ЭЛЕКТРОНИКА

БК 0010-01

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ микро – ЭВМ

язык «БЕЙСИК»

описание языка

АННОТАЦИЯ

Данный документ представляет собой руководство по языку программирования Бейсик и предназначен для пользователей микро-ЭВМ "Электроника БК 0010" (или её модификаций).

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 и руководством оператора по Бейсик-системе.

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.

1.1. 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 "<--/". 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.

1.2. 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.

1.3. СООБЩЕНИЯ ОБ ОШИБКАХ

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

```
ОШИБКА XX В СТРОКЕ YYYYY
```

Где XX – код ошибки, YYYYY – номер строки, содержащей ошибку.

В режиме непосредственного выполнения номер строки не указывается.

Коды ошибок приведены в приложении 1.

1.4. 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
```

1.5. 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
/	Наклонная черта
^	Надчёркивание или знак возведения в степень
(	Открывающая скобка
)	Закрывающая скобка
,	Запятая
;	Точка с запятой
.	Точка
"	Кавычки
'	Апостроф
$	Знак денежной единицы
<	Меньше
>	Больше
?	Знак вопроса
@	Знак "относительно"
\	Символ целого деления
&	Амперсанд
!	Восклицательный знак
%	Процент
#	Знак числа
_	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, SAVE, LOAD, RUN;
- имена операторов и разделителей, например: LET, IF, THEN, GOTO, GOSUB, FOR, NEXT;

- имена стандартных и символьных функций, например: SIN, COS, TAN, EXP, LOG, SQR.

Полный список зарезервированных слов приводится в приложении 4.

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.

2. ДАННЫЕ И ОПЕРАЦИИ НАД НИМИ

2.1. КОНСТАНТЫ

В Бейсик-системе используются числовые и текстовые константы.

2.1.1. 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
```

Для обозначения констант двойной точности вместо буквы E используется буква D.

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
```

2.1.2. ТИПЫ ЧИСЛОВЫХ КОНСТАНТ

По внутреннему представлению числовые константы подразделяются на целые, вещественные одинарной точности и вещественные двойной точности.

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 могут быть представлены с одинарной или двойной точностью. Для констант одинарной точности хранятся 7 десятичных цифр, а для двойной – 17 и выделяются соответственно 2 или 4 машинных слова. По умолчанию в Бейсик-системе точность для констант принимается двойной.

Константа одинарной точности различается по признакам:

- для её записи используется экспоненциальная форма с буквой E;
- при её записи используется восклицательный знак ! .

Примеры:

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

Константа двойной точности различается по признакам:

- любая запись числа без указания типа;
- экспоненциальная форма записи с буквой D;

- признак числа двойной точности #.

Примеры:

```
3489
3451415.92
-1.09432D-09
348.9#
```

2.1.3. ТЕКСТОВАЯ КОНСТАНТА

Формальное описание текстовой константы:

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

Текстовая константа – последовательность алфавитно-цифровых, специальных символов, символов полуграфики (цепочка знаков), заключённая в кавычки. Количество символов не должно превышать 255.

Символ пробела внутри цепочки знаков является значащим.

Примеры:

```
"ПРИВЕТ"
"ЭЛЕКТРОНИКА БК0010"
```

2.2. ПЕРЕМЕННЫЕ

2.2.1. ОПРЕДЕЛЕНИЕ ПЕРЕМЕННОЙ

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 одинарной точности;
#	двойной точности.

При отсутствии знака считается, что переменная двойной точности.

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

Примеры:

```
L#
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. 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. Для объявления массивов используется оператор DIM.

2.3. 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.

2.3.1. АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ

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

                                [*]
<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].

2.3.2. 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
```

2.3.3. 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.

2.3.4. 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.

Приведём таблицы, определяющие логические операции:

```
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

2.3.5. ФУНКЦИИ

Функции – это заранее определённые операции над данными. Бейсик-система имеет "встроенные" функции, которые называются стандартными. Примером таких функций могут служить SQR( ) – квадратный корень или SIN( ) – синус. Более подробно стандартные функции описываются далее.

2.3.6. 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 Во время вычисления значения арифметического выражения все операнды преобразуются в один тип данных – тот, к которому относится операнд, имеющий наибольшую точность.

Пример:

```
10 D=6/7!
20 PRINT D
RUN
.85714285714285714
OK
```

(Арифметическое выражение было подсчитано с двойной точностью).

Пример:

```
10 D!=6/7!
20 PRINT D!
.8571429
```

(При присвоении результата арифметического выражения переменной D! произошло округление из-за разницы в типах).

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

4 При преобразовании типов из двойной точности в одинарную происходит округление. При преобразовании в целый тип отбрасывается дробная часть.

Пример:

```
10 C%=55.88
20 C!=1.23456789
30 PRINT C%
40 PRINT C!
RUN
55
1.234568
OK
```

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

2.3.7. 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 символов, взятыми из соответствующих строк. Если все коды совпадают, значит строки равны. Если коды символов различаются, то меньше та строка, у которой код рассматриваемого символа меньше. Если одна строка кончается, а другая – нет, то более короткая строка меньше (пробелы учитываются).

Примеры (результат всех выражений – "Истина"):

"AA" <> "AB"
"A" = "A"
"X&" > "X"
"CL " > "CL"
B$ < "9.12.84.", если B$="8.12.8"
"AAC" < "ABB"

3. КОМАНДЫ СИСТЕМЫ

Команды Бейсика служат для связи программиста с системой. Они не имеют номера вводимой строки и выполняются только в непосредственном режиме.

3.1. КОМАНДЫ ПУСКА ПРОГРАММЫ

3.1.1. КОМАНДА RUN

Команда начинает выполнение программы.

Формат:

```
RUN [<АРГУМЕНТ>]
```

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
RUN 100
```

(Начинает выполнение программы со строки с номером 100)

3.1.2. 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, END или при нажатии клавиши "СТОП". Команда CONT продолжает выполнение программы с того места, на котором оно было прервано. На самом деле выполняется команда 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.

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.

Пример:

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

3.1.3. КЛАВИША (КОМАНДА) "ШАГ"

Клавишей "ШАГ" программу можно выполнить в шаговом режиме, т.е. по одной строке (см. "Бейсик. Руководство оператора").

3.1.4. КОМАНДА MONIT

Команда осуществляет переход в мониторную систему микро-ЭВМ.

Формат:

```
MONIT
```

Команда используется для передачи управления мониторной системе микро-ЭВМ.

3.2. ХРАНЕНИЕ ПРОГРАММ НА КАССЕТНОЙ МАГНИТНОЙ ЛЕНТЕ

Программы пользователей хранятся на кассетной магнитной ленте. Описываемые дальше команды предназначены для чтения (записи) программ с (на) магнитной ленты. Так же имеются команды 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.

Бейсик-система допускает работу с тремя типами файлов (в скобках дано их условное обозначение):

- Текстовые файлы (.ASC);
- Файлы во внутреннем формате системы (.COD);
- Двоичные файлы (.BIN).

Бейсик-система присваивает тип файлу при записи его на магнитную ленту и показывает тип рядом с именем файла при чтении информации с ленты.

При использовании Бейсик-системы в микро-ЭВМ "Электроника БК0010Ш" команды, описанные в этом разделе, позволяют также осуществлять обмен информацией между рабочим местом ученика и рабочим местом преподавателя (см. "Комплекс учебной вычислительной техники. Руководство оператора").

3.2.1. КОМАНДА LOAD

Команда загружает текст программы пользователя.

Формат:

```
LOAD "<СПЕЦИФИКАЦИЯ ФАЙЛА>[,R]
```

При наличии R программа после её загрузки запускается на выполнение.

Команда используется для загрузки текста программы, размещенного на кассетной магнитной ленте с помощью команды SAVE. Наличие параметра R приводит к немедленному запуску программы после её загрузки. Текст программы представлен на магнитной ленте в кодах символов микро-ЭВМ (файл типа .ASC).

Пример:

```
LOAD "MOD2",R
MOD2 .ASC 003
OK
```

3.2.2. КОМАНДА SAVE

По этой команде текст программы пользователя записывается на кассетную магнитную ленту.

Формат:

```
SAVE "<СПЕЦИФИКАЦИЯ ФАЙЛА>
```

Эта команда предназначена для записи текста программы на кассетную магнитную ленту в кодах символов микро-ЭВМ.

Главное применение команды SAVE – сохранить текст программы пользователя на магнитной ленте в файле типа .ASC.

Пример:

```
FIND "ПЕРВЫЙ"
ПЕРВЫЙ.COD
OK
SAVE"ВТОРОЙ"
OK
```

3.2.3. КОМАНДА MERGE

Команда 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 .ASC.

В случае совпадения номера загружаемой строки с номером строки, хранящейся в памяти, при объединении текстов оставляется загруженная строка.

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
```

3.2.4. КОМАНДА CLOAD

Команда загружает с кассетой магнитной ленты программу пользователя, записанную во внутреннем формате.

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
```

3.2.5. КОМАНДА CSAVE

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

Формат:

```
CSAVE "<СПЕЦИФИКАЦИЯ ФАЙЛА>
```

По этой команде текст программы записывается во внутреннем представлении системы на магнитную ленту.

Пример:

```
CSAVE "1001"
OK
```

3.2.6. КОМАНДА FIND

Команда предназначена для поиска файла, записанного на кассетную магнитную ленту.

Формат:

```
FIND [<"СПЕЦИФИКАЦИЯ ФАЙЛА>]
```

Команда FIND позволяет подвести головку магнитофона к концу любого записанного на ленту файла. Имя нужного файла указывается аргументом. Команду удобно использовать при указании свободного участка магнитной ленты для записи нового файла.

Как и команда LOAD, эта команда при чтении выдаёт на экран имена всех обнаруженных файлов и их типы.

Если имя файла не указано, то команда просто выдаёт на экран имена всех встреченных файлов.

При выполнении команды содержание ОЗУ не меняется. Выполнение команды можно прервать нажатием клавиши "СТОП".

Команду можно использовать и для пуска двигателя магнитофона при необходимости перемотать ленту вперёд или назад.

Пример:

```
FIND"ПЕРВЫЙ"
ПЕРВЫЙ.COD
```

```
OK
SAVE"ВТОРОЙ"
OK
```

3.2.7. КОМАНДА BLOAD

Загружает "двоичный" файл (машинный код или данные) в память.

Формат:

```
BLOAD "<АРГУМЕНТ1>[,R][,<АРГУМЕНТ2>]
```

<АРГУМЕНТ1>::= спецификация файла;

R является признаком немедленного запуска программы после её загрузки;

<АРГУМЕНТ2>:: = адрес начальной загрузки файла.

Команда загружает содержимое файла, записанного на кассетную магнитную ленту с помощью команды BSAVE (файл типа .BIN). В файле хранятся начальный адрес и длина массива, согласно которым содержимое файла размещается в памяти машины.

При наличии второго аргумента, т.е. R, после загрузки содержимого файла происходит немедленный запуск программы с начального адреса загрузки. Указав третий аргумент, можно разместить содержимое файла в памяти, начиная с произвольного адреса оперативной памяти. Адрес может быть целой константой из интервала [0, 65535], а также шестнадцатеричной, восьмеричной или двоичной константой. При отсутствии аргумента R для указания аргумента 2 дополнительная запятая не требуется.

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

Команда позволяет загрузить программы в машинном представлении в оперативную память с целью их дальнейшего использования посредством функции USR. Отметим, что при загрузке в постоянную память эта команда никаких действий не производит.

Пример:

```
CLEAR 200,&HF40
OK
BLOAD "DATA",&HF40
DATA  .BIN
OK
BLOAD "PROG",R
PROG  .BIN
```

(Файл PROG загружен и запущен с начала).

3.2.8. КОМАНДА BSAVE

Записывает содержимое оперативной памяти на кассетную магнитную ленту.

Формат:

```
BSAVE "<АРГУМЕНТ1>,<АРГУМЕНТ2>,<АРГУМЕНТ3>
```

<АРГУМЕНТ1>::= Спецификация файла;
<АРГУМЕНТ2>::= Адрес начала сохраняемой области;
<АРГУМЕНТ3>::= Адрес конца сохраняемой области.

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. Аргумент 1 и аргумент 2 могут быть целыми константами из интервала [0, 65535], а также шестнадцатеричными, восьмеричными или двоичными константами. 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.

В основном эта команда даёт возможность сохранять содержимое оперативной памяти на магнитной ленте в двоичном формате (файл типа .BIN). Особенно полезно её использование при сохранении программ, написанных на машинном языке и используемых в Бейсик-системе при помощи функции USR.

Пример:

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

3.3. РАБОТА С ТЕКСТАМИ ПРОГРАММ

При работе с текстами программ используется понятие номера строки – целой константы из интервала от 0 до 65535. Вместо номера строки можно использовать символ ".", означающий "текущую" строку. "Текущая" строка – это строка, которая последней была редактирована, выведена на экран, введена с клавиатуры или строка, в которой была обнаружена ошибка.

3.3.1. 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 сообщения об ошибке не выдаются. 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".

Вывод строк можно приостановить одновременным нажатием на клавиши "СУ" и "@". Продолжение действий – нажатие любой клавиши. Как всегда, любые действия прерывает 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
```

3.3.2. 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
```

3.3.3. 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 1, 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
```

3.3.4. 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 "<--/" 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 "<--/". Если редакции строки не требуется, то, нажав клавишу "<--/", можно перейти к вводу следующей строки.

Для окончания режима AUTO достаточно нажать на клавиши "СТОП", "СУ" и "С" одновременно или клавишу "КТ", что возвращает систему в непосредственный режим.

Пример:

```
110 END
AUTO 100
100 PRINT "ПРИМЕР"
110 END (НАЖИМАЕТЕ КЛАВИШУ "<--/")
120     (НАЖИМАЕТЕ КЛАВИШИ "СУ" И "С"
         ОДНОВРЕМЕННО)
LIST
100 PRINT "ПРИМЕР"
110 END
OK
```

3.3.5. 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. "Бейсик. Руководство оператора").

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

3.3.6. КЛАВИША (КОМАНДА) "BC"

Нажатием клавиши "BC" можно вызвать последнюю введённую строку для дальнейшей корректировки и повторного ввода (см. "Бейсик. Руководство оператора.").

3.3.7. КОМАНДА NEW

По этой команде удаляется программа пользователя.

Формат:

```
NEW
```

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

4. ОПЕРАТОРЫ

Операторы языка изменяют значения переменных, естественный порядок вычислений согласно алгоритму решения задачи, резервируют память под переменные и константы и т.д.

4.1. 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.

4.1.1. 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
```

4.1.2. 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
```

4.1.3. 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.

4.1.4. ОПЕРАТОР RETURN

Оканчивает подпрограмму и осуществляет возврат к следующему за GOSUB оператору

Формат

```
RETURN
```

RETURN осуществляет возврат к оператору, который находится непосредственно за оператором, вызвавшим подпрограмму. После возвращения все незаконченные циклы FOR-NEXT в подпрограмме завершаются. В подпрограмме возможны вложенные вызовы других подпрограмм

Пример

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

4.1.5. 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
```

<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
```

4.1.6. 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.

Вход в цикл, минуя оператор FOR, не допускается.

4.1.7. 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 для большей наглядности, чтобы указать, 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.

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
```

4.1.8. 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 порядковому номеру одного из номеров строк из списка (считая слева 1, 2 и т.д.), то этот номер строки употребляется оператором GOSUB или GOTO.

Если выполняется оператор GOSUB <НОМЕР СТРОКИ>, 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 A$=INKEY$
20 IF A$="" THEN 10
30 ON ASC(A$)-64% GOSUB 100,200,300
40 GOTO 10
100 PRINT 100
102 RETURN
200 PRINT 200
202 RETURN
300 PRINT 300
302 RETURN
```

4.1.9. 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 YYYYY
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
```

4.1.10. 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 GOSUB 30
20 END
30 PRINT "ПОДПРОГРАММА"
40 RETURN
RUN
ПОДПРОГРАММА
OK
```

4.1.11. ОПЕРАТОР CALL

Вызывает программу из внешнего модуля 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 сменном блоке ПЗУ.

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)
```

4.1.12. ОПЕРАТОР REM

Обозначает комментарии в тексте программы.

Формат:

```
REM [<ТЕКСТОВАЯ КОНСТАНТА>]
```

Оператор служит исключительно только 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
```

4.2. ОПЕРАТОРЫ ОПИСАНИЯ, РЕЗЕРВИРОВАНИЯ ПАМЯТИ И УСТАНОВКИ НАЧАЛЬНЫХ ЗНАЧЕНИЙ

Операторы резервирования памяти служат резервированию оперативной памяти для массивов и постоянных величин. Память для переменных резервируется в локальной или обшей области данных. Память для постоянных величин резервируется в области генерируемой программы при компиляции. Для резервирования памяти переменным и массивам используется оператор DIM. Для резервирования памяти постоянным величинам – DATA. Для присвоения переменным значений постоянных величин используются операторы READ и RESTORE.

4.2.1. 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].

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)
```

4.2.2. 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 16 символов).

Оператор используется для изменения значений функциональных ключей, установленных по умолчанию. Первые 16 символов аргумента 2 присваиваются ключу, указанному аргументом 1.

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

Для ввода символа < <--/ > или кавычек можно воспользоваться функцией CHR$.

Если значение первого аргумента выходит за пределы допустимого интервала, получается ошибка 5.

По умолчанию функциональные клавиши имеют следующие значения:

```
COLOR
```

```
AUTO
GOTO
LIST
RUN <<--/>
COLOR 1,0 <<--/>
CLOAD "
CONT <<--/>
. <<--/>
<СБР> RUN <<--/>
```

(Управляющий символ <СБР> имеет восьмеричный код 14 и означает очистку экрана).

Пример:

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

4.2.3. ОПЕРАТОР CLEAR

Очищает значения переменных, резервирует символьную и свободную от Бейсик-системы 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 70000.

При любой форме оператора очищаются массивы и переменные, определённые функции, символьные строки, а также аппаратный стек, закрываются все файлы, а любая распечатка заканчивается символом <<--/>, 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, 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. Указание второго аргумента при отсутствии первого вызывает ошибку 2. Если аргумент 2 больше восьмеричного 70000, выдаётся сообщение об ошибке 5.

Пример:

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

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

4.2.4. 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.

4.2.5. 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,&O22,&H3F
15 FOR I%=0% TO 5%
20 DATA 234E-7
30 READ X(I%)
40 NEXT I%
30 DATA -33%, 44.56, TEXT, "A,B"
60 READ A$, B$
```

4.2.6. 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 1000
30 END
1000 RESTORE
1010 FOR I%=0% TO 10%
1020 READ X(I%)
1030 NEXT I%
```

```
1040 RETURN
```

(Подпрограмма, заполняющая массив X)

4.2.7. ОПЕРАТОР DEF

4.2.7.1. ОПЕРАТОР DEF USR

Определяет функции на машинном языке.

Формат:

```
DEF USR[<ЦИФРА>]=<ВЫРАЖЕНИЕ>
```

<ЦИФРА>::=	Целая константа из интервала [0,9], указывающая номер функции, по умолчанию 0;
<ВЫРАЖЕНИЕ>::=	Целое выражение, определяющее адрес запуска функции USR.

Оператор определяет начальный адрес машинной подпрограммы, составленной пользователем. Значение выражения переводится в целый тип 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. Адреса вызова подпрограмм хранятся в области USRTAB (&O2100), где каждой USR( ) выделяется по одному слову. Действия оператора можно представить, как заполнение USRTAB.

Для выполнения программы на машинном языке необходимо выполнить последовательность операторов

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

Обычным способом загрузки машинной программы в память является использование команды BLOAD, которая запоминает начальный адрес загруженной программы в ячейке BUFSTA (&O264). Таким образом, логична последовательность

```
BLOAD "PROG"
PROG .BIN
OK
DEF USR5=PEEK(&O264)
OK
```

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

4.2.7.2. ОПЕРАТОР DEF FN

Определяет пользовательские функции.

Формат:

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

<ИМЯ>::=	Любое допустимое имя переменной;
<СПИСОК>::=	Один или несколько имён формальных параметров (допустимых имён переменных), разделённых запятыми;
<ВЫРАЖЕНИЕ>::=	Любое выражение такого же типа, как и <ИМЯ>.

При выполнении этого оператора создаётся переменная <ИМЯ>, отмеченная как имя функции. Никаких вычислений оператор не производит и может находиться в любом месте программы. Переменная <ИМЯ> недоступна обычным путём, и может быть создана другая 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. Этот оператор – один из немногих, запрещённых в непосредственном режиме.

Пример:

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

4.3. ОПЕРАТОРЫ ВВОДА/ВЫВОДА

Операторы ввода/вывода позволяют организовать обмен информацией с внешними устройствами.

4.3.1. 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 .ASC, блоками по 256 символов. Конец файла отмечается символом "СУ/Z" (код &O32). Аналогичным форматом 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 .ASC.

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).

4.3.2. ОПЕРАТОР CLOSE

Заканчивает операции ввода/вывода и освобождает буфер.

Формат:

```
CLOSE
```

Если оператором OPEN был открыт выходной файл, к выводимым данным добавляются символ < <--/ > и указатель конца файла "CУ/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
```

4.3.3. 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 данные не помещаются в зоне, берётся следующая зона. 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
```

4.3.4. 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
```

4.4. ОПЕРАТОРЫ НЕПОСРЕДСТВЕННОГО ДОСТУПА К ПАМЯТИ

4.4.1. ОПЕРАТОР POKE

Записывает данные в любую ячейку памяти ЭВМ.

Формат:

```
POKE <АДРЕС>,<ВЫРАЖЕНИЕ>
```

<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. 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 A%=PEEK(4%)
20 POKE 6%,2%
30 POKE 4%,6%
...
100 POKE 4%,A%
110 POKE 6%,0%
```

(Во время выполнения программы блокируется действие клавиши "СТОП").

4.4.2. ОПЕРАТОР OUT

Записывает данные в оперативную память по маске.

Формат:

```
OUT <АДРЕС>,<МАСКА>,<КОД>
```

<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
```

4.5. ОПЕРАТОРЫ УПРАВЛЕНИЯ ЭКРАНОМ ТЕЛЕВИЗОРА

4.5.1. 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.

Пример:

```
120 CLS
130 PRINT AT(10,10);"КУ-КУ!
140 CLEAR
```

4.5.2. 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 – красный;
2 – зелёный;
3 – синий;
4 – чёрный.

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 @(20,0)
140 DRAW A$
```

4.5.3. 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$.

На положение графического курсора оператор LOCATE не влияет.

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
```

4.6. ОПЕРАТОРЫ И ФУНКЦИИ ГРАФИКИ

4.6.1. 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];
@ - Координаты указанной точки подсчитываются
STEP относительно последней точки, обработанной операторами графики.

Если заданная точка находится за пределами экрана (пределы экрана определяются координатами X – [0,255], Y – [0,240]), 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:

```
10 FOR Y=1 TO 120
```

```
20 PSET(SQR(Y),Y)
30 NEXT
```

4.6.2. 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 [0,255] и [0,240] 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%
```

4.6.3. 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
```

4.6.4. 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 (<АРГ1> и <АРГ2>) и конечной (<АРГ3> и <АРГ4>) точки объекта (для прямоугольника – это точки диагонали). 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, 255] и [0, 240] 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
```

4.6.5. 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 эксцентриситет (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, 255], Y – [0, 240]).

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. 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 (6.283185).

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 COLOR 3
20 CIRCLE(100,100),75,,-1,-.001,1.2
30 CIRCLE(10,-5),75,,-.001,-1,1.2
```

4.6.6. 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.

Точкой, с которой начинается окрашивание, может быть любая внутренняя точка 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
```

4.6.7. 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, 255] и [0, 240] 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
```

Для черчения с указанием координат точек используется команда M.

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

Команда чертит от текущей точки до точки с координатами (<АРГ1>,<АРГ2>). 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$;"
```

4.7. ОПЕРАТОР УПРАВЛЕНИЯ ЗВУКОМ

4.7.1. ОПЕРАТОР BEEP

Оператор выдаёт короткий звуковой сигнал.

Формат:

```
BEEP
```

Этот оператор используется для выдачи короткого звукового сигнала. Этот же сигнал выдаётся системой в случае ошибки.

Пример:

```
100 FOR I=1 TO 10
110 BEEP
120 NEXT
130 INPUT "НОМЕР МЕСЯЦА";MN
140 IF MN>12 OR MN<1 THEN 100
150 PRINT MN
```

4.8. ОПЕРАТОРЫ ТРАССИРОВКИ ПРОГРАММЫ

4.8.1. ОПЕРАТОР TRON

Запускает трассировку номеров строк.

Формат:

```
TRON
```

В режиме трассировки номер каждой выполняемой строки программы выдаётся на экран в формате [NNN]. 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.

4.8.2. 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
```

5. ФУНКЦИИ

Язык Бейсик позволяет использовать аппарат функций. Различают стандартные функции, которые заранее определены в системе, и внутренние, которые определяет сам пользователь. Стандартные в свою очередь подразделяются на числовые, предназначенные для работы с числовыми величинами, и символьные, предназначенные для действий над символьными строками.

5.1. ЧИСЛОВЫЕ ФУНКЦИИ

Результаты всех числовых функций – числа двойной точности.

5.1.1. 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)
1.7320508075688773
OK
```

5.1.2. ФУНКЦИЯ SIN

Функция вычисляет синус угла, заданного в радианах.

Формат:

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

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

Пример:

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

5.1.3. ФУНКЦИЯ COS

Функция вычисляет косинус угла, заданного в радианах.

Формат:

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

Пример:

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

5.1.4. 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)
```

5.1.5. 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
3.1415926535897932
OK
```

5.1.6. ФУНКЦИЯ PI

Результат функции – число "пи".

Формат:

```
X=PI
```

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

```
PRINT PI
3.1415926535897932
OK
PRINT SIN(PI/2)
1
OK
```

5.1.7. ФУНКЦИЯ EXP

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

```
X=EXP(<АРГУМЕНТ>)
```

<АРГУМЕНТ>::= Арифметическое выражение, принимающее значения из интервала [-88.499999999999992, 88.029685974121093].

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)).
Пример:

```
PRINT EXP(LOG(3))
2.9999999999999911
OK
```

5.1.8. 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))
```

```
.9999999999999793
OK
```

5.1.9. 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
2.6666666666666667
OK
```

5.1.10. 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
IF X/N=FIX(X/N) THEN PRINT "X ДЕЛИТСЯ НА N"
OK
```

5.1.11. ФУНКЦИЯ INT

Результат функции – целое значение, меньшее, чем аргумент.
Формат:

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

Функция выдаёт ближайшее целое число, не превосходящее аргумент. Функция INT( ) подобна 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:

```
IF ABS(INT(Z))=ABS(INT(-Z)) THEN PRINT "Z-ЦЕЛОЕ"
OK
PRINT INT(-5.3), FIX(-5.3)
-6               -5
OK
```

5.1.12. 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:

```
100 INPUT A
110 PRINT B*SGN(A)
```

5.1.13. 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)
.47099434036014261
.53368281796108777
.53368281796108777
OK
PRINT RND(3),RND(0)
.60542409693567598
.60542409693567598
OK
```

5.1.14. ФУНКЦИЯ FRE

Указывает, сколько осталось свободной памяти компьютера для программы пользователя.

Формат:

```
X=FRE[(<АРГУМЕНТ>)]
```

<АРГУМЕНТ>::= Либо символьное, либо 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.

Пример:

```
PRINT FRE(" "),FRE(0)
200              11000
```

5.2. ФУНКЦИИ ПРЕОБРАЗОВАНИЯ ТИПОВ

5.2.1. 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=<X=<32767], то выдаётся ошибка переполнения.

Функция используется по умолчанию, когда значение арифметического выражения присваивается переменной целого типа.

Примеры:

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

5.2.2. ФУНКЦИЯ CSNG

Функция преобразует аргумент в число одинарной точности, округляя его.

Формат:

```
X=CSNG(<АРГУМЕНТ>)
<АРГУМЕНТ>::= <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. Функцию удобно использовать, когда временно нужно значение переменной одинарной точности, например, для печати.

Примеры:

```
PRINT CSNG(PI)
3.141593
OK
```

5.2.3. ФУНКЦИЯ CDBL

Функция преобразует результат арифметического выражения в число двойной точности.

Формат:

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

Функция предназначена для преобразования значений арифметических выражений в числа двойной точности, т.е. с мантиссой из 17-ти десятичных цифр. Эта функция используется по умолчанию, когда происходит присвоение переменной двойной точности.

Пример:

```
PRINT CDBL(2/3)
.6666666666666667
OK
```

5.3. ФУНКЦИИ НЕПОСРЕДСТВЕННОГО ОБРАЩЕНИЯ К ПАМЯТИ

5.3.1. ФУНКЦИЯ PEEK

Позволяет прочесть содержимое любой ячейки памяти.

Формат:

```
X=PEEK(<АРГУМЕНТ>)
```

<АРГУМЕНТ>::= Целое выражение, задающее адрес ячейки.

Эта функция позволяет исследовать любое определённое место памяти в адресном пространстве. 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. Значения аргумента от -32768 до -1 обозначают адреса от 32768 до 65535 соответственно.

Пример:

```
PRINT PEEK(&O176),PEEK(&O200)
100               128
OK
```

(Печатаются значения координат графического курсора).

5.3.2. ФУНКЦИЯ INP

Разрешает чтение данных в оперативной памяти по маске.

Формат:

```
X=INP(<АДРЕС>,<МАСКА>)
```

<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
```

(Программа печатает коды клавиш клавиатуры).

5.4. СИМВОЛЬНЫЕ ФУНКЦИИ

5.4.1. ФУНКЦИЯ ASC

Функция выдаёт значение кода первого символа строки символов.
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 кодов, приведённой в приложении 3. 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
```

5.4.2. ФУНКЦИЯ CHR$

Функция переводит целое число 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 кодов и преобразовывается в символьную строку из одного символа, соответствующего этому коду. Таким образом, CHR$(65) и "A" тождественны. Функция CHR$( ) 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 многосимвольными 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 многосимвольных строк также можно использовать функцию MID$. Заметим, что существуют символы, которые можно занести в строку только при помощи функции CHR$, например:

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

(Выдаётся звуковой сигнал).

5.4.3. 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:

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

Пример:

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

5.4.4. 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 <АРГ1>, 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 Al$
RUN
ПЕРВЫЙ ФАЙЛ
OK
```

5.4.5. 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 символа которым заполняется новая символьная строка;

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

Предназначением функции STRING$( ) является создание символьной переменной любой длины, содержащей одинаковые символы. Вы можете определить этот символ либо задав его по таблице кодов (см. Приложение 3), либо указав нужный символ с помощью символьного выражения, в частности текстовой константы. Тип второго аргумента функции STRING$ 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 вами длина символьной строки равна 0, то будет создана пустая строка, несмотря на инициализирующий символ.

Пример:

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

5.4.6. 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. Это число может включать знак, десятичную точку и обозначение порядка. Функция VAL( ) даёт значение двойной точности. 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, но VAL("23A12:") – число 23.

Функция VAL является обратной к функции STR$( ).

Пример:

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

5.4.7. 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).

5.4.8. ФУНКЦИЯ STR$

Превращает числовые данные в строку символов.

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$. (Не путайте STR$( ) с функцией STRING$( )!). Например, если N=1/3, то STR$(N) даст строку ".33333333333333333" 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 компоновки 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)
3
OK
```

5.4.9. 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$( ) – 16 символов. Главным образом BIN$( ) используется для печати двоичного представления чисел.

Пример:

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

5.4.10. 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$. Пробелов не создаётся, незначащие нули отбрасываются. Таким образом, аргумент 12 даёт результат "14", т.е., строку из двух символов "1" и "4" (восьмеричные коды 61 и 64 соответственно). Максимальная длина строки, полученной при помощи OCT$, – шесть символов.

Функция используется для печати восьмеричного представления чисел.

Пример:

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

5.4.11. 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 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
```

5.5. ВСПОМОГАТЕЛЬНЫЕ ФУНКЦИИ ВВОДА/ВЫВОДА

5.5.1. 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).

5.5.2. 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
```

5.5.3. 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
```

5.5.4. 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 ("Истина"), в противном случае – 0 ("Ложь"). 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
```

5.6. ФУНКЦИИ ОПЕРАТОРА PRINT

5.6.1. 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
```

5.6.2. 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 той же строке, т.е. TAB(1) совпадает с TAB(33), TAB(65) и т.д.

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
```

5.6.3. 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)
```

5.7. ФУНКЦИИ, ОПРЕДЕЛЁННЫЕ ПОЛЬЗОВАТЕЛЕМ

5.7.1. 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)
```

5.7.2. ФУНКЦИЯ 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, двойной точности,

1 – число с плавающей запятой одинарной точности.

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

В вершине стека находится адрес возврата, а за ним – значение аргумента. Таким образом в R5 хранится значение SP+2. 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.

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%)
```

ПРИЛОЖЕНИЕ 1. СООБЩЕНИЯ ОБ ОШИБКАХ

В сообщениях об ошибках указывается код ошибки в виде:

```
ОШИБКА XX В СТРОКЕ YYYYY,
```

Где XX – код ошибки, YYYYY – номер строки, содержащей ошибку.

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 Исчерпан список оператора DATA
При выполнении оператора READ обнаружено, что список оператора DATA исчерпан.

5 Недопустимое использование функции

6 Переполнение
Результат арифметической операции не может быть записан в формате, принятом для чисел в Бейсик системе. (В случае потери порядка результат приравнивается нулю и ошибки не выдаётся).

7 Переполнение памяти
Программа не помещается в памяти, либо использовано слишком много операторов FOR или переменных.

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

9 Недопустимый индекс (за пределами)
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 Недопустимый оператор в режиме непосредственного выполнения.

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

14 Не хватает места символьным переменным
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 Не определена

17 Продолжение выполнения программы невозможно.
Была сделана попытка продолжить выполнение программы, которая:
1. была прервана из-за возникновения ошибки;
2. была изменена;
3. не существует.

18 Не определена функция пользователя
Попытка обратиться к функции USR до её определения оператором DEF.

19 Ошибка устройства 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 Ошибочный номер файла
Оператор или команда ссылается на файл, который не открыт; или номер файла выходит за первоначально определённый предел номеров файлов.

53 Не определена

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

55 Попытка прочитать запись окончания 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 Ошибочное имя файла
В операторах LOAD, SAVE или OPEN использовано неправильное имя файла (имя файла состоит из слишком большого количества

символов).

57 Команда непосредственного выполнения в программе во время загрузки файла типа .ASC. Загрузка заканчивается.

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. (в данной версии языка отсутствует).

63-255 Не определены

ПРИЛОЖЕНИЕ 2. СПИСОК КОМАНД, ОПЕРАТОРОВ И ФУНКЦИЙ

В приложении приводятся команды, операторы и функции Бейсик системы, 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 (формат .BIN).
BSAVE	Записывает блок памяти на магнитную ленту в двоичном коде (формат .BIN).
CLOAD	Загружает программу с магнитной ленты во внутреннем формате Бейсик-системы (формат .COD).
CSAVE	Записывает программу на магнитную ленту во внутреннем формате (формат .COD).
LOAD	Загружает текст программы с магнитной ленты (формат .ASC).

SAVE Записывает текст программы на магнитную ленту (формат .ASC).

MERGE Загружает программу (файл в формате .ASC) с магнитной ленты и объединяет её с программой в ОЗУ.

FIND 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.

MONIT Осуществляет переход в режим монитора машины.

<ШАГ> Осуществляет шаговый режим выполнения программы.

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	Определение пользовательских функций.
DEF USR	Определяет стартовые адреса пользовательских машинных подпрограмм-функций.

Операторы, изменяющие значение переменных

FOR	Оператор цикла. Меняет 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.
CDBL( )	Преобразование результата выражения в тип двойной точности.
CINT( )	Преобразование результата выражения в целый тип.
COS( )	Косинус угла, заданного в радианах.
CSNG( )	Преобразование результата выражения в тип одинарной точности.
EXP( )	Возведение e в степень.
FIX( )	Отбрасывает дробную часть аргумента.
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).

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.

Символьные функции

ASC( )	Переводит символ в числовой код.
BIN$( )	Преобразование целого числа в строку, содержащую двоичное представление числа.
CHR$( )	Переводит числовой код в символ.
HEX$( )	Преобразование целого числа в строку, содержащую 16-ричное представление числа.
LEN( )	Текущая длина символьного выражения.
MID$( )	Выбор части символьного выражения.
MID$()=	Присвоение результата символьного выражения части символьной переменной.
OCT$( )	Преобразование целого числа 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	Вызов подпрограммы.
GOTO	Переход на новую строку программы.
IF	В зависимости от выполнения условия выполняется одна из ветвей оператора IF.
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

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.

Управление экраном дисплея

CLS	Очистка экрана.
COLOR	Переключение цвета фона и текущего цвета символов.

LOCATE	Передвигает курсор, высвечивая или погашая его.
PRINT ?	Выводит данные на экран.

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 магнитной ленте.
EOF	Функция проверки конца входного файла на магнитной ленте.
INPUT#	Ввод данных из файла на магнитной ленте.
OPEN	Инициализация обмена данными с файлом на магнитной ленте 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

AND, OR, NOT, EQV, IMP, XOR	Логические операции
~, *, /, +, -, MOD, \	Бинарные арифметические операции
-, +	Унарные арифметические операции

+	Бинарная символьная операция (конкатенация)
= , <>, >, <, >, <=, =<, =>, >=	Логические операции отношения

ПРИЛОЖЕНИЕ 3. ТАБЛИЦА КОДОВ СИМВОЛОВ

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		СБР ТАБ *	ПРО БЕЛ	0	@	P	`	p		ШАГ	¶	┤	ю	п	Ю	П
1			!	1	A	Q	a	q	ПОВТ **	К	┴	←	а	я	А	Я
2		⇖	"	2	B	R	b	r	ИСУ *	З	♥	╬	б	р	Б	Р
3	КТ	⇑	#	3	C	S	c	s		С	┐	↑	ц	с	Ц	С
4		⇓	$	4	D	T	d	t	БЛР *	Ч	╡	♣	д	т	Д	Т
5		⇓	%	5	E	U	e	u		ГРАФ	├	─	е	у	Е	У
6		⇦	&	6	F	V	f	v		ЗАГ	└	╫	ф	ж	Ф	Ж
7	ЗВ	⇨	'	7	G	W	g	w		СТИР	═	│	г	в	Г	В
8	←	⇦	(	8	H	X	h	x			╤	♦	х	ь	Х	Ь
9	З	→	)	9	I	Y	i	y	ТАБ **	⇨	♠	┘	и	ы	И	Ы
A	←/	↑	*	:	J	Z	j	z		КУР СОР *	┌	╪	й	з	Й	З
B		↓	+	;	K	[	k	{		32/64 *	┬	╥	к	ш	К	Ш
C	СБР	↖	,	<	L	\	l	|	РП	ИНВС	╨	╧	л	э	Л	Э
D	УСТ ТАБ	↗	-	=	M	]	m	}		ИНВЭ *	↓	╞	м	щ	М	Щ
E	РУС **	↘	.	>	N	^	n	~		УСТ ИНД *	┼	→	н	ч	Н	Ч
F	ЛАТ **	↙	/	?	O	_	o	ЗБ		ПОДЧ	╟	▒	о	ъ	О	Ъ

Примечание. * – коды передаются с драйвера клавиатуры на драйвер ТВ-приёмника, минуя внешнюю программу. ** – коды используются только драйвером клавиатуры.

ПРИЛОЖЕНИЕ 4. 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	60	BEEP	56
AND	15	BLOAD	24
ASC	65	BSAVE	24
AT	73		
ATN	58		
AUTO	28		
CDBL	63	COLOR	47
CHR$	65	CONT	20
CINT	62	COS	58
CIRCLE	52	CSAVE	23
CLEAR	38	CSNG	63
CLOAD	22	CSRLIN	71
CLS	47		
DELETE	26	ELSE	31
DIM	36	END	35
DRAW	54	EOF	73
		EXP	59
FIND	23	GOSUB	31
FIX	60	GOTO	30
FOR	32		
FRE	62		
HEX$	71	IF	31
		IMP	16
		INKEY$	69

		STR$	69
TAB	74	VAL	68
TAN	58		
THEN	31		
TO	32		
TROFF	56		
TRON	56		
XOR	16		

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

ПРИЛОЖЕНИЕ 5. ОСОБЕННОСТИ ВЕРСИИ 1986.07.24

1 В данной версии не реализованы:
- Команда MERGE
- Команда KEYLIST
- Функция SPC
- Закрашивание прямоугольника BF
- Автозапуск программы после загрузки R

2 В данной версии операция возведения в степень при основании степени, меньшей или равной нулю, выдаёт сообщение об ошибке 5.

3 Верхняя граница ОЗУ пользователя может быть увеличена с &O40000 до &O70000 (переключение в режим расширенной памяти осуществляется восьмеричным кодом 214, который можно подать с клавиатуры при одновременном нажатии клавиши "АР2" и "СБР").

Рабочая область ОЗУ при этом увеличивается с 16 кбайт до 28 кбайт за счёт уменьшения количества отображаемых строк на экране с 24 до 4.

Возможность расширения объёма рабочей области ОЗУ может быть использована в операторах CLEAR и DEF USR.