.Команда RUN

Команда начинает выполнение программы.

Формат:

```
RUN [<НОМЕР СТРОКИ>]
```

Команда RUN позволяет выполнить загруженную в ОЗУ программу. По

этой команде сначала создаётся объектный код программы, производится распределение памяти, а затем объектный код запускается на выполнение. Обычно выполнение начинается с первой строки программы. Если используется аргумент, то он указывает, с какой строки программы следует начать её выполнение. При каждом новом вводе команды RUN старый объектный код и таблица имён полностью стираются и формируются заново.

Пример:

```
RUN
OK
RUN 100
OK
```

(Вторая команда начинает выполнение программы со строки с номером 100)

3.2.Команда SYSTEM

Команда осуществляет переход в монитор команд оператора драйверной системы микро-ЭВМ «Электроника БК0011».

Формат:

```
SYSTEM
```

Команда используется для передачи управления мониторной системе микро-ЭВМ.

3.3.Оператор CALL

Вызывает дополнительное программное обеспечение (например, находящееся в кассете ПЗУ).

Формат:

```
CALL[<ПАРАМЕТР>]
```

<ПАРАМЕТР>::= имя программы и параметры для неё.

Вместо слова CALL можно использовать знак подчёркивания «_». Оператор позволяет обратиться к программам, находящимся в сменной кассете ПЗУ или к программам, загруженным в свободные страницы ОЗУ (см. руководство системного программиста).

Текст, находящийся за словом CALL, Бейсик-система передаёт вызываемой программе. Это может быть имя программы, а также параметры для неё. Оператор CALL позволяет использовать расширения Бейсик-системы, управлять нестандартными устройствами, запускать игровые программы.

Пример:

```
CALL GAME
_RADIO(ON)
```

3.4.Оператор STOP

Временно останавливает выполнение программы.

Формат:

```
STOP
```

Оператор STOP используется для организации диалога при отладке программ. При выполнении оператора STOP программа останавливается, на терминале индицируется сообщение

```
СТОП В СТРОКЕ NNNNN
OK
```

и ЭВМ переходит в непосредственный режим. После приостановки программы можно распечатать и даже изменить значение переменных. Командой CONT выполнение программы может быть продолжено с оператора, следующего за оператором STOP. После редактирования текста программы попытка возобновить выполнение программы приводит к ошибке 17. В этом случае для пуска программы используется команда RUN.

Пример:

```
10 A=10
20 STOP
30 A=20
40 PRINT A
RUN
СТОП В СТРОКЕ 20
PRINT A
10
OK
CONT
20
OK
```

3.5.Команда CONT

Команда продолжает выполнение программы с того места, на котором оно было прервано.

Формат:

```
CONT
```

Выполнение программы приостанавливается всякий раз, когда встречаются операторы STOP или при нажатии клавиши «СТОП». Команда CONT продолжает выполнение программы с того места, на котором оно было прервано. На самом деле выполняется оператор GOTO по номеру «старой» строки. Если после прерывания программы производятся действия, после которых её выполнение невозможно (например, была отредактирована строка

текста программы), попытка продолжить выполнение при помощи команды CONT приводит к ошибке 17.

Команда CONT в сочетании с оператором STOP и клавишей «СТОП» широко используется во время отладки программы. После приостановки программы можно распечатать текущие значения переменных, даже изменить их и продолжить выполнение программы (см. руководство оператора).

Пример:

```
1000 A$=«ПЕРВЫЙ»
1050 STOP
1100 PRINT «ВТОРОЙ»
RUN
СТОП В СТРОКЕ 1050
OK
? A$
ПЕРВЫЙ
OK
CONT
ВТОРОЙ
OK
```

3.6.Оператор TRON

Запускает трассировку номеров строк.

Формат:

```
TRON
```

В режиме трассировки номер каждой выполняемой строки программы выдаётся на экран в формате [NNNNN]. Единственный способ остановить вывод номеров строк – это перезапустить ЭВМ или выполнить оператор TROFF. Команда RUN не отменяет действия оператора TRON.

Этот режим помогает следить за выполнением сложных и труднопонятных программ, установить, которая из ветвей программы выполняется.

3.7.Оператор TROFF

Оператор выключает включённый оператором TRON режим трассировки.

Формат:

```
TROFF
```

Пример:

```
...
249 TRON
250 ...
```

```
...
300 ...
301 TROFF
...
RUN
[250][260][270][280][290][300]
OK
```

4. ОСНОВНЫЕ ОПЕРАТОРЫ

К основным операторам относятся: операторы присваивания, безусловного и условного перехода, цикла, обращения к программе и др.

В операторах перехода используется номера строк – целые константы из интервала [0,65535]. Использование точки «.» вместо номера строки недопустимо.

4.1.Оператор LET

Это основной оператор для присвоения новых значений переменным.

Формат:

```
[LET]<АРГУМЕНТ>=<ВЫРАЖЕНИЕ>

<АРГУМЕНТ>::=<ПЕРЕМЕННАЯ>
             <ФУНКЦИЯ MID$>
```

Оператор LET служит для присвоения значения выражения, находящегося справа от знака равенства, переменной, находящейся слева от этого знака. Тип выражения должен совпадать с типом переменной.

При присваивании, если это необходимо, производится преобразование типов переменных. Попытка присвоить символьное значение числовой переменной или наоборот приведёт к ошибке 13.

Заметим, что слово LET в этом операторе не обязательно.

Пример:

```
10 LET X=1
20 Y=2*3
22 B$=«БК»
24 C$=«  -0010»
30 A$=B$+« »
40 LET D$=A$
50 MID$(C$,1%,2%)=D$
RUN
OK
? X,Y
1        6
OK
```

```
? C$
БК-0010
OK
```

4.2.Оператор GOTO

Оператор передаёт управление строке с указанным номером.

Формат:

```
GOTO <НОМЕР СТРОКИ>
```

Оператор перехода GOTO служит для изменения естественного порядка выполнения программы (т.е., выполнения в порядке возрастания номеров строк).

При выполнении оператора GOTO управление передаётся на строку с указанным номером. Если строка с указанным номером отсутствует, выдаётся сообщение об ошибке 8.

Пример:

```
10 INPUT X
20 IF X<>0 THEN GOTO 40
30 END
40 PRINT X
50 GOTO 10
```

4.3.Оператор GOSUB

Оператор передаёт управление подпрограмме.

Формат:

```
GOSUB <НОМЕР СТРОКИ>
```

Оператор GOSUB используется для вызова подпрограммы группы операторов, которая начинается с указанного номера строки и заканчивается оператором RETURN. Подпрограмма может находиться в любом месте программы и вызываться сколько угодно раз. RETURN возвратит управление оператору, следующему за вызвавшим подпрограмму оператором GOSUB.

При отсутствии строки с указанным номером выдаётся сообщение об ошибке 8. Вход в подпрограмму не через GOSUB вызывает ошибку 3.

4.4.Оператор RETURN

Оканчивает подпрограмму и осуществляет возврат к следующему за GOSUB оператору

Формат

```
RETURN [<НОМЕР СТРОКИ>]
```

RETURN осуществляет возврат к оператору, который находится непосредственно за оператором, вызвавшим подпрограмму. После возвращения все незаконченные циклы FOR-NEXT в подпрограмме

завершаются, т.е., вся информация о них «забывается», например, в программе

```
10 GOSUB 100
20 NEXT
30 END
100 FOR 1 = 1 TO 10
110 PRINT «KUKU»
120 RETURN
```

произойдёт ошибка 1 в строке 20.

В подпрограмме возможны вложенные вызовы других подпрограмм. Дополнительной возможностью оператора RETURN является использование в нем аргумента, указывающего оператор, которому должно передаться управление после завершения подпрограммы, что позволяет вернуться необязательно к месту вызова.

Пример

```
10 I = 1
20 GOSUB 100
30 I=2
40 GOSUB 100
50 END
100 REM ПОДПРОГРАММА
110 PRINT «SUB»;I
120 RETURN
```

4.5.Оператор IF

Оператор IF выполняет одну из ветвей в зависимости от условия.

Формат:

```
              THEN <ОПЕРАТОР>        <ОПЕРАТОР>
IF <АРГУМЕНТ>                  [ELSE              ]
              THEN <НОМЕР СТРОКИ>    <НОМЕР СТРОКИ>
              GOTO <НОМЕР СТРОКИ>
```

<АРГУМЕНТ>::= целое выражение, определяющее условие;
<ОПЕРАТОР>::= любой оператор Бейсика;
<НОМЕР СТРОКИ>::= номер существующей строки.

Оператор вычисляет значение аргумента и преобразовывает его в целый тип. Нулевой результат считается как «Ложь», ненулевой – «Истина».

Если условие оказывается «Истина», то выполняется указанный оператор или происходит переход к строке с указанным номером. В случае номера строки вместо слова THEN можно использовать слово GOTO.

Если аргумент принимает значение «Ложь», то выполняется ветвь ELSE или управление сразу же передаётся следующей строке, если ветвь ELSE отсутствует.

В качестве <оператора> может быть любой оператор Бейсика, даже

другой оператор IF. В качестве условия обычно используются выражения, соединённые операциями отношения и логическими операциями.

Пример:

```
10 INPUT A, B, C
20 IF A<B THEN PRINT A;«<»;B
30 I%=NOT (B<C AND A<C)
40 IF I% GOTO 10 ELSE PRINT A;«И»;B;«<»;C
```

4.6.Оператор ON

Оператор осуществляет ветвление в зависимости от значения выражения.

Формат:

```
ON <ВЫРАЖЕНИЕ> GOTO <СПИСОК>
               GOSUB
```

<ВЫРАЖЕНИЕ>::= целое выражение, принимающее значения из интервала [0,32767];

<СПИСОК>::= список номеров существующих строк, разделённых запятыми

Этот оператор позволяет осуществить переход на несколько строк. При выборе номера строки вычисляется значение выражения. Если результат равен нулю или больше, чем число номеров строк в списке, то управление передаётся следующему оператору. Если значение выражения соответствует позиции номера строки из списка (считая слева 1, 2 и т.д.), то этот номер строки употребляется оператором GOSUB или GOTO.

Если выполняется оператор GOSUB, то оператор RETURN возвращает управление оператору, следующему за оператором ON.

Этот оператор употребляется в том случае, если есть ряд задач, которые выбирают в зависимости от того, какое значение принимает выражение. Если задачи короткие и связаны со значением выражения, то употребляется ON GOTO. Если же эти задачи больше и, возможно, связаны с другим оператором ON или к ним обращаются из других мест программы, то употребляется оператор ON GOSUB, и каждую задачу нужно оформить в виде подпрограммы.

Отрицательное значение выражения приводит к ошибке 5.

Пример:

```
10 PRINT «ВВЕДИ ЛЮБОЕ ЧИСЛО»
20 INPUT A
30 ON A GOSUB 100, 200, 300
40 GOTO 10
50 END
100 PRINT «ВЕТВЬ 1»
102 RETURN
```

```
200 PRINT «ВЕТВЬ 2»
202 RETURN
300 PRINT «ВЕТВЬ 3»
302 RETURN
```

4.7.Оператор FOR

Оператор FOR вместе с оператором NEXT организует циклическое выполнение группы операторов.

Формат:

```
FOR <ПАРАМЕТР>=<АРГУМЕНТ1> TO <АРГУМЕНТ2> [STEP
<АРГУМЕНТ3>]
```

<ПАРАМЕТР>::=	числовая переменная (счётчик повторений цикла)
<АРГУМЕНТ1>::=	арифметическое выражение (начальное значение <Параметра>);
<АРГУМЕНТ2>::=	арифметическое выражение (конечное значение <Параметра>);
<АРГУМЕНТ3>::=	арифметическое выражение (приращение <Параметра>, по умолчанию равно 1).

Во время первого выполнения цикла значение аргумента 1 присваивается параметру цикла. Выполнение операторов, входящих в цикл, продолжается до появления оператора NEXT. После этого значение параметра увеличивается (или уменьшается, если значение аргумента 3 отрицательное) на значение аргумента 3 и сравнивается со значением аргумента 2. Если значение параметра больше (меньше, если аргумент 3 отрицательный) значения аргумента 2, то выполняются следующие за NEXT операторы, в противном случае повторяется выполнение операторов, находящихся между операторами FOR и NEXT.

Допустимы вложенные циклы. Они не должны пересекаться, т.е., NEXT внутреннего цикла должен появиться раньше, чем NEXT наружного цикла.

Возможен досрочный выход из цикла, при помощи операторов перехода, минуя оператор NEXT, но в этом случае стек циклов останется неочищенным, что в дальнейшем может привести к ошибке 7.

Вход в цикл, минуя оператор FOR, не допускается.

4.8.Оператор NEXT

Определяет конец операторов цикла.

Формат:

```
NEXT [<ПАРАМЕТР>[,<ПАРАМЕТР>...]]
```

<ПАРАМЕТР>::=	числовая переменная, соответствующая параметру оператора FOR.

Оператор NEXT выполняет приращение параметра цикла, проверяет

достижение предельного значения и, в зависимости от этого, осуществляет выход из цикла.

NEXT с параметром используется с целью указания, к какому FOR он относится. NEXT со списком параметров, разделённых запятыми, используется для заканчивания в одном месте нескольких вложенных циклов. В таком случае первая переменная в списке должна принадлежать самому внутреннему циклу, а последняя наружному. При заканчивании наружного цикла информация о внутренних циклах теряется, например:

```
10 FOR I=1 TO 2
20 FOR J=1 TO 10
30 PRINT J
40 NEXT I
RUN
1
1
OK
```

Тело цикла должно размещаться в одной программной секции, т. е., цикл, объявленный в главной программе, нельзя заканчивать в подпрограмме и наоборот. Вход в цикл не через оператор FOR вызывает ошибку 1.

Пример:

```
10 FOR X=1 TO 10
20 FOR Y=10 TO 1 STEP -2
30 PRINT Y
40 NEXT Y
50 PRINT TAB(3);X
60 NEXT
```

4.9.Оператор END

Заканчивает выполнение программы и закрывает файлы.

Формат:

```
END
```

Оператор прекращает выполнение программы и осуществляет возврат в непосредственный режим. END может находиться в любом месте программы. Оператор удобен для отделения главной программы от подпрограмм во избежание передачи управления подпрограмме не через оператор GOSUB.

Пример:

```
10 INPUT A$
20 GOSUB 40
30 END
40 PRINT A$
50 RETURN
```

4.10. Оператор REM

Обозначает комментарии в тексте программы.

Формат:

```
REM [<ТЕКСТ>]
'
```

<ТЕКСТ> — любая последовательность символов.

Оператор служит для включения комментария в текст программы с целью удобства чтения алгоритма, для идентификации и определения порядка работы программ. Весь текст за словом REM игнорируется при компиляции программ. Другой способ определения замечаний в тексте программы – это апостроф (« ' »). В отличие от REM, для которого необходима отдельная строка программы, апостроф может использоваться после любого другого оператора, автоматически заканчивая его.

Ввиду того, что комментарий занимает дополнительное место в оперативной памяти, уменьшается место для хранения других операторов.

Пример:

```
10 REM ПОДПРОГРАММА ВВОДА
200 X=1 'ПЕРЕМЕННАЯ ЦИКЛА
```

5. ЧИСЛОВЫЕ ФУНКЦИИ

Результаты всех числовых функций – числа с плавающей запятой.

5.1.Функция SQR

Результат функции – квадратный корень.

Формат:

```
X=SQR(<АРГУМЕНТ>)
```

<АРГУМЕНТ>::= арифметическое выражение, принимающее неотрицательные значения.

Функция вычисляет квадратный корень. Попытка обратиться к этой функции с отрицательным значением аргумента приведёт к ошибке 5.

Пример:

```
PRINT SQR(3)
1.732051
OK
```

5.2.Функция SIN

Функция вычисляет синус угла, заданного в радианах.

Формат:

```
X=SIN(<АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ>)
```

Эта функция вычисляет синус выражения. Для того, чтобы перевести градусы в радианы, можно использовать преобразование X=X*PI/180.

Пример:

```
PRINT SIN(PI/2)
1
OK
```

5.3.Функция COS

Функция вычисляет косинус угла, заданного в радианах.

Формат:

```
X=COS(<АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ>)
```

Пример:

```
PRINT COS(0)
1
OK
```

5.4.Функция TAN

Функция вычисляет тангенс угла, заданного в радианах.

Формат:

```
X=TAN(<АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ>)
```

Пример:

```
100 PRINT "РАССТОЯНИЕ=";S;"УГОЛ=";A
110 PRINT "ВЫСОТА=";S*TAN(A)
```

5.5.Функция ATN

Используется для вычисления арктангенса (в радианах).

Формат:

```
X=ATN(<АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ>)
```

Эта функция вычисляет арктангенс выражения. Результат функции указывается в радианах и находится между PI/2 и -PI/2.

Пример:

```
PRINT ATN(1)*4
3.141593
OK
```

5.6.Функция PI

Результат функции – число "пи".

Формат:

```
X=PI
```

Эта функция используется для обозначения числа "ПИ".
Примеры:

```
PRINT PI
3.141593
OK
PRINT SIN(PI/2)
1
OK
```

5.7.Функция EXP

Результат функции – **e** в указанной степени.
Формат:

```
X=EXP(<АРГУМЕНТ>)
```

<АРГУМЕНТ>::= арифметическое выражение, принимающее значения меньшие, чем 88.02969.

Функция выдаёт возведение **e** в указанную степень. Применяется главным образом для научных вычислений. Эта функция является обратной к функции LOG( ), которая вычисляет натуральные логарифмы (с основанием **e**), поэтому X=EXP(LOG(X)).

Нарушение указанных границ вызывает ошибку 6.
Пример:

```
PRINT EXP(LOG(3))
3
OK
```

5.8.Функция LOG

Результат функции – натуральный логарифм.
Формат:

```
X=LOG(<АРГУМЕНТ>)
```

<АРГУМЕНТ>::= Арифметическое выражение, принимающее положительные значения.

Функция используется для вычисления натурального логарифма. Применение функции к аргументу, имеющему отрицательное или равное нулю значение, вызывает ошибку 5.

LOG( ) можно использовать для вычисления логарифма с любым основанием. Например, для вычисления логарифмов с основанием 10 можно использовать формулу: LG(X)=LOG(X)/LOG(10).

Пример:

```
PRINT LOG(EXP(1))
1
OK
```

5.9.Функция ABS

Результат функции абсолютное значение аргумента.

Формат:

```
X=ABS(<АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ>)
```

Результатом функции является значение аргумента с положительным знаком.

Пример:

```
X=-1/3
OK
PRINT ABS(X*2)+2
2.666667
OK
```

5.10. Функция FIX

Результат функции – целая часть аргумента.

Формат:

```
X=FIX(<АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ>)
```

Функция отбрасывает дробную часть аргумента для положительных аргументов FIX( ) и INT( ) являются тождественными, но для отрицательных значений аргументов результаты этих функций различаются (см. INT( )).

Пример:

```
PRINT FIX(-5.3)
-5
OK
PRINT FIX(6.25)
6
OK
OK
```

5.11. Функция INT

Результат функции – наибольшее меньшее целое значение аргумента.

Формат:

```
X=INT(<АРГУМЕНТ>)
<АРГУМЕНТ>::=<АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ>
```

Функция выдаёт ближайшее целое число, не превосходящее аргумента. Функция INT( ) похожа на FIX( ). Но, в отличие от INT( ), FIX( ) просто отбрасывает дробную часть аргумента. Различия между функциями проявляются при отрицательных аргументах. Для описываемой функции выражения X=ABS(INT(Z)) и X=ABS(INT(-Z)) для реальных Z дадут разные результаты.

Пример:

```
PRINT INT(-5.3), FIX(-5.3)
-6               -5
OK
```

5.12. Функция SGN

Результат функции равен -1, 0 или 1, в зависимости от знака аргумента.
Формат:

```
X=SGN(<АРГУМЕНТ>)
<АРГУМЕНТ>::=<АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ>
```

Функцию SGN( ) называют функцией знака. Эта функция даёт следующие результаты:

- если аргумент < 0, результат -1;
- если аргумент = 0, результат 0;
- если аргумент > 0, результат +1.

Функцию можно использовать, например, когда необходимо без оператора IF присвоить переменной число 0 только тогда, когда другая переменная имеет значение 0.

Пример:

```
...
110 C=B*SGN(A)
...
```

5.13. Функция RND

Результат функции RND – случайное число из интервала [0,1].
Формат:

```
X=RND(<АРГУМЕНТ>)
<АРГУМЕНТ>::=<АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ>
```

Функция используется для получения псевдослучайных чисел, имеющих равномерное распределение на интервале [0,1]. Её удобно использовать при создании игровых программ.

Выдаваемые функцией значения зависят только от знака аргумента:

- если значение аргумента положительно, то результат RND( ) – очередное псевдослучайное число из последовательности псевдослучайных чисел. Сама последовательность зависит от начального значения;

- если значение аргумента отрицательно, то изменяется начальное значение последовательности псевдослучайных чисел. Каждому отрицательному аргументу соответствует одна последовательность, причём она не зависит от порядка аргумента;

- если используется нулевое значение аргумента, то результат функции тот же, что и в предшествовавшем обращении к ней.

```
PRINT RND(1), RND(-5),RND(-50)
.4709943
.5642278
.5642278
OK
PRINT RND(3),RND(0)
.3861489
.3861489
OK
```

6. СИМВОЛЬНЫЕ ФУНКЦИИ

6.1.Функция LEN

Результат функции – длина символьной строки.
Формат:

```
X=LEN(<СИМВОЛЬНОЕ ВЫРАЖЕНИЕ>)
```

Функция LEN( ) подсчитывает количество символов строки и может дать любое число от 0 (если строка пустая) до 255 (если строка наибольшей допустимой длины). Эта функция позволяет узнать длину переменной до её обработки. Применение некоторых функций для "пустых" символьных переменных может привести к ошибке, а использование функции LEN позволяет избежать этого, например:

```
...
200 IF LEN(A$)>0 THEN T$=MID$(A$,1,1)
...
```

Пример:

```
...
100 IF LEN(S$)>32-POS THEN PRINT CHR$(10)
110 PRINT S$
...
```

6.2.Функция INSTR

Выдаёт номер позиции вхождения подстроки в строку.
Формат:

```
X=INSTR([<АРГ1>,]<АРГ2>,<АРГ3>)
```

<АРГ1>::= целое выражение, принимающее значения из интервала [1,255]; означает начальную позицию поиска;
<АРГ2>::= символьное выражение, означающее строку, в которой ведётся поиск;
<АРГ3>::= символьное выражение, означающее подстроку.

Функция выдаёт номер позиции, с которой все символы подстроки и строки, заданной аргументом 2, совпадают. Начало поиска задаёт аргумент 1, при его отсутствии поиск ведётся с первого байта. Если фрагмент не найден, то функция получает значение 0.

Пример:

```
PRINT INSTR(3,«ABCDEABCD»,«AB»)
6
ok
```

6.3.Функция MID$

Функция выбирает или заменяет часть символьной переменной.

Формат:

```
X$=MID$(<АРГ1>,<АРГ2>[,<АРГ3>])
```

или

```
MID$(<АРГ4>,<АРГ2>[,<АРГ3>])=<СИМВОЛЬНОЕ ВЫРАЖЕНИЕ>
```

<АРГ1>::= символьное выражение;

<АРГ2>::= целое выражение, принимающее значения от 1 до 255 и определяющее номер символа, с которого начинается используемая в операции часть символьной строки;

<АРГ3>::= целое выражение, принимающее неотрицательные значения и определяющее количество символов, участвующих в операции;

<АРГ4>::= символьная переменная.

Функция MID$ позволяет выбрать часть символьной переменной. Это единственная функция, которая может встречаться по обе стороны знака равенства.

В первом варианте MID$ используется для выбора подстроки из результата символьного выражения, заданного аргументом 1, начиная с символа, указанного вторым аргументом, причём выбирается столько символов, сколько задано третьим аргументом.

Значение второго аргумента не должно превышать длину символьного выражения, указанного первым аргументом. Если третий аргумент не указан или выходит за границы строки, то выделяется часть символьной строки от символа, указанного вторым аргументом, до конца.

Заметим, что выделенная функцией MID$ подстрока помещается в области памяти, отведённой под символьные переменные.

Во втором варианте, когда функция MID$( ) находится с левой стороны знака равенства, дополнительная память из области, отведённой под символьные переменные, не используется. В этом случае обрабатываемые символьные данные присваиваются символьной переменной, значение которой уже находится в указанной области. В этом случае аргументы 2 и 3

указывают изменяемое место в символьной переменной и должны соответствовать аналогичным требованиям, как и в первом случае.

```
100 A$="ФАЙЛ ПЕРВЫЙ"
110 A1$=STRING$(20," ")
120 MID$(A1$,1)=MID$(A$,6)
130 MID$(A1$,7)=MID$(A$,1,4)
140 PRINT A1$
RUN
ПЕРВЫЙ ФАЙЛ
OK
```

6.4.Функция STRING$

Функция инициализирует символьную переменную.

Формат:

```
X$=STRING$(<АРГУМЕНТ1>,<АРГУМЕНТ2>)
                        <АРГУМЕНТ3>
```

<АРГУМЕНТ1>::= целое выражение, принимающее значения от 0 до 255 и определяющее длину создаваемой символьной строки;

<АРГУМЕНТ2>::= целое выражение, принимающее значения 0 до 255, соответствующие коду КОИ-8, которым заполняется новая символьная строка;

<АРГУМЕНТ3>::= символьное выражение, первым символом которого заполняется новая символьная строка.

Предназначением функции STRING$( ) является создание символьной переменной любой длины, содержащей одинаковые символы. Вы можете определить этот символ либо задав его по таблице кодов КОИ-8, либо указав нужный символ с помощью символьного выражения, в частности текстовой константы. Тип второго аргумента функции STRING$ указывает, каким путём определяется символ, которым инициализируется символьная переменная. Например, выражения STRING$(41,32) и STRING$(41," ") приводят к одинаковому результату.

Будет зафиксирована ошибка 5, если один из числовых параметров выйдет за пределы интервала [0,255]. Если заданная длина символьной строки равна 0, то будет создана пустая строка, несмотря на инициализирующий символ.

Пример:

```
...
300 PRINT NAZV$;STRING$(50-LEN(NAZV$),".");N
...
```

7. ФУНКЦИИ ПРЕОБРАЗОВАНИЯ ТИПОВ

7.1.Функция CINT

Функция преобразует тип значения арифметического выражения в целый

Формат:

```
X=CINT(<АРГУМЕНТ>)
```

<АРГУМЕНТ>::= арифметическое выражение, принимающее значения из интервала [-32768, 32767]

Функция отбрасывает дробную часть арифметического выражения также, как это делает функция FIX( ). Если значение выходит за пределы 16-битного целого числа [-32768, 32767], то выдаётся ошибка 6. Функция используется по умолчанию, когда значение арифметического выражения присваивается переменной целого типа.

Примеры:

```
PRINT CINT(5.93), CINT(-6.352)
5                 -6
OK
```

7.2.Функция CSNG

Функция преобразует результат арифметического выражения в вещественный тип.

Формат:

```
X=CSNG(<АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ>)
```

Функция используется для преобразования значения арифметического выражения в вещественное число одинарной точности. Она используется по умолчанию, когда производится присвоение значения переменной одинарной точности.

Примеры:

```
PRINT CSNG(5%)
5
OK
```

7.3.Функция ASC

Функция переводит один байт строки символов в код КОИ-8.

Формат:

```
X=ASC(<АРГУМЕНТ>)
```

<АРГУМЕНТ>::= непустое символьное выражение.

Эта функция обрабатывает первый символ строки символов. Значение этой функции выдаётся как целое число, которое является кодом

соответствующего символа по таблице КОИ-8, поэтому результат функции принадлежит интервалу [0,255]. Обратной к функции ASC( ) является функция CHR$( ). Для обработки последующих символов символьной строки можно применять функцию MID$.

Если аргумент – пустая строка, то вызов функции приводит к ошибке 5.

Пример:

```
ZZ$="ABC"
OK
PRINT ASC(ZZ$)
65
OK
PRINT ASC(MID$(ZZ$,2))
66
OK
```

7.4.Функция CHR$

Функция переводит целое число (код КОИ-8) в символьную строку из одного символа.

Формат:

```
X$=CHR$(<АРГУМЕНТ>)
```

<АРГУМЕНТ>::= целое выражение, принимающее значения из интервала [0, 255].

При вызове функции значение аргумента трактуется как код из таблицы КОИ-8 и преобразовывается в символьную строку из одного символа, соответствующего этому коду. Таким образом, CHR$(65) и "A" тождественны.

Пример:

```
PRINT CHR$(65)
A
OK
```

Функция CHR$( ) является обратной к функции ASC( ). Если X попадает в необходимый диапазон, и переменная X$ состоит из одного символа, то всегда верно:

```
X=ASC(CHR$(X)) и X$=CHR$(ASC(X$)).
```

Если значение аргумента выходит за пределы указанного интервала, то фиксируется ошибка 5.

Рассматриваемую функцию можно использовать и при работе с многобайтными строками, например:

```
10 OTVET$=" "
20 FOR I=1 TO 3
30 PRINT "УКАЖИТЕ НОМЕР";I;" ОТВЕТА"
```

```
40 INPUT NOMER
50 OTVET$=OTVET$+CHR$(NOMER+49)
60 NEXT
70 PRINT "ОТВЕТЫ="; OTVET$
```

Для многобайтных строк также можно использовать функцию MID$. Заметим, что существуют символы, которые можно занести в строку только при помощи функции CHR$, например:

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

(Выдаётся звуковой сигнал).

7.5.Функция VAL

Результат функции – числовой эквивалент части символьной переменной, содержащей цифры.

Формат:

```
X=VAL(<СИМВОЛЬНОЕ ВЫРАЖЕНИЕ>)
```

Функция применяется к символьному выражению, которое предположительно содержит действительное число, записанное в кодах КОИ-8. Это число может включать знак, десятичную точку и обозначение порядка. Результат символьного выражения просматривается слева направо до недопустимого для чисел кода. Если первый символ строки нельзя отнести к числу, то результат будет нулевой. Например, результат VAL("A12:") равен 0, а VAL("23A12:") – число 23.

Функция VAL является обратной к функции STR$( ).

Пример:

```
PRINT VAL(MID$ ("A12:",2))
12
OK
```

7.6.Функция STR$

Превращает числовые данные в строку символов в КОИ-8 коде.

Формат:

```
X$=STR$(<АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ>)
```

Эта функция переводит числа в строку символов аналогично оператору PRINT. Численное выражение вычисляется как обычно, а результат "печатается" в новую временную строку в области строк, что и является результатом функции STR$. (не спутайте STR$( ) с функцией STRING$( )!). Например, если N=1/3, то STR$(N) даст строку ".3333333" и каждое число "3" в самом деле есть символ, определённый как "3" (восьмеричный код 63).

Эта функция является обратной к функции VAL. Например, результат VAL(STR$(N)) будет N, а STR$(VAL(NM$)) будет строка NM$, если NM$

содержит действительное число.

Функция удобна для форматирования числовых данных перед выводом на экран, для вкомпоновки чисел в текст. От оператора PRINT функция отличается тем, что не выдаёт пробела в конце строки.

Пример:

```
A=123
OK
PRINT MID$(STR$(A),LEN(STR$(A))-1)
23
OK
```

7.7.Функция BIN$

Превращает целые числа в символы двоичного формата.

Формат:

```
X$=BIN$(<ЦЕЛОЕ ВЫРАЖЕНИЕ>)
```

Результат выражения переводится в целый тип и превращается в символьную строку, представляющую двоичный его код. Например, аргумент со значением 3 даст результат "11", т.е., строку из двух символов "1" (восьмеричный код 61). Знака или пробелов функция не генерирует, незначащие нули перед числом не обрабатываются. Максимальная длина строки результата функции BIN$( ) – 16 байтов. Главным образом BIN$( ) используется для печати двоичного представления чисел.

Пример:

```
PRINT BIN$(44)
101100
OK
```

7.8.Функция OCT$

Превращает целое число в символьную строку восьмеричного представления.

Формат:

```
X$=OCT$(<ЦЕЛОЕ ВЫРАЖЕНИЕ>)
```

Значение аргумента переводится в целый тип, полученный результат превращается в строку символов в соответствии с его значением в восьмеричной системе счисления аналогично функции BIN$. Пробелов не создаётся, незначащие нули отбрасываются. Таким образом, аргумент 12 даёт результат "14", т.е., строку из двух символов "1" и "4" (восьмеричные коды КОИ-8 61 и 64 соответственно). Максимальная длина строки, полученной при помощи OCT$, – шесть символов.

Функция используется для печати восьмеричного представления чисел.

Пример:

```
PRINT OCT$(&HFFFF)
177777
OK
```

7.9.Функция HEX$

Превращает числовой аргумент в символьную строку шестнадцатеричного представления.

Формат:

```
X$=HEX$(<ЦЕЛОЕ ВЫРАЖЕНИЕ>)
```

Функция HEX$ переводит значение аргумента в целый тип и результат превращает в символьную строку, символы которой представляют значение аргумента в шестнадцатеричном формате. Знак и пробелы не генерируются, незначащие нули отбрасываются. Это значит, что аргумент 43 даёт результат "2B", другими словами, строку из двух символов с восьмеричными кодами КОИ-8 62 и 102. Из этого следует, что максимальная длина значения функции HEX$( ) – 4 байта.

Функция используется для распечатки шестнадцатеричного представления числа.

Пример:

```
PRINT HEX$(&H1A)
1A
OK
```

8. СЛУЖЕБНЫЕ ОПЕРАТОРЫ И ФУНКЦИИ

В этом разделе описываются операторы резервирования памяти, инициализации переменных, а также функции, определенные пользователем.

Операторы резервирования памяти служат резервированию оперативной памяти для переменных, массивов и постоянных величин. Память для простых переменных резервируется в таблице имён, для массивов – в свободной области пользователя (см. руководство системного программиста). Память для виртуальных массивов резервируется в свободных страницах ОЗУ. Постоянные величины располагаются в области объектного кода, генерируемого при компиляции программы. Для резервирования памяти переменным и массивам используется оператор DIM, для резервирования памяти постоянным величинам — оператор DATA. Для присвоения переменным значений постоянных величин используются операторы READ и RESTORE.

8.1.Оператор DIM

Резервирует память для переменных и массивов, придавая им нулевые значения.

Пример:

```
DIM <АРГУМЕНТ>[,<АРГУМЕНТ>...]
```

<АРГУМЕНТ>::= допустимое имя переменной, за которым может следовать заключённый в скобки список максимальных значений индексов массива.

Этот оператор обычно используется для определения массивов, но может определять и отдельные переменные. Каждый массив имеет заданное в списке число индексов. Максимальное значение индекса указывается целым выражением, принимающим значение из интервала [0,255].

Если список максимальных значений индексов заключается не в круглые, а в квадратные скобки, то определяется виртуальный массив. Память для такого массива выделяется не в общей свободной памяти, а в скрытых страницах ОЗУ. Использование виртуальных массивов позволяет хранить большое количество информации, но скорость выборки элемента виртуального массива меньше, чем обычного массива. Одновременно могут существовать виртуальный и обычный массивы с одинаковыми именами. Для увеличения размера свободной памяти виртуальных массивов можно использовать оператор SCREEN.

Возможно неявное определение переменных и массивов. Такое определение происходит каждый раз при появлении имени неопределённой переменной. Простые переменные обычно определяются именно таким способом. При появлении имени индексированной переменной определяется одномерный массив с максимальным значением индекса 10.

Неявное определение массива невозможно в непосредственном режиме и вызывает ошибку 12. Попытка повторного определения уже объявленного или определённого по умолчанию массива приводит к ошибке 10. Если не хватает оперативной памяти для определения массива, выдаётся сообщение об ошибке 7.

Пример:

```
10 DIM X,I%,Z(5,20,3),A$(30),A%[8000%]
```

8.2.Оператор DATA

Сохраняет заданные в тексте программы константы для последующего пользования.

Формат:

```
DATA <ЭЛЕМЕНТ>[,<ЭЛЕМЕНТ>...]
```

<ЭЛЕМЕНТ>::= числовая константа, текстовая константа в кавычках или без кавычек, но не включающая запятых.

Этот оператор позволяет сохранять «начальные» величины в теле программы. В режиме непосредственного выполнения оператор DATA игнорируется. Во время выполнения программы он тоже опускается, и только оператор READ может использовать его.

Во время чтения данные выбираются, начиная с оператора DATA с наименьшим номером строки, а следующие операторы выбираются последовательно. Когда все данные исчерпаны, дальнейшее чтение невозможно до появления оператора RESTORE.

В списке оператора DATA запятая используется как разделитель. Апостроф не завершает списка и не означает начала комментария. Разрешены любые числовые константы, включая &H, &O, &B, экспонентную форму, константы с явно указанным типом. Если в текстовой константе нет разделителя, то кавычки могут быть опущены.

8.3.Оператор READ

Вводит данные из оператора DATA в программу.

Формат:

```
READ <ПЕРЕМЕННАЯ>[,<ПЕРЕМЕННАЯ>...]
```

<ПЕРЕМЕННАЯ>::= допустимое имя переменной (включая элемент массива).

Этот оператор читает следующие ещё непрочитанные данные из оператора DATA. Процесс повторяется до тех пор, пока все переменные из списка заберут свои значения. Оператор READ просматривает все записи каждого оператора DATA, анализируя их с начала программы до конца. Для чтения данных несколько раз можно воспользоваться оператором RESTORE.

В случае обнаружения конца операторов DATA выдаётся сообщение об ошибке 4. При попытке читать текстовые данные в числовые переменные получается ошибка 13.

Пример:

```
10 DATA 1,2,3,4,5,6
20 FOR I%=0% TO 5%
30 READ X
40 A(I%)=X
50 PRINT "I=";I%,A(I%)
60 NEXT I%
RUN
I=0           1
I=1           2
I=2           3
I=3           4
I=4           5
I=5           6
OK
```

8.4.Оператор RESTORE

Изменяет порядок чтения данных из оператора DATA оператором READ.

Формат:

```
RESTORE [<НОМЕР СТРОКИ>]
```

<НОМЕР СТРОКИ>::= номер существующей строки.

Этот оператор изменяет значение указателя, определяющего, который из операторов DATA будет использован следующим в операторе READ. READ перемещает указатель последовательно, пока все операторы DATA не будут использованы. Оператор RESTORE возвращает указатель в начало программы и все данные заново могут быть введены. В случае присутствия номера строки будут рассматриваться операторы DATA с указанным или большими номерами строк. Повторно использовать оператор RESTORE можно в любом месте программы. Он позволяет смотреть на операторы DATA как на устройство ввода данных с возможностью повторного чтения.

Пример:

```
10 DATA 0,1,2,3,4,5,6,7,8,9,10
20 GOSUB 100
30 END
100 RESTORE
105 PRINT «ПОДПРОГРАММА»
110 FOR I%=0% TO 10%
120 READ X%
130 PRINT X%
140 NEXT I%
150 RETURN
RUN
ПОДПРОГРАММА
1
2
3
4
5
6
7
8
9
10
```

8.5.Оператор CLEAR

Очищает значения переменных, резервирует символьную и не бейсиковую память.

Формат:

```
CLEAR [<АРГУМЕНТ1>[,<АРГУМЕНТ2>]]
```

<АРГУМЕНТ1>::= целое выражение, указывающее число байтов

области памяти, резервируемой для символьных переменных;

<АРГУМЕНТ2>::= целое выражение, указывающее адрес верхней границы области оперативной памяти, используемой Бейсик-системой не должен превышать восьмеричного 100000.

При любой форме оператора очищаются массивы и переменные, определённые функций FN, символьные строки, а также аппаратный стек, закрываются все файлы, а любая распечатка заканчивается символом <ПС>, определения функций USR остаются неизменными.

Первый аргумент используется для установления размера области памяти для хранения текста строк, по умолчанию он равен 200 байтов. Если задан аргумент 1, то для символьных строк резервируется указанное число байтов.

Второй аргумент используется для указания начала области, которая не может использоваться для программ и данных Бейсика. Таким образом определяется область для неформатированных данных, функций USR или для другого использования. После выполнения CLEAR с двумя параметрами в косвенном режиме объектный код теряется, и Бейсик-система переходит в непосредственный режим.

Многие операции делают CLEAR автоматически, например, RUN, NEW, MERGE, LOAD и любое изменение текста программы Бейсика. Если параметры CLEAR не указаны, то соответствующие значения величины области строк и верхней границы Бейсик-системы не изменяются.

Слишком большое значение первого или слишком малое второго аргумента вызывает недостаток оперативной памяти, и выдаётся сообщение об ошибке 7. Если аргумент 2 больше допустимого значения, выдаётся сообщение об ошибке 5.

Пример:

```
CLEAR 1000,&O30000
OK
```

(Резервируется 1000 байтов для строк и ОЗУ свыше &O30000 для пользовательских целей).

8.6.Оператор DEF

8.6.1. Оператор DEF USR

Определяет функции на машинном языке.

Формат:

```
DEF USR[<ЦИФРА>]=[<АРГ1>,]<АРГ2>
```

<ЦИФРА>::= целая константа из интервала [0,9], указывающая номер функции, по умолчанию

0;

<АРГ1>::= целое выражение, определяющее номер страницы памяти;

< АРГ2>::= целое выражение, определяющее адрес запуска функции USR.

Оператор определяет адрес запуска машинной подпрограммы, составленной пользователем. Можно также указать страницу памяти, в которой находится эта подпрограмма. Значения выражений переводится в целый тип запуска программы.

Аргумент 1 означает номер страницы памяти, в которой находится машинная подпрограмма (см. руководство системного программиста). Если этот аргумент опущен, то при вызове подпрограммы остаётся ранее подключённая страница. В этом случае запятая вместо пропущенного аргумента не ставится.

Аргумент 2 означает адрес запуска программы. Отрицательные значения от -32768 до -1 означают адреса 32768 до 65535 соответственно. Для указания адресов удобно пользоваться восьмеричными константами.

Одновременно могут быть определены 10 функций USR. Номер определяемой функции указывается цифрой.

Для выполнения программы на машинном языке необходимо выполнить, например, такую последовательность операторов

```
10 DEF USR=&O70000
20 A=USR(A)
```

Обычным способом загрузки машинной программы в память является использование команды BLOAD.

Для выделения памяти функциям USR( ) используется оператор CLEAR. Этот оператор сохраняет все определения DEF USR.

8.6.2. Оператор DEF FN

Определяет пользовательские функции.

Формат:

```
DEF FN <ИМЯ>[(<СПИСОК>)]=<ВЫРАЖЕНИЕ>
```

<ИМЯ>::= любое допустимое имя переменной, означающее имя определяемой функции;

<СПИСОК>::= одно или несколько имён формальных параметров (допустимых имён переменных), разделённых запятыми;

<ВЫРАЖЕНИЕ>::= любое выражение такого же типа, как и имя функции.

При выполнении этого оператора создаётся переменная, имя которой отмечено как имя функции. Никаких вычислений оператор не производит и может быть в любом месте программы. Эта переменная не доступна обычным

путём, и может быть создана нормальная переменная с таким же самым именем. В случае создания другой функции с таким же самым именем старое определение теряется.

Функция может иметь любое число параметров. Они определяются как список имён переменных любого типа, разделённых запятыми и заключённых в скобки. Параметры являются формальными и реально не существуют, т.е., любая обычная переменная может иметь такое же имя. Каждый аргумент должен быть такого типа, что ему могло бы присвоиться значение фактического аргумента, задаваемого при вызове функции FN. Выражение за знаком равенства вычисляется при вызове функции, и результат возвращается в место вызова функции. Если в этом выражении есть имена переменных, объявленных формальными аргументами, то их значения берутся соответственно списку фактических аргументов в вызове функции. В выражении также можно использовать любые другие переменные, если их имена не совпадают с именами формальных аргументов. В выражении могут быть любые обращения к функциям, также и нерекурсивные обращения к функциям FN.

Ошибки проверяются только при вызове функции. При обнаружении ошибки в строке, вызывающей пользовательскую функцию, следует проверить определение этой функции.

Этот оператор даёт единственную возможность изменять ход программы, не учитывая номера строки. DEF FN, в отличие от DATA, является выполняемым оператором, т.е., функция должна определяться раньше её вызова. Выполнение этого оператора возможно только в косвенном режиме. В непосредственном режиме выдаётся сообщение об ошибке 12.

Пример:

```
10 DEF FN QU$(QU,ST)=MID$(QL$(QU),ST)
20 PRINT FN QU$(CO,LN(CO))
...
```

8.7.Функции, определённые пользователем

8.7.1. Функции FN

Обозначают место вызова пользовательских функций.

Формат:

```
X=FN <ИМЯ>[(<СПИСОК>)]
```

<ИМЯ>::= имя функции, определённой оператором DEF FN;
<СПИСОК>::= одно или несколько выражений, разделённых запятыми.

Функции FN – это функции Бейсик-системы, определённые пользователем. Функция должна быть определена при помощи оператора DEF FN раньше её вызова. Выражения в списке являются фактическими

параметрами и должны по числу, расположению в списке и типу соответствовать списку формальных параметров в операторе DEF FN. Значения фактических параметров вставляются в выражение в определении функции, и значение этого выражения считается результатом функции.

При появлении ошибок в вызове функции следует проверить её определение, так как в операторе DEF FN ошибки не проверяются.

Пример:

```
...
10 DEF FN LG(X)=LOG(X)/LOG(10)
20 PRINT FN LG(100)
...
```

8.7.2. Функции USR

Позволяет обращаться к подпрограммам на машинном языке.

Формат:

```
X=USR[<ЦИФРА>](<АРГУМЕНТ>)
```

<ЦИФРА>::= целая константа из интервала [0,9], указывающая номер вызываемой функции; по умолчанию 0;

<АРГУМЕНТ>::= выражение, значение которого передаётся функции USR.

Перед обращением к функции необходимо определить её при помощи оператора DEF USR. Одновременно могут быть определены 10 различных функций USR с номерами от 0 до 9.

Вызов функции происходит при помощи машинной инструкции JSR PC. Подпрограмма может находиться в любом месте допустимого адресного пространства, включая также подпрограммы самого Бейсика. Возможны непосредственная передача подпрограмме одного аргумента любого типа и возврат одного значения обязательно того же типа. После передачи управления подпрограмме в регистре общего назначения R5 находится адрес, а в R3 – тип аргумента. Тип закодирован следующим образом: единица в 15-ом разряде означает символьную строку, информация о других типах хранится в младшем байте и имеет значения:

1 – целый аргумент,

0 – число с плавающей запятой,

-1 – символьный аргумент.

В случае символьной строки в качестве значения аргумента передаются два слова, в первом из которых хранится длина, в другом – начальный адрес строки.

Вернуть управление вызывающей программе возможно при помощи инструкции RTS PC, предварительно поместив результат функции в ячейки, указываемые R5. Если содержимое этих ячеек не будет изменяться, то

результатом останется значение аргумента. В подпрограмме USR все регистры свободны для использования.

Использование функций USR позволяет расширить возможности Бейсик-системы, выполнить невозможные или медленно выполняемые в ней действия.

Вызов неопределённой функции USR приводит к ошибке 5.

Пример:

```
10 PRINT USR(0%)
20 A%=USR2(A%)
```

8.8.Оператор KEY

Этот оператор изменяет значения функциональных клавиш.

Формат:

```
KEY <АРГУМЕНТ1>,<АРГУМЕНТ2>
```

<АРГУМЕНТ1>::= целое выражение, определяющее номер функциональной клавиши, число из интервала [1,10];

<АРГУМЕНТ2>::= символьное выражение, определяющее значение ключа (используются только первые 63 символа).

Оператор используется для изменения значений функциональных клавиш, установленных по умолчанию. Первые 63 символа аргумента 2 присваиваются функциональной клавише, указанной аргументом 1.

В служебной строке экрана высвечиваются первые символы значений функциональных клавиш. Оператор KEY производит соответствующие изменения служебной строки.

Для ввода управляющих символов или кавычек можно воспользоваться функцией CHR$.

Выход значения первого аргумента за пределы допустимого интервала вызывает ошибку 5.

По умолчанию функциональные клавиши имеют следующие значения:

```
1 - COLOR
2 - AUTO
3 - GOTO
4 - LIST
5 - RUN <ПС>
6 - COLOR 4,0 <ПС>
7 - CLOAD "
8 - CONT <ПС>
9 - .<ПС>
10 - <СБР>RUN<ПС>
```

(Управляющий символ <СБР> имеет восьмеричный код 14 и означает

очистку экрана). Код <ПС> получается при нажатии клавиши «ВВОД» и имеет восьмеричный код 12. Использование <ПС> позволяет немедленно выполнить соответствующий оператор.

Пример:

```
KEY 1,"LINE"
KEY 2,"SAVE"+CHR$(34)
KEY 3,"PRINT PI"+CHR$(10)
```

9. УПРАВЛЕНИЕ ДОПОЛНИТЕЛЬНЫМИ ВОЗМОЖНОСТЯМИ ЭВМ

9.1.Операторы управления экраном телевизора

9.1.1. Оператор CLS

Очищает экран.

Формат:

```
CLS
```

Оператор CLS окрашивает весь экран в текущий цвет фона. Используя его, можно наиболее быстрым путём получить сплошной экран любого цвета.

9.1.2. Оператор SCREEN

Переключает экраны.

Формат:

```
                      SHOW <АРГУМЕНТ2>
SCREEN <АРГУМЕНТ1>[ ON             ]
                      OFF
```

<АРГУМЕНТ1>, <АРГУМЕНТ2>::= целые выражения, означающие номера экранов, числа, принимающие значения 0 или 1.

В микро-ЭВМ «ЭЛЕКТРОНИКА БК0011» имеются два экранных ОЗУ: страницы 5 и 6 (см. руководство системного программиста). Они являются экранами 0 и 1 соответственно. Для вывода информации и отображения на экране телевизионного приёмника могут независимо друг от друга подключаться обе страницы. Оператор SCREEN управляет подключением страниц экранов. Задание только первого аргумента обеспечивает вывод информации и отображение на экране телевизионного приёмника страницы, указанной этим аргументом. Если заданы оба аргумента, то информация (текстовая или графическая) будет выводиться на экран, указанный первым аргументом, а отображаться будет экран, указанный вторым аргументом.

Комбинации SCREEN OFF и SCREEN ON используются для объявления страниц экранов недоступными для вывода информации (или отмены этого режима). Освободившуюся страницу ОЗУ можно использовать,

например, для увеличения области виртуальных массивов (см. оператор DIM, функцию VARPTR). Оператор SCREEN ON будет действовать лишь в том случае, если раньше был использован оператор SCREEN OFF, а область виртуальных массивов ещё не достигла страницы экрана.

Пример:

```
...
100 SCREEN 1 SHOW 0
...
200 SCREEN 0
...
300 SCREEN 1 OFF
310 DIM A%[16000]
...
```

9.1.3. Оператор COLOR

Устанавливает цвет экрана.

Формат:

```
COLOR [<АРГУМЕНТ1>][,<АРГУМЕНТ2>]
```

<АРГУМЕНТ1>::= целое выражение, указывающее номер цвета переднего плана, число в пределах [0,4];

<АРГУМЕНТ2>::= целое выражение, указывающее номер цвета фона, число в пределах [0,4].

Оператор устанавливает текущие цвета переднего плана и фона экрана. При отсутствии параметра соответствующий цвет не меняется.

Имеются следующие номера цветов:

0 – прозрачный (цвет фона);

1 – чёрный;

2 – синий;

3 – зелёный;

4 – красный.

Номер 0 для фона означает чёрный цвет.

Оператор COLOR изменяет цвет выводимого текста, все операторы графики также по умолчанию используют цвет, установленный этим оператором.

Если передний и задний планы случайно оказались одного цвета, то создаётся впечатление, что экран пустой, а компьютер не реагирует на клавиатуру. Для приведения микро-ЭВМ в готовность после остановки программы клавишей "СТОП" необходимо нажать ключ 6, восстанавливающий стандартные цвета.

При отсутствии обоих параметров выдаётся сообщение об ошибке 24. В случае нарушения допустимых границ выдаётся сообщение об ошибке 5.

Пример:

```
...
100 COLOR 2,3
110 DRAW A$
120 COLOR 1
130 PSET STEP(20,0)
140 DRAW A$
...
```

9.1.4. Оператор LOCATE

Передвигает курсор на экране, высвечивая и погашая его.

Формат:

```
LOCATE [<АРГ1>][,<АРГ2>][,<АРГ3>]
```

<АРГ1>::= позиция X (столбец), на котором должен быть расположен курсор, целое выражение со значением из интервала [0,255];

<АРГ2>::= позиция Y (строка), на которой должен быть расположен курсор, целое выражение со значением из интервала [0,255];

<АРГ3>::= целое выражение, указывающее, гаситься (нулевое значение) или высвечиваться (ненулевое значение) должен курсор;

Оператор LOCATE контролирует местоположение курсора на экране. Строки имеют номера от 0 (верхняя) до 23 (нижняя). Столбцы нумеруются от 0 до 31 слева направо. Курсор передвигается на позицию, указанную в операторе LOCATE. Если опущен любой из первых двух параметров, то сохраняется предыдущее его значение.

Если присутствует аргумент 3 и он не равен 0, то оператор высвечивает курсор, который обычно гасится во время выполнения программы. Курсор высвечивается также оператором INPUT. Для ввода данных с погашенным курсором необходимо использовать функцию INKEY$.

Функции CSRLIN и POS позволяют узнать положение курсора, установленное этим оператором. Если номер столбца или строки больше числа возможных на экране столбцов или строк, но не больше 255, то курсор будет перенесён через максимальное число позиций в заданном направлении.

Отрицательные или большие, чем 255 значения первых двух параметров приводят к ошибке 5.

Пример:

```
...
100 LOCATE 0,22
110 INPUT "ЧИСЛО";X
120 LOCATE 0,0,0
130 PRINT X
...
```

9.2.Операторы и функции графики

9.2.1. Оператор PSET

Используется для окрашивания точки на экране в заданный цвет.
Формат:

```
PSET [ @  ](<АРГ1>,<АРГ2>)[,<АРГ3>]
      STEP
```

<АРГ1>::= целое выражение, задающее координату X точки;
<АРГ2>::= целое выражение, задающее координату Y точки;
<АРГ3>::= целое выражение, задающее цвет точки;
число из интервала [0,4];
STEP (@) – координаты указанной точки подсчитываются относительно последней точки, обработанной операторами графики.

Если заданная точка находится за пределами экрана (пределы экрана определяются координатами X – [0,511], Y – [0,239]), то команда никаких действий не производит. В противном случае точка окрашивается в цвет, определённый аргументом 3, или, если он отсутствует, в текущий цвет, определённый оператором COLOR.

Если номер цвета выходит за допустимые пределы, то выдаётся сообщение об ошибке 5.

Пример:

```
5 CLS
10 COLOR 1
15 FOR Y=1 TO 240
20 PSET (SQR(Y),Y)
30 NEXT
```

9.2.2. Оператор PRESET

Оператор используется для окрашивания точки на экране в цвет фона.
Формат:

```
PRESET [ @  ](<АРГ1>,<АРГ2>)[,<АРГ3>]
       STEP
```

<АРГ1>::= целое выражение, задающее координату X точки;
<АРГ2>::= целое выражение, задающее координату Y точки;
<АРГ3>::= целое выражение, задавшее цвет точки;
число из интервала [0,4];
STEP (@) – координаты указанной точки подсчитываются относительно последней точки, обработанной операторами графики

Этот оператор окрашивает заданную точку экрана в цвет фона. Если задан аргумент 3, то оператор превращается в оператор PSET – точка окрашивается в указанный цвет. Координаты точки X и Y должны находиться в интервалах [0,511] и [0,239] соответственно, иначе никаких действий не произойдёт.

Если номер цвета выходит за допустимые пределы, то выдаётся сообщение об ошибке 5.

Пример:

```
...
100 C%=POINT(X%,Y%)
110 PRESET(X%,Y%)
...
200 IF POINT(X%, Y%)<>C% THEN PSET(X%,Y%),C%
...
```

9.2.3. Функция POINT

Функция выдаёт номер цвета указанной точки экрана.

Формат:

```
X=POINT(<АРГУМЕНТ1>,<АРГУМЕНТ2>)
```

<АРГУМЕНТ1>::= целое выражение, задающее координату X;
<АРГУМЕНТ2>:: = целое выражение, задающее координату Y;

Функция POINT выдаёт значение целого типа – номер цвета заданной точки экрана, число от 1 до 4. Если координаты точки выходят за пределы экрана, то выдаётся значение -1. Если точка не принадлежит ни одному из начерченных на экране объектов, то выдаётся номер цвета фона.

С помощью функции POINT можно определить, вычерчивалось ли что-нибудь в заданном месте экрана.

Пример:

```
...
100 GOSUB 1300
110 IF POINT(X0, Y0)=4% THEN 100
120 GOSUB 2000
...
```

9.2.4. Оператор LINE

Оператор используется для вычерчивания на экране линий и прямоугольников.

Формат:

```
LINE [[ @  ](<АРГ1>,<АРГ2>)]-
       STEP
      [ @  ](<АРГ3),<АРГ4>)[,<АРГ5>,[B ]
       STEP                          BF
```

<АРГ1>::= целое выражение, задающее координату X начальной точки объекта;
<АРГ2>::= целое выражение, задающее координату Y начальной точки объекта;
<АРГ3>::= целое выражение, задающее координату X конечной точки объекта;
<АРГ4>::= целое выражение, задающее координату Y конечной точки объекта;
<АРГ5>::= целое выражение, задающее номер цвета объекта, число из интервала [0,4];

B – вычерчивается прямоугольник между заданными вершинами,
BF – вычерчивается закрашенный прямоугольник;
STEP(@) – указывает, что координаты подсчитываются относительно последней точки, обработанной операторами графики.

Этот оператор используется для вычерчивания на экране линий, контуров прямоугольников или закрашенных прямоугольников.

Аргументы с первого по четвёртый задают координаты начальной аргументы 1 и 2) и конечной аргументы 3 и 4) точки объекта (для прямоугольника – это диагонально противоположные углы). Если первые два аргумента опущены, то по умолчанию берутся координаты точки, заданные в предыдущем операторе графики.

Если аргумент 5 опущен, то выбирается текущий цвет, определённый оператором COLOR.

Если задан параметр B (BF), то вычерчивается прямоугольник (закрашенный прямоугольник). Если ни одного из этих параметров нет, то чертится линия.

Аргументы с первого по четвёртый могут принимать значения из интервала [-32768, 32767], но на экране отображаются только точки с координатами X и Y в интервалах [0, 511] и [0, 239] соответственно.

Если код цвета выходит за допустимые границы, то выдаётся сообщение об ошибке 5.

Пример:

```
10 LINE -@(SCALE*3,SCALE*4),2%
20 LINE (100,100)-(200,200),,B
30 LINE (125,125)-@(50,50),,BF
```

9.2.5. Оператор CIRCLE

Используется для вычерчивания на экране окружностей, эллипсов и дуг произвольного размера и цвета.

```
CIRCLE [ @  ](<АРГ1>,<АРГ2>),<АРГ3>
```

```
        STEP
[,<АРГ4>[,<АРГ5>[,<АРГ6>[,<АРГ7>]]]]
```

<АРГ1>::= целое выражение, задающее координату X центра окружности;

<АРГ2>::= целое выражение, задающее координату Y центра окружности;

<АРГ3>::= целое выражение, задающее радиус окружности;

<АРГ4>::= целое выражение, задающее номер цвета; число из интервала [0,4];

<АРГ5>::= арифметическое выражение, задающее положение начальной точки дуги (в радианах);

<АРГ6>::= арифметическое выражение, задающее положение конечной точки дуги (в радианах);

<АРГ7>::= арифметическое выражение, задающее коэффициент «сжатия» эллипса;

STEP (@) – координаты указанной точки вычисляются относительно последней точки, обработанной операторами графики.

Этот оператор используется для вычерчивания окружностей, эллипсов или их частей (дуг). На экран выводится лишь та часть окружности, которая помещается на координатной плоскости (координата X в пределах [0, 511], Y – [0, 239]).

Первые два аргумента задают положение центра окружности на координатной плоскости.

Аргумент 3 задаёт радиус вычерчиваемой окружности.

Аргумент 5 и аргумент 6 указывают начало и конец вычерчиваемой дуги соответственно и должны принадлежать интервалу [-6.283185, 6.283185] ([-2*PI, 2*PI]). Если эти аргументы отрицательны, то берутся их абсолютные значения, а центр окружности соединяется с соответствующим концом дуги. По умолчанию они равны 0 и 2*PI.

Если задан аргумент 7, то вычерчивается эллипс. По умолчанию этот параметр равен 1.

Неправильное задание аргументов может привести к синтаксической ошибке или ошибке переполнения. Если номер цвета выходит за допустимые пределы, то выдаётся сообщение об ошибке 5.

Пример:

```
10 CLS
20 COLOR 3
30 CIRCLE(100,100),75,,-1,-.001,.6
40 CIRCLE(100,200),75,,-.001,-1,.6
```

9.2.6. Оператор PAINT

Используется для окрашивания части экрана в один цвет

```
PAINT [ @  ](<АРГ1>,<АРГ2>)[,<АРГ3>][,АРГ4]
       STEP
```

<АРГ1>::= целое выражение, задающее координату X начальной точки;
<АРГ2>::= целое выражение, задающее координату Y начальной точки;
<АРГ3>::= целое выражение, задающее номер цвета; число из интервала [0, 4];
<АРГ4>::= целое выражение, задающее цвет границы закрашиваемой области; число из интервала [0, 4];
STEP(@) – координаты указанной точки подсчитываются относительно последней точки, обработанной операторами графики.

Этот оператор предназначен для окрашивания части экрана в указанный цвет. Если аргумент 3 опущен, то область окрашивается текущим цветом, определённым оператором COLOR. Если аргумент 4 опущен, то берётся область с границей цвета закрашивания.

Точка, с которой начинается окрашивание, может быть любой внутренней точкой области.

Границы области должны быть чётко очерчены, иначе произойдёт выход за пределы области. За пределами экрана оператор никаких действий не производит.

Следует отметить, что за границу области принимаются контура не только с цветом, указанным аргументом 4, но и с цветом закрашивания, поэтому, если внутри окрашиваемой области окажется закрытый контур с цветом закрашивания, то он останется незакрашенным. Если номера цветов выходят за допустимые пределы, то выдаётся сообщение об ошибке 5.

Пример:

```
...
100 DRAW A$
110 PAINT(105,22),3
...
```

9.2.7. Оператор DRAW

Выполняет строки графических команд.

Формат:

```
DRAW <СИМВОЛЬНОЕ ВЫРАЖЕНИЕ>
```

<СИМВОЛЬНОЕ ВЫРАЖЕНИЕ>::= строка, содержащая графические команды.

Оператор даёт возможность удобно управлять графикой. Для этого имеются команды «графического языка», передаваемые оператору DRAW через строку символов. Возможны вызовы подстрок с командами, что

освобождает от ограничений длины строк (255 символов).

Команда оператора DRAW состоит из буквы, идентифицирующей команду, со следующими за ней аргументами. Аргументом может быть целая константа из интервала [0,32767] или обращение к переменной (индексированной переменной):

```
=<ИМЯ ЧИСЛОВОЙ ПЕРЕМЕННОЙ>;
```

Значение переменной должно быть в интервале [-32768, 32767].

На экране отображаются только те значения аргументов, которые соответствуют координатам X и Y точек в интервалах [0, 511] и [0, 239] соответственно. В качестве разделителей между командами можно использовать пробелы или точку с запятой, но их может и не быть.

Имеются следующие команды оператора DRAW:

```
U[<ДЛИНА>] черчение вверх от текущей точки;
D[<ДЛИНА>] вниз;
L[<ДЛИНА>] налево;
R[<ДЛИНА>] направо;
E[<ДЛИНА>] в правый верхний угол;
F[<ДЛИНА>] в правый нижний угол;
G[<ДЛИНА>] в левый нижний угол;
H[<ДЛИНА>] в левый верхний угол.
```

В графическом представлении (все углы 45 градусов):

```
H U E
 \|/
L-+-R
 /|\
G D F
```

Пример:

```
10 CLS
20 COLOR 1
30 PSET (150,0)
40 DRAW «F100»
50 COLOR 2
60 DRAW «M0,100»
```

Для черчения с указанием координат точек используется команда M.

```
M[+]<АРГ1>,[+]<АРГ2>
  -          -
```

Команда чертит от текущей точки до точки с координатами X и Y, задаваемыми аргументами 1 и 2. Если перед первым аргументом указан знак «+» или «-» то координаты подсчитываются относительно координат текущей

точки. Знак перед вторым аргументом не влияет на этот режим. В команде M «;» после имени числовой переменной в первом аргументе не заменяет запятой.

Имеются команды, используемые непосредственно перед перечисленными командами, изменяющие их действие:

B - указывает, что команда должна передвигать текущую точку, но не чертить;

N - после выполнения команды возвращаются бывшие координаты текущей точки.

Следующие команды изменяют режим для всех последующих команд во всех последующих операторах DRAW, исключая команду M с абсолютными координатами.

A[<ПОВОРОТ>] Изменяет направление черчения, значения аргумента означают:
0 – нормальная ориентация (принимается по умолчанию);
1 – 90 градусов по часовой стрелке;
2 – 180 градусов по часовой стрелке;
3 – 270 градусов по часовой стрелке;

S[<МАСШТАБ>] Позволяет изменять масштаб черчения; аргумент должен быть в пределах [1, 255]. По умолчанию принимается значение 4. Значение 0 тоже указывает отсутствие масштабирования, т.е. значение 4, значение аргумента оператор DRAW делит на 4 и умножает на аргументы команд черчения. Таким образом, S1 означает 4-кратное уменьшение, S12 – 3-кратное увеличение.

Для изменения цвета имеется команда C:

C[<ЦВЕТ>] Задаёт новый цвет черчения, номер которого указывается аргументом и должен быть в пределах [0,4].

Для включения подстрок используется команда X:

```
X<ИМЯ СИМВОЛЬНОЙ ПЕРЕМЕННОЙ>;
```

Переменная интерпретируется как подстрока, содержащая графические команды.

Оператор DRAW удобен для черчения различных фигур, создания шрифтов и библиотек элементов чертежей с последующей их компоновкой при помощи команды X, размещения их в любом месте, направлении и масштабе. Нарушение пределов аргументов или неправильная запись команд вызывает ошибку 5.

Пример:

```
10 A%=100
20 B%=150
```

```
30 A$=«E10F10D20L20U20R20»
40 DRAW"BM=A%;,100S2XA$;BM=B%;,100S4XA$;"
```

9.3.Оператор выдачи звука

9.3.1. Оператор BEEP

Оператор выдаёт короткий звуковой сигнал.

Формат:

```
BEEP[<АРГУМЕНТ1>][,<АРГУМЕНТ2>]
```

<АРГУМЕНТ1>::= целое выражение, означающее продолжительность звука;

<АРГУМЕНТ2>::= целое выражение, означающее высоту тона.

Этот оператор используется для программирования музыки, а также для получения других звуковых эффектов. При отсутствии обоих аргументов выдаётся такой же сигнал, как в случае возникновения ошибки.

Пример:

```
100 FOR I=1 TO 10
110 BEEP
120 NEXT
```

10. ПРОГРАММИРОВАНИЕ ВВОДА/ВЫВОДА

Операторы ввода/вывода INPUT, PRINT и другие позволяют организовать обмен информацией с внешними устройствами.

10.1. Оператор PRINT

Выводит данные на экран, печатающее устройство или в файл на магнитной ленте.

Формат:

```
PRINT
?       [#][<СПИСОК>]
LPRINT
```

<СПИСОК>::= одно или несколько выражений любого типа, функций оператора PRINT, разделённых запятой или точкой с запятой;

L указывает, что вывод осуществляется на печатающее устройство;

обозначает вывод в файл на магнитной ленте.

Слово PRINT может быть заменено вопросительным знаком "?", за исключением варианта LPRINT.

Оператор пересылает данные на устройство вывода. Обычно устройством вывода является экран, но этот оператор может вывести данные и

на печатающее устройство или в открытый оператором OPEN файл на кассетной магнитной ленте. Список выводимых данных может быть пустым или содержать несколько выражений любого типа, значения которых преобразуются в текст и выводятся на указанное устройство. Запятая между двумя элементами списка данных означает, что данные будут выводиться зонами, т.е., каждому элементу списка выделяется по 14 позиций. Остаток зоны за данными заполняется пробелами. Если между двумя аргументами списка данных стоит точка с запятой, то значения выводятся непосредственно друг за другом. Если запятая или точка с запятой является последней в списке данных, то курсор остаётся на той же позиции, на которой был после вывода последнего значения, в противном случае осуществляется перевод строки.

Оператор LPRINT выводит все данные на печатающее устройство.

Пример:

```
PRINT 1;-2,3
1   -2             3
OK
```

10.2. Оператор INPUT

Вводит данные с клавиатуры или из файла на магнитной ленте.

Формат:

```
INPUT [<АРГУМЕНТ>;][<СПИСОК>]
          #
```

<АРГУМЕНТ>::= подсказка в виде текстовой константы;
<СПИСОК>::= одна или несколько переменных или элементов массива, разделанных запятыми;

указывает на ввод из файла на магнитной ленте.

Этот оператор организует ввод с клавиатуры или из открытого оператором OPEN файла на магнитной ленте. Если знака "#" нет, то ввод осуществляется с клавиатуры. Если в случае ввода с клавиатуры указан <АРГУМЕНТ>, то он выводится на экран как подсказка. Затем выводится вопросительный знак "?", высвечивается курсор, и Бейсик-система ожидает ввода данных. В качестве данных могут быть любые константы, по типу и числу соответствующие элементам списка. В качестве разделителей используются запятые. В текстовых константах могут отсутствовать кавычки, если они не содержат запятых. Если ввод прекращается раньше конца списка переменных, то на следующей строке выдаются знаки "??", и Бейсик-система ждёт продолжения ввода. Если вводится больше данных, чем переменных в списке, то последние значения игнорируются.

Если при вводе числовых данных встречаются недопустимые знаки, то выдаётся сообщение об ошибке 13 и осуществляется повторный ввод. При попытке оператором INPUT# прочесть метку конца файла выдаётся сообщение об ошибке 55.

Пример:

```
INPUT "ДАННЫЕ";A,B$,C$
ДАННЫЕ?123,"ABC,EFG"
??P"Q
OK
PRINT A,B$,C$
123         ABC,EFG
P"Q
OK
```

10.3. Функция INKEY$

Даёт один символ, введённый с клавиатуры.

Формат:

X$=INKEY$

Эта функция считывает символ из буфера ввода клавиатуры. Если не было введено ни одного символа, то INKEY$ выдаёт пустую символьную строку (""). Если символ был введён, то он считывается из буфера и выдаётся через INKEY$. После обращения к этой функции буфер очищается, поэтому надо заботиться и сохранении результата.

Эта функция может быть применена везде в качестве символьной строки. Надо обратить внимание на то, что результатом функции может быть пустая строка, в некоторых случаях вызывающая ошибку. Эта функция является главным способом считывания кодов клавиш.

Достоинствами этой функции являются то, что она не повторяет считывания кода клавиши, а также не ждёт символа, если не было нажатия ни на одну клавишу, что обеспечивает недиалоговый режим ввода с клавиатуры (в отличие от оператора INPUT). INKEY$ может прочитать коды не только алфавитно-цифровых клавиш. При считывании не производится эхо-печать на экране.

Пример:

```
10 CH$=IHKEY$
20 IF CH$="" THEN 10
30 PRINT ASC(CH$);
40 GOTO 10
```

(Программа печатает коды клавиш).

10.4. Оператор OPEN

Открывает файл данных на магнитной ленте.

Формат:

```
OPEN <СПЕЦИФИКАЦИЯ ФАЙЛА>[FOR INPUT
                          OUTPUT]
```

<СПЕЦИФИКАЦИЯ ФАЙЛА>::= символьное выражение, определяющее имя файла;

Оператор начинает процесс ввода/вывода данных. Буфер ввода/вывода связывается с именем файла. Слова INPUT и OUTPUT указывают, что открываются соответственно входной или выходной файл. Данные на магнитной ленте хранятся в формате КОИ-8, блоками по 256 символов. Конец файла отмечается символом "СУ/Z" (код &O32). Аналогичным форматом в командах SAVE, LOAD и MERGE на магнитной ленте хранится текст программы. При открытии входного файла блок данных считывается в буфер.

Данные из входного файла можно читать оператором INPUT#, записывать в выходной файл – оператором PRINT#. Имя файла должно состоять не более, чем из 6 символов. Также можно указать тип файла. Тип файла должен состоять не более чем из 3 символов и отделяться от имени файла точкой:

```
DATA.TXT
```

По умолчанию в операторе OPEN устанавливается тип файла DAT. Можно указать любой другой тип файла, только в случае ввода этот файл должен быть записан в формате КОИ-8.

В случае слишком длинного или пустого имени файла выводится сообщение об ошибке 56. Попытка повторно открыть файл приводит к ошибке 54.

Пример:

```
10 OPEN "PROG.ASC" FOR INPUT
20 IF EOF THEN 60
30 INPUT# A$
40 PRINT A$
50 GOTO 20
60 CLOS
```

(На экран выводится текст программы PROG).

10.5. Оператор CLOSE

Заканчивает операции ввода/вывода и освобождает буфер.

Формат:

```
CLOSE
```

Если оператором OPEN был открыт выходной файл, к выводимым данным добавляются символ <ПС> и указатель конца файла "СУ/Z" (код &O32), выводится последний блок файла и очищается буфер. В случае входного файла просто очищается буфер.

Некоторые операции делают CLOSE автоматически. Это END, CLEAR, LOAD, NEW и любое изменение текста программы.

Пример:

```
10 OPEN "DATA" FOR OUTPUT
20 FOR I%=0% TO 10%
```

```
30 PRINT# X(I%)
40 NEXT
50 CLOSE
```

10.6. Функция EOF

Проверяет достижение конца файла.

Формат:

```
X=EOF
```

Функция проверяет, равен ли следующий ещё не введённый оператором INPUT# символ указателю конца файла «СУ/Z» (восьмеричный код 32). Если нет, то выдаётся значение – 1 (TRUE), в противном случае – 0 (FALSE). Если при обращении к функции EOF буфер оказывается пустым, то вводится очередной блок файла.

Так как чтение метки конца файла вызывает ошибку 55, то целесообразно перед каждым INPUT# проверять достижение конца файла при помощи функции EOF. Функция даёт логический результат, поэтому её удобно использовать в операторе IF. Если файл не открыт, выдаётся сообщение об ошибке 59. Если открыт выходной файл, то обращение к EOF приводит к ошибке 52.

Пример:

```
50 IF NOT EOF THEN 20
60 CLOSE
```

10.7. Функция CSRLIN

Результат функции – номер строки экрана, на которой находится курсор.

Формат:

```
X=CSRLIN[(<АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ>)]
```

Результат функции – число целого типа, указывающее, на которой строке экрана находится курсор. Верхняя строка имеет номер 0, нижняя – 23. Заключённое в скобки арифметическое выражение не используется, хотя подсчитывается, что может привести к ошибке.

CSRLIN вместе с функцией POS применяется для определения места на экране, куда будет выводиться информация. Оператором LOCATE можно менять местоположение курсора.

Пример:

```
? CSRLIN
10
OK
```

(Результат примера зависит от местонахождения курсора).

10.8. Функция POS

Функция выдаёт положение курсора в печатной строке (номер столбца).

Формат:

```
X=POS[(<АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ>)]
```

Функция позволяет получить номер столбца, в котором находится курсор. Эта функция вместе с CSRLIN применяется для определения положения курсора. Значение заключённого в скобки арифметического выражения хотя и подсчитывается, но функцией не используется.

Пример:

```
1000 X=POS
1010 Y=CSRLIN
1020 LOCATE X1,Y1
1030 GOSUB 2000
1040 STOP
2000 LOCATE X,Y
2100 RETURN
```

10.9. Функция LPOS

Даёт положение головки печатающего устройства.

Формат:

```
X=LPOS[(<АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ>)]
```

Функция LPOS выдаёт значение счётчика положения головки печатающего устройства. Первая позиция имеет номер 1. Каждый напечатанный символ увеличивает значение счётчика, переход к новой строке очищает его, присваивая значение 1. Действие функции надёжно только при выводе текста, так как Бейсик-система не контролирует управляющих символов, которые могут повлиять на положение головки.

Функцию LPOS( ) можно использовать при выводе данных на принтер столбцами.

Пример:

```
250 IF LPOS>20 THEN LPRINT
260 LPRINT TAB(20);A
```

10.10. Функции оператора PRINT

10.10.1. Функция AT

Функция позволяет организовать вывод данных оператором PRINT с произвольной позиции экрана.

Формат:

```
PRINT AT(<АРГУМЕНТ1>,<АРГУМЕНТ2>)
```

<АРГУМЕНТ1>::= целое выражение, значение которого указывает колонку экрана, начиная с которой будут выводиться данные; число из интервала [0,255]

<АРГУМЕНТ 2>::= целое выражение, значение которого указывает строку экрана; число из интервала [0,255].

Действия функции AT( ) аналогичны действиям оператора LOCATE. Отличие состоит в том, что, используя AT( ), можно выводить текст в произвольных местах экрана одним оператором PRINT. Значение первого аргумента задаёт колонку экрана, а значение второго – строку. Тем самым задаётся позиция на экране, с которой начинается вывод последующих данных. Колонки экрана нумеруются от 0 до 31 слева направо, строки – от 0 до 23 сверху вниз. Если значение первого аргумента превышает 31, то подсчёт колонок продолжается с начала той же строки. Аналогичные действия происходят со строками.

Если значения аргументов выходят за допустимые пределы, то выдаётся сообщение об ошибке 5.

Пример:

```
PRINT AT(15,14);"A"
OK
PRINT AT(15,38);"C"
OK
```

10.10.2. Функция TAB

Функция помогает организовать построение колонок в операторах PRINT.

Формат:

```
PRINT TAB(<АРГУМЕНТ>)
```

<АРГУМЕНТ>::= целое выражение, указывающее номер позиции, с которой должен продолжаться вывод; число из интервала [0, 255].

Функция TAB( ) может применяться и в операторе LPRINT. Она выводит пробелы до тех пор, пока не будет достигнута колонка с номером, равным значению аргумента. Если курсор находится на позиции с большим номером, чем значение аргумента, то TAB( ) игнорируется.

Позиции строки экрана нумеруются от 0 до 31 слева направо. Если значение аргумента превышает 31, то подсчёт колонок продолжается на следующей строке.

Если значение аргумента выходит за допустимые пределы, то фиксируется ошибка 5.

Пример:

```
100 CW=8
110 PRINT A;TAB(CW);B;TAB(2*CW);C;TAB(3*CW);D
```

10.10.3. Функция SPC

В операторе PRINT выводит пробелы.

Формат:

```
PRINT SPC(<АРГУМЕНТ>)
```

<АРГУМЕНТ>::= целое выражение, означающее число пробелов и принимающее значения в пределах [0,255].

Эта функция используется в операторах PRINT и LPRINT для вывода необходимого числа пробелов. Она не использует символьной области.

SPC( ) может употребляться для очистки экрана впереди и после выводимых данных от «мусора», возможно оставшегося после предыдущего вывода.

Пример:

```
220 ZZ=8
230 PRINT SPC(ZZ);AX;SPC(ZZ);BX;SPC(ZZ)
```

11. ХРАНЕНИЕ И ЗАГРУЗКА ПРОГРАММ

Тексты программ пользователей «ЭЛЕКТРОНИКА БК 0011» хранятся на кассетной магнитной ленте. Описываемые далее операторы предназначены для чтения (записи) программ с (на) магнитной ленты. Так же имеются средства для хранения на магнитной ленте двоичной информации.

Для идентификации программ используется спецификация файла – текстовая константа, содержащая символическое имя файла. Имя файла может состоять не более чем из шести символов.

В операторах BLOAD и BSAVE границы областей памяти указываются при помощи адреса – целого выражения, значение которого берётся без знака и обозначает адрес из интервала [0,65535]. Для удобства адрес можно задавать шестнадцатеричными или восьмеричными константами.

11.1. Оператор LOAD

Загружает текст программы пользователя (в кодах КОИ-8).

Формат:

```
LOAD <СПЕЦИФИКАЦИЯ ФАЙЛА>[,R]
```

При наличии R программа после её загрузки запускается на выполнение.

Оператор используется для загрузки текста программы, размещенного на кассетной магнитной ленте с помощью оператора SAVE. Наличие параметра R приводит к немедленному запуску программы после её загрузки. Текст программы представлен на магнитной ленте в кодах КОИ-8.

Пример:

```
LOAD «MOD2»,R
```

```
MOD2 .ASC 003
OK
```

11.2. Оператор SAVE

Оператор предназначен для записи текста программы на кассетную магнитную ленту (в кодах КОИ-8).

Формат:

```
SAVE <СПЕЦИФИКАЦИЯ ФАЙЛА>
```

Главное применение команды SAVE – сохранить текст программы пользователя на магнитной ленте.

Пример:

```
FILES «ПЕРВЫЙ.COD»
ПЕРВЫЙ.COD
OK
SAVE«ВТОРОЙ»
OK
```

11.3. Оператор MERGE

Оператор объединяет текст программы в памяти ЭВМ с текстом программы, загружаемым с кассетной магнитной ленты.

Формат:

```
MERGE <СПЕЦИФИКАЦИЯ ФАЙЛА>
```

При выполнении этой команды загружается программа с магнитной ленты и объединяется с программой, хранящейся в памяти. Загружаемая программа должна быть в формате КОИ-8.

При объединении текстов в случае совпадения номера загружаемой строки с номером строки, хранящейся в памяти, оставляется загружаемая строка.

Оператор позволяет при помощи DELETE, RENUM и SAVE запомнить часть одной программы, а потом объединить её с другой программой. MERGE всегда портит объектный код и аннулирует все определения имён, что требует новой компиляции всей программы.

Наличие этой команды в системе позволяет хранить на магнитной ленте набор типовых программ.

Пример:

```
LOAD «ПЕРВЫЙ»
ПЕРВЫЙ .ASC 005
OK
MERGE «ВТОРОЙ»
ВТОРОЙ .ASC 001
OK
```

11.4. Оператор CLOAD

Оператор загружает с кассетной магнитной ленты программу пользователя, записанную во внутреннем представлении системы.

Формат:

```
CLOAD <СПЕЦИФИКАЦИЯ ФАЙЛА>
```

Команда загружает с магнитной ленты текст программы, хранящейся во внутреннем представлении системы. Текст программы должен быть записан командой CSAVE.

Пример:

```
CLOAD «ПЕРВЫЙ»
ПЕРВЫЙ.COD
OK
```

11.5. Оператор CSAVE

Оператор записывает текст программы на магнитную ленту во внутреннем формате системы.

Формат:

```
CSAVE <СПЕЦИФИКАЦИЯ ФАЙЛА>
```

При выполнении этого оператора текст программы записывается во внутреннем представлении системы на магнитную ленту.

Пример:

```
CSAVE «1001»
OK
```

11.6. Оператор FILES

Оператор предназначен для поиска файла, записанного на кассетную магнитную ленту.

Формат:

```
FILES [<СПЕЦИФИКАЦИЯ ФАЙЛА>]
```

Оператор FILES позволяет подвести головку магнитофона к концу любого записанного на ленту файла. Имя нужного файла указывается аргументом. FILES удобно использовать при поиске свободного участка магнитной ленты для записи нового файла.

Как и LOAD, этот оператор при чтении выдаёт на экран имена всех обнаруженных файлов.

Если имя файла не указано, то FILES просто выдаст на экран имена всех найденных файлов.

При выполнении команды содержание ОЗУ не меняется. Оператор можно прервать нажатием клавиши «СТОП».

Оператор можно использовать и для пуска двигателя магнитофона при

необходимости перемотать ленту вперёд или назад.

Пример:

```
FILES"ПЕРВЫЙ"
ПЕРВЫЙ.COD
OK
```

11.7. Оператор BLOAD

Загружает «ДВОИЧНЫЙ» файл (машинный код или данные) в память.

Формат:

```
BLOAD <АРГ1>[,R][,<АРГ2>][,<АРГ3>][,<АРГ4>]
```

<АРГ1>::= спецификация файла;
R является признаком немедленного запуска программы после её загрузки;
<АРГ2>:: = адрес начальной загрузки файла.
<АРГ3>,<АРГ4>::= целые выражения, означающие номера страниц ОЗУ, подключаемых в окна адресного пространства.

Оператор загружает содержимое файла, записанного на кассетную магнитную ленту с помощью BSAVE. В файле хранятся начальный адрес и длина массива, согласно которым содержимое файла размещается в памяти машины.

При наличии аргумента, R, после загрузки содержимого файла происходит немедленный запуск программы с начального адреса загрузки. Указав аргумент 2, можно разместить содержимое файла в памяти, начиная с произвольного адреса оперативной памяти. При отсутствии аргумента R для указания аргумента 2 дополнительная запятая не требуется.

Если указаны аргументы 3 и 4, то в окна адресного пространства с 40000 по 100000 и с 100000 по 140000 (восьмеричные) подключаются соответствующие страницы памяти. Если эти аргументы (или один из них) пропущены, то в соответствующем окне остаётся ранее подключённая страница. Таким образом оператором BLOAD возможно загружать информацию в скрытые страницы ОЗУ (см. руководство системного программиста).

Загрузку можно прервать нажатием клавиши «СТОП». Во время загрузки на экран выдаётся имя файла.

Оператор позволяет загрузить программы в машинном представлении в оперативную память с целью их дальнейшего использования посредством функции USR. Отметим, что попытка загрузки в постоянную память вызывает прерывание загрузки.

Пример:

```
CLEAR 200,&O60000
OK
```

```
BLOAD "DATA",&O60000
DATA  .BIN
OK
BLOAD "PROG",R
PROG  .BIN
```

(Файл PROG загружен и запущен с начала).

11.8. Оператор BSAVE

Записывает содержимое оперативной памяти на кассетную магнитную ленту.

Формат:

```
BSAVE <АРГ1>,<АРГ2>,<АРГ3>[,<АРГ4>][,<АРГ5>]
```

<АРГ1>::=	спецификация файла;
<АРГ2>::=	адрес начала сохраняемой области;
<АРГ3>::=	адрес конца сохраняемой области.
<АРГ4>,<АРГ5>::=	целые выражения, означающие номера страниц памяти, подключаемых к окнам адресного пространства.

При выполнении этой команды содержимое оперативной памяти, начина с адреса, указанного аргументом 2, и кончая адресом, указанным аргументом 3, сохраняется на кассетной магнитной ленте в файле, имя которого указано аргументом 1. При отсутствии имени файла выдаётся сообщение об ошибке 5, при отсутствии аргументов – об ошибке 2. Выполнение команды можно прервать нажатием клавиши «СТОП». Кроме содержимого памяти на магнитную ленту записываются начальный адрес и длина области оперативной памяти.

Значения аргументов 4 и 5 такие же, как и в операторе BLOAD.

Выполнение оператора можно прервать нажатием клавиши «СТОП». Кроме содержимого памяти на магнитную ленту записываются начальный адрес и длина области оперативной памяти.

В основном этот оператор даёт возможность сохранять содержимое оперативной памяти на магнитной ленте в двоичном формате. Особенно полезно его использование при сохранении программ, написанных на машинном языке и используемых в Бейсик-системе при помощи функции USR.

Пример:

```
BSAVE"PROG",&O10000,&O20000
OK
```

12. НЕПОСРЕДСТВЕННЫЙ ДОСТУП К ПАМЯТИ

12.1. Оператор POKE

Записывает данные в любую ячейку памяти ЭВМ.

Формат:

```
POKE [<СТРАНИЦА>,]<АДРЕС>,<ВЫРАЖЕНИЕ>
```

<СТРАНИЦА>::= целое выражение, указывающее страницу памяти;

<АДРЕС>::= целое выражение, определяющее адрес ячейки;

<ВЫРАЖЕНИЕ>::= целое выражение, определяющее записываемые данные.

Этот оператор записывает значение выражения в указанную ячейку (слово) памяти, оба параметра при этом переводятся в целый тип. Первый аргумент указывает страницу ОЗУ, в которую должны записываться данные и должен принимать значения из интервала [0,11] (см. руководство системного программиста). Если он опущен, то производится запись в последнюю подключённую в соответствующее окно адресного пространства страницу, в этом случае лишняя запятая не ставится. Если адрес нечётный, то берётся слово, в котором находится указанный байт, т.е., значение адреса уменьшается на 1. Этот оператор можно использовать для изменения указателей и таблиц Бейсик-системы, размещения в памяти программ, написанных в машинных кодах, для работы с регистрами внешних устройств.

Пользоваться оператором POKE надо осторожно, так как ошибочные данные или адрес могут вызвать нежелательные результаты.

Пример:

```
10 POKE 3%,0%
20 PRINT PEEK(3%)
RUN
0
OK
```

12.2. Функция PEEK

Позволяет прочесть содержимое любой ячейки памяти.

Формат:

```
X=PEEK([<АРГУМЕНТ1>,]<АРГУМЕНТ2>)
```

<АРГУМЕНТ1>::= целое выражение, задающее номер страницы памяти;

<АРГУМЕНТ2>::= целое выражение, задающее адрес ячейки.

Эта функция позволяет исследовать любое определённое место памяти в адресном пространстве в любой странице памяти. С её помощью можно

читать системные переменные драйверов и Бейсик-системы. Также функция PEEK удобна для ввода данных непосредственно с регистров внешних устройств. Результатом функции является целое число – содержимое слова памяти, адрес которого задаётся аргументом 2. Аргумент 1 указывает номер страницы памяти и может принимать значения из интервала [0,11] (см. руководство системного программиста). Если указан только аргумент 2, то подключение страниц памяти не меняется. Значения аргумента 2 от -32768 до -1 обозначают адреса от 32768 до 65535 соответственно.

Пример:

```
...
100 PRINT PEEK(&O1000);PEEK(5%,&O40000)
...
```

12.3. Оператор OUT

Записывает данные в оперативную память по маске.

Формат:

```
OUT [<СТРАНИЦА>,]<АДРЕС>,<МАСКА>,<КОД>
```

<СТРАНИЦА>::= целое выражение, означающее страницу ОЗУ;

<АДРЕС>::= целое выражение, указывающее физический адрес ячейки памяти;

<МАСКА>::= целое выражение, указывающее участвующие в операции биты;

<КОД>::= целое выражение, указывающее, должны ли очищаться или устанавливаться выбранные биты; нулевое значение указывает на очистку, ненулевое – на установку.

Оператор позволяет записывать или стирать отдельные биты непосредственно в ячейках оперативной памяти. Биты ячейки, соответствующие нулевым битам маски, остаются неизменными, остальные биты очищаются (если значение (кода) равно нулю), или устанавливаются (в противном случае).

Пример:

```
10 POKE A%,&O177777
20 POKE B%,0%
30 OUT A%,&O111111,0%
40 OUT B%,&O66666,1%
50 PRINT OCT$(PEEK(A%)),OCT$(PEEK(B%))
RUN
66666           66666
OK
```

12.4. Функция INP

Разрешает чтение данных в оперативной памяти по маске.

Формат:

```
X=INP([<СТРАНИЦА>,]<АДРЕС>,<МАСКА>)
```

<СТРАНИЦА>::= целое выражение, означающее страницу ОЗУ;
<АДРЕС>::= целое выражение, указывающее физический адрес ячейки ОЗУ;
<МАСКА>::= целое выражение, указывающее, какие биты считываются из содержимого ячейки.

При помощи функции INP можно аналогично функции PEEK непосредственно читать содержимое ячеек оперативной памяти, при этом можно выбирать только необходимые биты. Функция даёт результат целого типа, в котором биты, соответствующие ненулевым битам маски, читаются из указанной ячейки, а другие биты имеют значение 0.

Пример:

```
POKE A%,&B10101
OK
PRINT BIN$(INP(A%,&B11001))
10001
OK
```

Функция удобна при работе с регистрами внешних устройств.

Пример:

```
10 POKE &O177660,&O100
20 IF INP(&O177660,&O200) THEN 30 ELSE 20
30 A%=PEEK(&O177662)
40 PRINT A%
50 GOTO 20
```

(при нажатии на клавиши программа печатает коды клавиш клавиатуры).

12.5. Функция VARPTR

Выдаёт значение физического адреса расположения переменной в памяти.

Формат:

```
A%=VARPTR (<ПЕРЕМЕННАЯ>)
```

Значение функции – указатель расположения переменной или элемента массива. Её удобно использовать при обмене данными с машинными подпрограммами USR. Например, указав в качестве аргумента USR указатель первого элемента массива, можно иметь доступ ко всему массиву, так как элементы массивов в Бейсике размещаются последовательно.

В случае числовой переменной (элемента числового массива) значение переменной находится прямо в указываемой ячейке (в одном машинном слове в случае переменной целого типа или в двух соседних словах в случае вещественной переменной).

В случае символьной переменной значение функции указывает на два слова, в первом из которых находится длина символьной переменной, во втором — адрес начала строки. В элементе символьного массива длина хранится только в одном байте, таким образом функция VARPTR выдаёт адрес трех байтов, в первом из которых хранится длина строки, в других двух (необязательно слове) — адрес начала строки.

Если указан элемент виртуального массива, то функция выдаёт адрес в виртуальном адресном пространстве, образуемом скрытыми страницами ОЗУ. В этом случае адреса от 0 до 37776 (восьмеричных) означают первую страницу, отведённую для виртуальных массивов, от 40000 до 77776 —вторую и т.д.

Элементом символьного виртуального массива является область памяти длиной 256 байт, первый из которых содержит длину строки, а далее следует сама строка. Оставшиеся байты области не используются. Адрес этой области в виртуальном адресном пространстве и выдаёт функция VARPTR.

Пример:

```
A% = 123
OK
PRINT PEEK(VARPTR(A%))
123
OK
```

12.6. Функция FRE

Указывает, сколько осталось свободной памяти компьютера для программы пользователя.

Формат:

```
X=FRE[(<АРГУМЕНТ>)]
```

<АРГУМЕНТ>::= символьное или арифметическое выражение (важен его тип, а не значение)

Если аргумент – число или отсутствует, то подсчитывается и выдаётся объём памяти, отведённой под программу пользователя. Если же аргумент символьного типа, то выдаётся свободный объём памяти, отведённой, под символьные переменные (см. оператор CLEAR). Таким образом, функция совершенно не зависит от значения аргумента, важен только его тип. С другой стороны, при обращении к функции значение выражения всё же высчитывается, и, в случае возникновения ошибки (например, переполнения), дальнейшее выполнение программы прекращается.

Свободной области виртуальных массивов функция не выдаёт.

Пример:

```
PRINT FRE(" "),FRE(0)
200              30514
```

Приложение 1.
Сообщения об ошибках

В приложении указаны коды ошибок, их сокращённые названия и описания.

1 NEXT без FOR
NEXT не предшествовал FOR или переменная, использованная в FOR не соответствует переменной, использованной NEXT.

2 Синтаксическая ошибка
Неверное использование символов, например, число открывающих скобок не соответствует числу закрывающих; неправильная запись операторов или их составных частей; неправильно использована запятая и т.п.

3 RETURN без GOSUB
При выполнении RETURN обнаружено, что не был выполнен GOSUB.

4 Кончились DATA
При выполнении оператора READ обнаружено, что список оператора DATA исчерпан.

5 Неправильный вызов функции
Ошибка возникает в следующих случаях:
- отрицательный индекс;
- неправильный аргумент для LOG, SQR;
- неправильные значения аргументов для графических и других операторов;
- неопределённая с DEF USR функция USR и др.

6 Переполнение
Результат арифметической операции слишком велик по абсолютному значению и не может быть записан в формате, принятом для чисел в Бейсик-системе. (В случае слишком малых величин результат приравнивается нулю и ошибки не выдаётся).

7 Нет больше памяти
Ошибка возникает в следующих случаях:
- программа не помещается в памяти;
- использовано слишком много вложенных операторов FOR или GOSUB;
- слишком много переменных;
- слишком много затребовано памяти для массива или области строк;
- слишком мала свободная область для генерирования кода

непосредственно выполняемых операторов;
- в операторах CLOAD или BLOAD файл не помещается в отведённое для него место в памяти;
- в операторе PAINT контур для закрашивания слишком сложный.

8 Не определён номер строки
В GOTO, GOSUB, IF, RESTORE, RENUM, AUTO или DELETE использован неопределённый номер строки.

9 Несоответствие индекса
Ссылка на элемент массива с индексом, который выходит за пределы размерности массива, либо указано неправильное число индексов.

10 Повторное определение массива
Массив определён двумя операторами DIM; или массив определён оператором DIM после того, как по умолчанию для этого массива была установлена размерность 10.

11 Деление на нуль
В выражении встретилось деление на нуль; или нуль был возведён в отрицательную степень.

12 Невозможно выполнить
Недопустимый оператор в режиме непосредственного выполнения или команда в косвенном режиме.

13 Ошибка типов
Попытка присвоить символьной переменной числовое значение или наоборот; функции, использующей числовой аргумент, передаётся символьный аргумент или наоборот.

14 Кончилась память строк
Превышено количество оставшейся памяти, которая была отведена символьным переменным по умолчанию или оператором CLEAR.

15 Слишком длинная строка
Была попытка создать строку длиной более чем 255 символов.

16 Слишком сложное выражение
Слишком большое число вложенных скобок или функций MID$ в выражении.

17 Продолжение невозможно.
Была сделана попытка продолжить выполнение программы, которая:
- была прервана из-за возникновения ошибки;

- была изменена;
- не существует.

18 Неопределённая функция
Попытка обратиться к функции FN до её определения оператором DEF FN.

19 Ошибка ввода/вывода
Встречается при работе устройства ввода/вывода

20 Не определена

21 Не определена

22 Не определена

23 Не определена

24 Отсутствует операнд
Выражение содержит оператор без операнда, или в операторе (команде) нет обязательных параметров.

25 Переполнение буфера ввода
Была сделана попытка ввести строку длиной более чем 255 символов.

26-51 Не определены

52 Недопустимое обращение к файлу
Оператор или команда ссылается на файл, который открыт с другой целью (не для записи или чтения).

53 Не определена

54 Файл уже открыт
Применён оператор OPEN для файла, который уже открыт.

55 Конец файла
Оператор INPUT# был выполнен после того, как все данные из файла были введены, или применён для пустого файла. Чтобы избежать этой ошибки, для обнаружения окончания файла применяйте функцию EOF.

56 Неправильное имя файла
В операторах LOAD, SAVE или OPEN использовано неправильное имя файла (имя файла состоит из слишком большого количества символов).

57 Строка без номера
Команда непосредственного выполнения обнаружена в загружаемой

программе во время выполнения оператора LOAD; загрузка прерывается.

58 Не определена

59 Файл не открыт
Оператор ввода/вывода применён к файлу, который не был открыт.

60 Не определена

61 Не определена

62 Ошибочное имя устройства
Было использовано ошибочное имя устройства.

Приложение 2.
Список команд, операторов и функций

В приложении приводятся команды, операторы и функции Бейсик-системы (версия 25.12.1987), сгруппированные по назначению

Модификация текста программы в памяти

AUTO	Автоматически нумерует строки при вводе текста программы.
DELETE	Удаляет группы строк из текста программы.
NEW	Удаляет всю программу и выполняет оператор CLEAR.
LIST	Выводит текст программы на экран.
LLIST	Выводит текст программы на печатающее устройство.
RENUM	Перенумеровывает строки текста программы.
.	Позволяет редактировать строку.
<ВС>	Позволяет редактировать последнюю введённую строку.

Хранение программ на магнитной ленте

BLOAD	Загружает с магнитной ленты в оперативную память двоичные коды.
BSAVE	Записывает блок памяти на магнитную ленту в двоичном коде.
CLOAD	Загружает программу с магнитной ленты во внутреннем формате Бейсик-системы.
CSAVE	Записывает программу на магнитную ленту во внутреннем формате.
LOAD	Загружает текст программы с магнитной ленты (формат КОИ-8).

SAVE	Записывает текст программы на магнитную ленту (формат КОИ-8).
MERGE	Загружает программу (файл в формате КОИ-8) с магнитной ленты и объединяет её с программой в ОЗУ.
FILES	Производит фиктивное чтение или поиск файла с указанным именем.

Инициализация системных переменных

CLEAR	Очищает все переменные, закрывает файлы и т.п. резервирует память под символьные переменные, определяет верхний адрес для Бейсика.
KEY	Переопределяет функциональные клавиши.
TRON	Включает режим трассировки.
TROFF	Выключает режим трассировки.

Команды, влияющие на выполнение программы

CONT	Продолжает выполнение программы после оператора STOP или клавиши «СТОП».
RUN	Начинает выполнение программы.
SYSTEM	Осуществляет переход в режим монитора машины.

Определение параметров системы

CLEAR	Устанавливает размеры области данных под символьные переменные, очищает все значения переменных, устанавливает верхний адрес памяти, доступный системе.
DIM	Определяет, резервирует память и инициализирует массивы.
DEF FN	Определяет пользовательские функции.

DEF USR	Определяет стартовые адреса пользовательских машинных подпрограмм-функций.

Операторы, изменяющие значение переменных

FOR	Оператор цикла; изменяет значение переменной цикла.
LET	Основной оператор присвоения новых значений переменным; слово LET не обязательно.

Числовые функции

ABS( )	Абсолютное значение.
ATN( )	Арктангенс. Результат выдаётся в радианах.
COS( )	Косинус угла, заданного в радианах.
EXP( )	Возведение **e** в степень.
FIX( )	Отбрасывает дробную часть аргумента.
INT( )	Округляет аргумент до целого числа, не превосходящего аргумента.
LOG( )	Натуральный логарифм.
PI	Число «ПИ».
RND( )	Псевдослучайное число.
SGN( )	Функция знака.
SIN( )	Синус угла, заданного в радианах.
SQR( )	Квадратный корень.
TAN( )	Тангенс угла, заданного в радианах.

Функции преобразования типов

CINT( )	Преобразовывает результат выражения в целый тип.
CSNG( )	Преобразовывает результат выражения в вещественный тип.

Символьные функции

ASC( )	Переводит символ в числовой код КОИ-8.
BIN$( )	Преобразовывает целое число в строку, содержащую двоичное представление числа.
CHR$( )	Переводит числовой код КОИ-8 в символ.
HEX$( )	Преобразовывает целое число в строку, содержащую 16-ричное представление числа.
INSTR( )	Номер позиции вхождения подстроки в строку.
LEN( )	Текущая длина символьного выражения.
MID$( )	Выбирает часть символьного выражения.
MID$()=	Присваивает результат символьного выражения части символьной переменной.
OCT$( )	Преобразовывает целое число в строку, содержащую восьмеричное представление числа.
SPC( )	Функция, входящая в команду PRINT. Выводит указанное количество пробелов.
STRING$()	Формирует строку повторяющегося указанного символа.
STR$( )	Преобразовывает число в символьную строку, содержащую число.
TAB( )	Функция, входящая в команду PRINT. Выводит пробелы до

указанной позиции.

VAL( ) Преобразует часть символьной переменной, содержащей цифры, в число.

Операторы, управляющие выполнением программы

CALL Вызов внешних модулей ПЗУ.

END Оканчивает выполнение программы.

FOR Начинает выполнение цикла.

GOSUB Вызывает подпрограммы.

GOTO Производит переход на новую строку программы.

IF В зависимости от условия выполняет одну из ветвей.

NEXT Оканчивает цикл, начатый оператором FOR.

RETURN Оканчивает подпрограмму и осуществляет возврат к следующему за GOSUB оператору.

ON Оператор выбора, выполняющий безусловный переход или вызывающий подпрограммы в зависимости от значения выражения.

STOP Временно приостанавливает выполнение программы и выдаёт сообщение.

Операторы инициализации переменных

DATA Определяет список внутренних данных.

READ Ввод данных из операторов DATA в переменные.

RESTORE Устанавливает ввод с определённого оператора DATA.

Операторы комментария

REM	Оператор для комментария. Весь текст в строке после этого знака рассматривается как комментарий.

Работа с памятью и регистрами внешних устройств

CLEAR	Определение верхнего адреса ОЗУ, доступного Бейсик-системе, резервирование памяти для символьных переменных
FRE( ) FRE	Результат в зависимости от типа аргумента – объём свободной области памяти (аргумент арифметический или отсутствует) или свободной области памяти, отведённой под символьные переменные (аргумент символьный).
INP( )	Чтение данных по маске.
OUT	Запись данных по маске.
PEEK( )	Чтение данных из памяти.
POKE	Запись данных в память.
USR( )	Вызов пользовательской подпрограммы-функции в машинных кодах.
VARPTR( )	Физический адрес размещения переменной.

Управление экраном дисплея

CLS	Очистка экрана.
COLOR	Переключение цвета фона и текущего цвета букв.
LOCATE	Передвигает курсор, высвечивая или погашая его.
PRINT ?	Выводит данные на экран.
SCREEN	Переключает экраны.

Функции управления курсором

AT( )	Функция, входящая в команду PRINT. Помещает курсор в произвольную позицию экрана.
CSRLIN( ) CSRLIN	Номер строки, на которой находится курсор. Параметр фиктивный.
POS( ) POS	Номер столбца, на котором находится курсор. Параметр фиктивный.
SPC( )	Функция, входящая в команду PRINT. Выводит указанное количество пробелов.
TAB( )	Функция, входящая в команду PRINT. Выводит пробелы до указанной позиции.

Операторы графики

CIRCLE	Рисует окружности, овалы, дуги.
DRAW	Выполняет строки графических команд.
LINE	Чертит линии и прямоугольники.
PAINT	Заполняет замкнутую область любым цветом.
POINT( )	Даёт номер цвета указанной точки экрана.
PRESET( )	Выкрашивает указанную точку в цвет фона.
PSET( )	Выкрашивает указанную точку в указанный цвет.

Управление звуком

BEEP	Выдаёт звуковой сигнал.

Работа с файлами данных

CLOSE	Заканчивает операции ввода (вывода) с файлом на МЛ.

EOF	Функция проверки конца входного файла на магнитной ленте.
INPUT#	Ввод данных из файла на магнитной ленте.
OPEN	Инициализация обмена данными с файлом на МЛ для ввода (вывода).
PRINT#	Вывод данных в файл на магнитной ленте.

Управление клавиатурой

INKEY$	Результат – введённый символ или пустая стока.
INPUT	Вводит строку символов.

Вывод на печатающее устройство

LPRINT	Печать данных на печатающем устройстве.

Операции

Логические операции –	AND, OR, NOT, EQV, IMP, XOR
Бинарные арифметические операции –	~, *, /, +, -, MOD, \
Унарные арифметические операции –	-, +
Бинарная символьная операция (конкатенация) –	+
Логические операции отношения –	= , <>, ><, <, >, <=, =<, =>, >=

Приложение 3.
Зарезервированные в Бейсик-системе слова

Приведённые в списке слова являются ключевыми в Бейсик-системе, поэтому их нельзя использовать как имена переменных. Рядом указывается страница, где описывается данное слово.

ABS	35	BEEP	63
AND	15	BLOAD	73
ASC	40	BSAVE	74
AT	68	BIN$	43
ATN	33		
AUTO	21		
CALL	23	COLOR	54
CHR$	41	CONT	23
			Ошибка! Закладка не определена.
CIRCLE	58	COS	
CINT	40	CSAVE	72
CLEAR	47	CSNG	40
CLOAD	72	CSRLIN	67
CLS	53		
DELETE	20	ELSE	28
DIM	44	END	31
DRAW	60	EOF	67
DATA	45	EQV	16
		EXP	**Ошибка! Закладка не определена.**
FILES	72	GOSUB	27
FIX	35	GOTO	27
FOR	29		

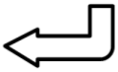
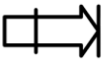
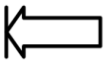
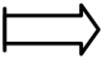
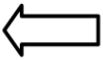
RENUM	20	SCREEN	53
RETURN	27	SGN	36
RND	36	SIN	32
RUN	22	SQR	32
RESTORE	46	STEP	30
		STOP	24
		STR$	42
		SYSTEM	23
		STRING$	39
TAB	69	VAL	42
TAN	33	VARPTR	77
THEN	28		
TO	30		
TROFF	25		
TRON	25		
XOR	16		

Примечание. В более поздних версиях Бейсик-системы зарезервированных слов может быть больше.

Приложение 4.
Особенности версии 1987.12.25

Стр.	Напечатано	Следует читать
25	3.6. Оператор TRON 3.7. Оператор TROFF	В данной версии не реализованы операторы трассировки номеров строк TRON/TROFF.
57	9.2.4. Оператор LINE	В данной версии не реализованы параметры B и BF оператора LINE. Прямоугольники можно вычертить отрезками линий
60	9.2.7. Оператор DRAW	В данной версии оператор DRAW не реализован.

Приложение 5.
Таблица соответствия клавиш клавиатуры и обозначения клавиш в тексте

Условное обозначение клавиш в тексте	Клавишная клавиатура
←–/, "ВВОД", "ВК", "ПС"	
–|→	
|←	
|→	
←–, "ЗАБОЙ"	
→	
↑	
↓	
←	
⇓