БАЗОВОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ БЫТОВОЙ ПЕРСОНАЛЬНОЙ ЭЛЕКТРОННОЙ ВЫЧИСЛИТЕЛЬНОЙ МАШИНЫ ВЕКТОР-06Ц

ИНТЕРПРЕТАТОР ЯЗЫКА БЕЙСИК

ОПИСАНИЕ ЯЗЫКА

УТВЕРЖДЕН

МВАИ 00030-01 35 01-ЛУ

БАЗОВОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ БЫТОВОЙ ПЕРСОНАЛЬНОЙ ЭЛЕКТРОННОЙ ВЫЧИСЛИТЕЛЬНОЙ МАШИНЫ "ВЕКТОР-06Ц"

Интерпретатор языка БЕЙСИК

Описание языка

МВАИ 00030-01 35 01

листов 107

1988

АННОТАЦИЯ

Одним из самых распространенных языков программирования у пользователей персональных компьютеров является БЕЙСИК. Популярность БЕЙСИКА объясняется его простотой, ориентацией на диалоговый характер программирования.

В настоящем руководстве описано программирование на весьма мощной версии интерпретатора БЕЙСИК, реализованной на БПЭВМ "ВЕКТОР-06Ц". Отличительной особенностью интерпретатора БЕЙСИК для БПЭВМ "ВЕКТОР-06Ц" является наличие мощного аппарата графических средств и управления цветом, что может быть использовано в самых разнообразных приложениях человеческой деятельности.

В руководстве основное внимание уделяется содержательному описанию директив, операторов и функций языка БЕЙСИК, особенностям их использования и техническим приемам программирования.

Раздел 2 содержит информацию о загрузке и запуске интерпретатора.

В разделе 3 изложено краткое описание директив, операторов и функций интерпретатора в соответствии с выполняемыми ими функциями.

Раздел 4 содержит общие сведения о языке БЕЙСИК и методы ввода и редактирования текстов программ.

В разделе 5 приводится полное описание всех ключевых слов языка БЕЙСИК.

В разделе 6 рассматриваются приемы программирования.

В разделе 7 приведены сообщения об ошибках, выдаваемых интерпретатором в процессе выполнения программы.

В приложениях приведена справочная информация, необходимая программисту:

- список зарезервированных слов;
- таблицы соответствия клавиш служебным словам;
- распределение памяти
- таблица кодов КОИ-7.

Изложение руководства позволяет без специальной подготовки освоить программирование на языке БЕЙСИК пользователю - непрофессионалу

СОДЕРЖАНИЕ

1. Введение 5
2. Загрузка интерпретатора языка БЕЙСИК 6
3. Классификация директив, операторов и функций 7
3.1. Изменение текста программы в памяти и вывод текста 7
3.2. Директивы чтения и записи на магнитную ленту 7
3.3. Управление выполнением программ 8
3.4. Начальная загрузка изменяемых параметров 8
3.5. Числовые функции 8
3.6. Строковые функции 9
3.7. Изменение порядка работы программ 9
3.8. Работа с данными, хранящимися в тексте программ 10
3.9. Комментарии в программе 10
3.10 Доступ к памяти и ввод-вывод 10
3.11. Управление экраном телевизора 10
3.12. Графические операторы 11
3.13. Управление звуком 11
3.14. Ввод с клавиатуры 11
3.15. Вывод на печать 11
3.16. Операция 11
4. Общие сведения о языке БЕЙСИК 13
4.1. Режимы работы 13
4.2. Формат строки 13
4.3. Набор символов 14
4.4. Константы 15
4.5. Переменные 15
4.6. Выражения и операции 16

4.7 Ввод и редактирование текста программ..... 18
5 Описание ключевых слов языка БЕЙСИК..... 20
6. Приемы программирования на языке БЕЙСИК..... 93
6.1. Разработка программы..... 93
6.2. Использование массивов и циклические вычисления..... 94
6.3 Вычисляемые переходы..... 95
6.4. Обработка строк..... 95
6.5. Организация подпрограмм..... 96
6.6 Применение графических операторов..... 96
6.7. Программирование музыки..... 97
6.8. Программирование ввода-вывода..... 98
7 Сообщения об ошибках..... 99
Приложение 1. Список зарезервированных слов и кодов их внутреннего представления..... 101
Приложение 2. Таблицы соответствия клавиш служебным словам 102
Приложение 3. Распределение памяти..... 104
Приложение 4. Таблица кодов КОИ-7..... 105

1. ВВЕДЕНИЕ

Прогресс в экономике, промышленности, науке и технике, в сфере образования сейчас во многом зависит от массового внедрения вычислительной техники. Задачу повышения "компьютерной грамотности" можно решить только развитием персональных ЭВМ (ПЭВМ).

БПЭВМ "ВЕКТОР-06Ц" является бытовой персональной ЭВМ, ориентированной на использование ее одним пользователем. Применение БПЭВМ "ВЕКТОР-06Ц" позволяет привлечь к работе на ЭВМ обширный круг пользователей, не являющихся специалистами в области вычислительной техники и программирования. Это достигается простотой работы с БПЭВМ "ВЕКТОР-06Ц", имеющей характер диалога с пользователем.

Язык БЕЙСИК был задуман разработчиками как язык программирования, доступный для освоения студентами гуманитарных вузов. Однако простота грамматики и синтаксиса, его сходство с языком программирования ФОРТРАН привели к широкому распространению языка БЕЙСИК.

Работа на БЕЙСИКе организуется с помощью специальной программы - интерпретатора, которая загружается в ОЗУ с магнитной ленты. Эта программа переводит каждый оператор БЕЙСИК-программы на машинный язык БПЭВМ.

Для БПЭВМ "ВЕКТОР-06Ц" реализована весьма мощная версия интерпретатора языка БЕЙСИК, обладающая широким набором средств программирования.

Примечание

В тексте настоящего руководства вместо знака денежной единицы "Д" используется знак "$"

2. ЗАГРУЗКА ИНТЕРПРЕТАТОРА ЯЗЫКА БЕЙСИК

Интерпретатор языка БЕЙСИК поставляется на магнитофонной кассете в составе комплекса программных средств для БПЭВМ "ВЕКТОР-06Ц"

Загрузка интерпретатора языка БЕЙСИК осуществляется в следующем порядке:

- установить в магнитофон кассету с интерпретатором БЕЙСИК;
- нажать одновременно на клавиши ВВОД и БЛ на системном блоке. На экране телевизора появятся восемь пар горизонтальных линий, представляющих собой незаполненную "карту загрузки" ОЗУ БПЭВМ, а в правом верхнем углу - изображение рабочих ячеек программы "начальный загрузчик";
- включить магнитофон в режим воспроизведения. На экране телевизора должно происходить заполнение прямоугольниками промежутков между парами горизонтальных линий, начиная с нижних. При завершении загрузки начинает мигать светодиод РУС/ЛАТ ;
- после загрузки нажать одновременно клавиши СБР и БЛ Через несколько секунд на экране появится изображение, приведенное на рис.1 . Если изображения нет, то загрузку следует повторить.



рис. 1.

3. КЛАССИФИКАЦИЯ ДИРЕКТИВ, ОПЕРАТОРОВ И ФУНКЦИЙ

В последующих разделах настоящего руководства подробно описаны все типы директив, операторов и функций, имеющиеся в интерпретаторе БЕЙСИК для БПЭВМ "ВЕКТОР-06Ц"

Ниже приводятся директивы, операторы и функции, которые для удобства изучения объединены в группы.

3.1. Изменение текста программы в памяти и вывод текста

AUTO	включает автоматическую нумерацию строк при вводе текста программы
DELETE	удаляет группу программных строк из памяти
EDIT	вызывает программную строку для редактирования
LIST	выводит текст программы на экран телевизора
LLIST	выводит текст программы на печатающее устройство
NEW	подготавливает интерпретатор для ввода новой программы с клавиатуры. Текст программы, набранный ранее, стирается
RENUM	пронумеровывает строки программы в памяти

3.2. Директивы чтения и записи на магнитную ленту

BSAVE	записывает часть оперативной памяти на магнитную ленту в двоичном формате
BLOAD	загружает информацию с магнитной ленты в ОЗУ в двоичном формате
CLOAD	загружает программу с магнитной ленты в оперативную память
CSAVE	записывает программу из оперативной памяти на магнитную ленту
MERGE	загружает с магнитной ленты программу и объединяет ее с программой в оперативной памяти
VERIFY	проверяет правильность записи программы на магнитную ленту

3.3. Управление выполнением программ

CONT	продолжает выполнение программы после оператора STOP
RUN	начинает выполнение программы

3.4. Начальная загрузка изменяемых параметров

CLEAR	инициализирует переменные; определяет размер области ОЗУ для символьных данных
DEF FN	описывает функции, определяемые пользователем
DIM	определяет, резервирует и инициализирует массивы
HIMEM	установка верхней границы ОЗУ под программу на языке БЕЙСИК
SCREEN	устанавливает режимы

3.5. Числовые функции

ABS()	определяет абсолютное значение аргумента
ACS()	арккосинус, результат в радианах
ADDR()	определяет адрес массивов или переменных в ОЗУ
ASN()	арксинус, результат в радианах
ATN()	арктангенс, результат в радианах
COS()	косинус угла в радианах
EXP()	число e возводится в указанную аргументом степень
FRE()	определяет размер свободного пространства в ОЗУ
INT()	выделяет целую часть числа
LG()	десятичный логарифм
LOG()	натуральный логарифм
PI	константа π, равная 3.14159
RND()	генератор псевдослучайных чисел
SGN()	определяет знак числа
SIN()	синус угла в радианах
SQR()	квадратный корень
TAN()	тангенс угла в радианах

3.6. Строковые функции

ASC()	преобразует байт данных из строкового формата в числовой
CHR$()	возвращает символ, код которого равен аргументу функции
LEFT$()	выделение подстроки из строковой переменной, начиная с крайнего левого символа
LEN()	определяет длину строковой переменной
MID$()	выделение подстроки из любого места строковой переменной
RIGHT$()	выделение подстроки из строковой переменной, начиная с крайнего правого символа
SPC()	позволяет вставить в распечатываемую строку пробелы
STR$()	преобразует число в символьную строку, содержащую это число
TAB()	функция табуляции, начинающая печать с позиции, заданной аргументом
VAL()	преобразует символьную переменную, содержащую цифры, в число

3.7. Изменение порядка работы программ

FOR	начинает многократное выполнение некоторой последовательности операторов
GOSUB	вызывает подпрограмму
GOTO	переход к строке с заданным номером
IF THEN	осуществляет переход в зависимости от условия
NEXT	заканчивает цикл, начатый оператором FOR
ON GOSUB	осуществляет переход на одну из строк, номера которых указаны в ON GOSUB, в зависимости от значения переменной (вычисляемый переход) с последующим возвратом по оператору RETURN
ON GOTO	то же, что и ON GOSUB, но без возврата
PAUSE	временная пауза в заданное число секунд
RETURN	заканчивает подпрограмму, возврат к оператору, стоящему после GOSUB

STOP	временный останов выполнения программы, с. сообщением. Выполнение можно продолжить, выполнив директиву CONT

3.8. Работа с данными, хранящимися в тексте программы

DATA	задает список значений данных, хранящихся в тексте программы
READ	присваивает значения из оператора DATA переменным
RESTORE	выбирает любой нумерованный оператор DATA в качестве "следующего" для чтения

3.9. Комментарии в программе

REM	служит для вставки в текст программы комментариев

3.10. Доступ к памяти и ввод-вывод

INP( )	вводит данные из порта ввода
OUT	посылает данные в порт вывода
PEEK( )	результатом работы является десятичное число, равное содержимому байта памяти
POKE	записывает данные по любому адресу ОЗУ
USR( )	обращается к заранее подготовленной подпрограмме на машинном языке

3.11. Управление экраном телевизора

CLS	очистка экрана; курсор устанавливается в левой верхней позиции
COLOR	устанавливает цвет изображения, фона и рамки
CUR	перемещает курсор на экране
HOME	очистка экрана; курсор устанавливается в левой верхней позиции
POS( )	возвращает позицию курсора в строке
PRINT	вывод на экран
SCREEN	устанавливает режим экрана

3.12. Графические операторы

SCREEN устанавливает режим экрана

GET запоминает в массиве графическую информацию с экрана

LINE рисует линии, прямоугольники и закрашенные прямоугольники, подготавливает вывод текста

PAINT заполняет замкнутые области любым цветом

PLOT рисует, стирает точку или перемещает графический курсор

POINT() возвращает код цвета любой точки экрана

PUT выдает из массива графическое изображение, запомненное оператором GET

CIRCLE рисует окружности, дуги и овалы

3.13. Управление звуком

BEEP выдает звуковой сигнал

PLAY исполняет музыку и создает звуковые эффекты

3.14. Ввод с клавиатуры

INKEY$ осуществляет ввод одного символа из буфера клавиатуры

INPUT осуществляет ввод данных с клавиатуры

3.15. Вывод на печать

LPRINT выводит данные на печатающее устройство

SCREEN печатает копию экрана в различных режимах печатающего устройства

3.16. Операции

3.16.1. Логические операции

AND логическое умножение "И"

OR логическое сложение "ИЛИ"

NOT отрицание "НЕ"

3.16.2. Числовые операции

+	сложение
-	вычитание
*	умножение
/	деление
^	возведение в степень
=	присваивает значения переменным

3.16.3. Строковые операции

+	сцепление строковых данных
=	присваивает значения переменным

3.16.4 Операции сравнения

=	равно
<>	не равно
>	больше
<	меньше
>=	больше или равно
<=	меньше или равно

4. ОБЩИЕ СВЕДЕНИЯ О ЯЗЫКЕ БЕЙСИК

4.1. Режимы работы

После запуска интерпретатор переходит в режим непосредственного счета. В этом режиме интерпретатор немедленно исполняет команды пользователя. Режим непосредственного счета используется для управления работой БПЭВМ, в том числе для подготовки программ и для выполнения несложных разовых вычислений.

Взаимодействие с машиной в режиме непосредственного счета осуществляется в форме диалога. Наличие на экране символов

```
=>
```

означает, что система готова к приему очередных директив пользователя. Чтобы выполнить один или несколько операторов БЕЙСИКа пользователь набирает эти операторы на клавиатуре, разделяя их двоеточием, и нажимает клавишу ВК . Введенная последовательность операторов, называемая строкой непосредственного счета, обрабатывается интерпретатором БЕЙСИК, и, если нет ошибок, строка непосредственного счета выполняется. После этого интерпретатор опять переходит в режим непосредственного счета. Например,

```
=>
PRINT 2+2 [ВК]
 4
```

В отличие от программы строки непосредственного счета не запоминаются и, следовательно, не могут быть выполнены повторно.

После запуска программы и до прекращения ее выполнения машина находится в программном режиме, в котором действия машины определяются последовательностью операторов выполняемой программы.

4.2. Формат строки

Строки программ на языке БЕЙСИК имеют следующий формат:

```
NNNNN  оператор[: оператор]...  ВК
```

В строке может быть несколько операторов. При этом каждый должен быть отделен от предыдущего двоеточием. Операторы IF и REM должны занимать всю оставшуюся часть строки.

Строка программы на языке БЕЙСИК всегда начинается с номера строки и заканчивается нажатием клавиши ВК. Строки программы запоминаются в памяти в соответствии со своими номерами.

Номер строки используется как указатель при передачах управления и при редактировании. Номер строки может иметь значение в пределах 0 - 65529.

Оператор всегда состоит из ключевого слова и необязательных операндов.

Пробелы могут встречаться в любом месте строки. Пробелы не допускаются:
- в начале строки;
- внутри служебного слова;
- внутри числовой константы;
- внутри номера строки;
- внутри символического имени;
- внутри символов отношения,состоящих из нескольких символов.

4.3. Набор символов

Набор символов, используемых в БПЭВМ "ВЕКТОР-06Ц" состоит из:
- прописных букв латинского алфавита;
- прописных букв русского алфавита;
- цифр от 0 до 9;
- специальных символов:

	пробел
=	знак равенства или присваивания
+	знак плюс
-	знак минус
*	звездочка или знак умножения
/	знак деления
^	знак возведения в степень
(	левая скобка
)	правая скобка
%	процент
#	номер
$	знак денежной единицы
!	восклицательный знак
[	левая квадратная скобка
]	правая квадратная скобка
,	запятая
.	десятичная точка
"	кавычки
'	апостроф

;	точка с запятой
:	двоеточие
&	амперсанд
?	знак вопроса
<	меньше чем
>	больше чем
\	обратная косая черта
@	знак цены

4.4. Константы

В языке БЕЙСИК существует два типа констант: числовые и строковые. Числовые - это любые десятичные числа в интервале от $-1.7*10^{+38}$ до $1.7*10^{-38}$, строковые - это последовательность любых отображаемых символов, заключенных в кавычки. Например, "КИШИНЕВ", "РАВНО", "АВТОР" - строковые константы, 314, -225, 3.14, -3.1E-03, 12E5 - числовые. Две последние константы заданы в экспоненциальной форме, на что указывает буква E. После буквы E следует знак (у положительных чисел знак "+" можно опускать) и величина порядка. Точность задания числовых констант - 6 значащих цифр.

4.5. Переменные

4.5.1. Простые переменные

Переменные - это имена, обозначающие области в оперативной памяти, которые БЕЙСИК выделяет для автоматического запоминания величин, используемых в программах. Значение переменной может быть явно задано программистом или получено в результате вычислений в программе. При запуске программы все числовые переменные получают нулевое значение, а строковые переменные - значение "" (пустая строка).

Имена переменных могут иметь любую длину, но различаются они лишь по первым двум символам (должны быть использованы латинские буквы и цифры, но первым символом должна быть буква).

Имена переменных не должны совпадать с ключевыми словами языка БЕЙСИК. Если переменная начинается с FN, то предполагается, что это обращение к функции, определенной пользователем.

Имя переменной определяет ее тип: знак $ в конце имени указывает, что переменная строковая, отсутствие знака означает, что переменная числовая.

Пример A1$="ТЕКСТ"

4.5.2. Массивы переменных

Массив представляет собой группу переменных (элементов массива), имеющих одно и то же имя. Каждый элемент массива обозначается именем массива, дополненным целым порядковым номером (индексом).

Каждый массив переменных имеет количество индексов, равное размерности массива. Например, A(10) обозначает одномерный массив B(2,4) обозначает двумерный массив, AB(7,2,3) - трехмерный и т.д.

Массивы могут быть числовыми и строковыми. Массивы удобно использовать для работы с таблицами данных.

Индекс массива может быть целым арифметическим выражением.

Для массивов в программе необходимо резервировать память с помощью оператора DIM, иначе массиву присваиваются умалчиваемые атрибуты (см. раздел 5, оператор DIM).

4.6. Выражения и операции

Выражение - это строковая или числовая константа, переменная или группа констант и переменных, соединенных между собой знаками операций так, что в результате выполнения этих действий получается единственное значение.

Операции выполняют арифметические и логические действия с элементами выражения.

В языке БЕЙСИК для БПЭВМ "Вектор - 06Ц" предусмотрены пять типов операций:

- арифметические операции;
- операции отношения;
- логические операции;
- операции - функции;
- строковые операции.

4.6.1. Арифметические операции

Знаки арифметических операций, использующихся в языке БЕЙСИК:

+	сложение
-	вычитание или унарная операция для присвоения отрицательного числа
*	умножение
/	деление
^	возведение в степень

Арифметические операции выполняются в следующем порядке: возведение в степень, унарная операция "-", умножение и деление, сложение и вычитание.

Для изменения порядка выполнения действий используются круглые скобки.

При вычислении арифметического выражения сначала выполняются действия в скобках. Внутри скобок действия выполняются в обычном порядке.

4.6.2. Операции отношения

Операции отношения применяются для сравнения двух величин. Результатом сравнения будет логическая величина "истина" (1) или "ложь" (0). Обычно результат операции отношения используют для принятия решения о дальнейшем ходе выполнения программы (оператор условного перехода IF).

Операции отношения :

=	равно
<>	не равно
<	меньше чем
>	больше чем
<=	меньше или равно
>=	больше или равно

При использовании в выражении арифметических операций и операций отношения арифметические выполняются первыми.

Например, выражение

```
X-Y < Z^2
```

истинно, если величина X-Y меньше, чем Z^2 .

4.6.3. Логические операции

В настоящей реализации интерпретатора языка БЕЙСИК предусмотрены логические операции:

AND	логическое умножение (" и ")
OR	логическое сложение ("или")
NOT	отрицание ("не")

Кроме того, могут применяться составные логические операции, производные от вышеупомянутых:

NOT AND	не "и"
NOT OR	не "или"

По приоритету логические операции следуют за арифметическими операциями и операциями отношения. Логические операции могут быть использованы для принятия решения о дальнейшем ходе выполнения программы (оператор IF).

Примеры:

```
1.  IF A < 100 AND B >=300 THEN 200
2.  IF X = 20 OR Y < 0 THEN 50
```

В первом примере если A меньше 100 и B больше или равно 300, то следующим будет выполняться оператор с номером строки 200.

Во втором примере, если X равно 20 или Y меньше 0, то следующим будет выполняться оператор с номером строки 50.

4.6.4. Операции-функции

Встроенные и определенные программистом функции используются в выражениях для выполнения определенных действий.

Язык БЕЙСИК содержит большое количество встроенных функций, выполняющих самые разнообразные действия. В разделе 5 описаны все встроенные функции, использующиеся в настоящей реализации языка БЕЙСИК.

4.6.5 Строковые операции

Строковые переменные и константы могут сцепляться в одну строковую переменную или константу с помощью знака "+".

Например:

```
10 A$="ИДЕТ"
20 S$=A$+"ДОЖДЬ"
```

Переменная S$ получает значение "ИДЕТ ДОЖДЬ".

Строковые переменные могут сравниваться с помощью тех же операций отношения, что и числовые: =, <>, <, >, <=, >=.

Строки сравниваются посимвольно слева-направо. Значением символа является его внутримашинный код. Сравнение происходит до первой пары несовпадающих символов. Строковая переменная, содержащая символ с меньшим кодом, считается меньшей. Если все коды совпадают, то строки равны.

Если строки имеют разную длину, то более краткая строка дополняется справа пробелами.

4.7 Ввод и редактирование текста программ.

Текст программы вводится построчно в режиме диалога. Ввод программных строк может быть осуществлен только после сообщения => на экране в начале строки, означающего, что Бейсик находится в ожидании приема директив оператора или ввода программных строк.

Программные строки можно вводить сразу после запуска интерпретатора или же в любой момент, как только появится сообщение => на экране, т.е. тогда когда интерпретатор находится в режиме непосредственного счета.

Перед вводом программы необходимо очистить память БПЭВМ директивой NEW.

В этом случае из оперативной памяти удаляются программа и данные. Если же необходимо очистить экран, то следует ввести директиву CLS или HOME.

При вводе программной строки пользователь должен сначала набрать ее номер, а затем один или несколько операторов, разделяя их двоеточием. Операторы интерпретатора Бейсик могут набираться как по символьно, так и по целым словам путем последовательного нажатия клавиши AP2 и клавиши, соответствующей вводимому служебному слову. Предусмотрена

дополнительная возможность ввода служебных слов посредством одновременного нажатия клавиш УС,СС и клавиши основного поля клавиатуры. Таблица соответствия клавиш служебным словам приведены в приложении 2.

Вводимую строку можно сразу редактровать,вставляя пропущеный символ или слово перед символом,под которым расположен курсор (при этом строка автоматически раздвигается),или удаляя символ клавиша ЗБ удаляет символ слева от курсора,F2 над курсором.Заканчивается ввод строки нажатием клавиши ВК,причем курсор может располагаться в любом месте строки.Максимальная длина строки 128 символов.

Редактирование уже введенных программных строк осуществляется в непосредственном режиме директивой EDIT.

По директиве EDIT с указаного номера строки на экране высвечивается строка ,которую можно редактировать,используя следующие клавиши:

<--	перемещение курсора на одну позицию вправо;
-->	перемещение курсора на одну позицию влево;
	ускоренное перемещение курсора влево;
СТР	ускоренное перемещение курсора вправо;
	перемещение курсора в начало строки;
	перемещение курсора в конец строки;
ЗБ	удаление символа слева от курсора;
F2	удаление символа над курсором.

Вставка символов также ,как и при вводе ,осуществляется автоматически перед символом,под которым расположен курсор.Заканчивается редактирование нажатием клавиши ВК.

Удаление строк из текста программы ,находящейся в памяти БПЭВМ может быть проведено директивой DELETE,в которой указываются номера удаляемых строк,или набором номера строки с последующим нажатием ВК ,если необходимо удалить одну строку из программы.

Надо иметь ввиду,что при вводе программы все программные строки должны иметь различные номера.Если номер вводимой программной строки совпадает с номером какой-либо строки,уже имеющейся в памяти ,то эта строка замещает в памяти строку,введеную ранее.

5. ОПИСАНИЕ КЛЮЧЕВЫХ СЛОВ ЯЗЫКА БЕЙСИК

Функция **ABS** Функция

Назначение: Функция ABS предназначена для возвращения абсолютной величины аргумента.

Формат: ABS(X)
X - число или арифметическое выражение.

Действие: Функция ABS возвращает положительное значение аргумента независимо от его знака.

Применение: Функция ABS может применяться при вычислении и сравнении чисел.

Пример:

X=ABS(A-B)
определяет разность чисел A и B, независимо от того, какое из них больше

Функция **ACS** Функция

Назначение: Функция ACS предназначена для вычисления арккосинуса; результат возвращается в радианах.

Формат: ACS(X)
X - число или арифметическое выражение.

Действие: Функция ACS вычисляет угол, косинус которого равен заданному аргументу X. X должно находиться в пределах от -1 до +1. Результат возвращается в радианах от 0 до π.

Применение: Функция ACS может быть использована в вычислениях.

Пример:

Y=ACS(X/3)
Y принимает значение арккосинуса выражения X/3

Функция **ADDR** **Функция**

Назначение: Функция ADDR предназначена для определения адреса переменной или массива в оперативной памяти.

Формат: ADDR(X)
X - имя переменной

Действие: Функция ADDR определяет адрес переменной (простой или индексированной). Возвращаемое значение - число, определяющее адрес первого байта памяти, начиная с которого указанная переменная находится в ОЗУ.

Применение: Функция ADDR может быть применена для определения адресов переменной или массива. Смотри GET, PUT, BSAVE, BLOAD.

Пример:

```
C=ADDR(A$)
```
C получает значение адреса переменной A$

Функция **ASC** **Функция**

Назначение: Функция ASC предназначена для преобразования одного байта из строкового формата в числовой.

Формат: ASC(<строка>)
<строка> - любое строковое выражение.

Действие: Функция ASC выбирает и анализирует первый байт строкового выражения. Функция возвращает число, соответствующее коду символа в рассматриваемом байте.

Применение: Функция ASC и обратная ей CHR$ могут применяться для хранения и извлечения компактно располагаемых небольших по значению величин (от 0 до 127). С использованием функций ASC и CHR$ такие величины можно хранить в одном байте.

Пример:

```
X=ASC("S")
```
X получает значение 83

Функция **ASN** Функция

Назначение: Функция ASN предназначена для вычисления арксинуса; результат возвращается в радианах.

Формат: ASN(X)
X - любое число или арифметическое выражение.

Действие: Функция ASN вычисляет угол, синус которого равен заданному аргументу X. X должно находиться в интервале от -1 до +1. Результат возвращается в радианах от $-\pi/2$ до $+\pi/2$.

Применение: Функция ASN может быть использована в вычислениях.

Пример:

```
Z=ASN(Y*2)
```
Z получает значение арксинуса выражения 2*Y

Функция **ATN** Функция

Назначение: Функция ATN предназначена для вычисления арктангенса, результат возвращается в радианах.

Формат: ATN(X)

Действие: Функция ATN вычисляет угол, тангенс которого равен заданному аргументу X. Результат возвращается в радианах в интервале от $-\pi/2$ до $\pi/2$.

Применение: Функция ATN может быть использована в вычислениях.

Пример:

```
Z=ATN(X)
```
Z получает значение арктангенса X

директива **AUTO** директива

Назначение: Директива AUTO предназначена для автоматической генерации на экране номера следующей программной строки.

Формат: AUTO [<номер строки>] [,<шаг>]
<номер строки> - числовая константа, по умолчанию значение номера строки равно 10
<шаг> - числовая константа, по умолчанию значение шага равно 10.

Действие: Директива AUTO устанавливает режим автоматической генерации номеров программных строк. После каждого нажатия клавиши ВК следующая строка автоматически получает больший номер. Курсор устанавливается после номера строки и пробела и можно после этого вводить текст строки.
Аргумент <номер строки> указывает номер начальной строки, второй аргумент - шаг между номерами. По умолчанию оба аргумента имеют значение 10.
Для выхода из режима автоматической генерации номеров программных строк необходимо нажать клавишу F4

Применение: Директиву AUTO применяют для экономии времени при вводе программы с клавиатуры, так как ручную нумерацию можно опустить.

Примеры:

```
AUTO
```
нумерация начинается со строки 10 и с шагом 10, т.е. строки нумеруются: 10, 20, 30 и т.д.

```
AUTO 100
```
нумерация начинается со строки 100 и с шагом 10, т.е. строки нумеруются: 100, 110, 120 и т.д.

```
AUTO , 20
```
нумерация начинается с номера 10 и с шагом 20, т.е. строки нумеруются: 10, 30, 50 и т.д.

```
AUTO 50, 20
```
нумерация начинается с номера 50 и с шагом 20, т.е. строки нумеруются: 50, 70, 90 и т.д.

оператор **BEEP** оператор

Назначение: Оператор BEEP предназначен для подачи звукового сигнала.

Формат: BEEP <продолжит. звука> ,<высота тона>
<продолжит.звука> - продолжительность звукового сигнала в секундах от 0.001 до 65
<высота тона> - высота тона звука от -12 до 12

Действие: Оператор BEEP подает звуковой сигнал продолжительностью, определенной параметром <продолжит.звука> в секундах, и высотой тона, определенной параметром <высота тона>. Если высота тона равна 0, то это соответствует ноте ДО первой октавы, если равно -12, то ноте ДО малой октавы, если равно +12, то ноте ДО второй октавы и т.д. Выполнение программы прекращается на время звучания.

Применение: Оператор BEEP может использоваться для сигнализации о наличии ошибок, обнаруживаемых программой пользователя.

Пример:

```
BEEP 5,8
```

директива **BLOAD** директива

Назначение: Директива BLOAD предназначена для считывания с магнитной ленты области оперативной памяти, записанной директивой BSAVE

Формат: BLOAD <имя файла> [, <смещение>]
<имя файла> - любое строковое выражение, задающее имя файла на кассете
<смещение> - целое арифметическое выражение.

Действие: Директива BLOAD читает с кассеты двоичный файл, имя которого задано в директиве. Если имя файла не задано, то используется первый найденный файл в двоичном коде.
Файл содержит начальный и конечный адреса, созданные директивой BSAVE, для размещения файла в оперативной памяти. Если задано смещение, то оно добавляется к этим значениям для помещения файла в любое место оперативной памяти, кроме области кодов самого интерпретатора языка БЕЙСИК.

Применение: Директива BLOAD может быть применена для загрузки данных и программ в оперативную память. Для обращения к программам, загруженным директивой BLOAD, можно воспользоваться функцией USR.

Примеры:

BLOAD "" - загружается первый найденный файл

BLOAD "PROG" - загружается файл с именем PROG

BLOAD "PROG",100 - загружается файл с именем PROG в оперативную память со смещением 100 байт.

директива **BSAVE** директива

Назначение: Директива BSAVE предназначена для записи из оперативной памяти на магнитную ленту машинных кодов или данных в двоичном виде.

Формат: BSAVE <имя файла>, <начальный адрес>, <конечный адрес>
<имя файла> - любое строковое выражение, задающее имя файла
<начальный адрес> - целое арифметическое выражение
<конечный адрес> - целое арифметическое выражение

Действие: Директива BSAVE записывает содержимое оперативной памяти с <начального адреса> по <конечный адрес> на кассету в файл с именем <имя файла>;
<начальный адрес> и <конечный адрес> записываются в начале файла в первые 4 байта.

Применение: Директива BSAVE используется для сохранения областей оперативной памяти путем записи их в двоичном виде в файл на кассете. Адреса могут быть указаны в шестнадцатиричных кодах.

Пример:

```
BSAVE "PROG", 4000,6300
```

будет записана область оперативной памяти с адреса 4000 до адреса 6300

Назначение: Функция CHR$ предназначена для преобразования целого числа, соответствующего коду символа, в символьную переменную.

Формат: CHR$(X)
X - любое число или выражение в интервале от 0 до 127.

Действие: Строка, созданная функцией CHR$, состоит из одного символа, внутримашинный код которого соответствует значению аргумента X.

Применение: Функция CHR$ и обратная ей ASC могут применяться для хранения и извлечения компактно располагаемых небольших по значению величин (от 0 до 127). С использованием функций ASC и CHR$ такие величины можно хранить в одном байте.

Пример:

```
X$=CHR$(83)
```
X$ получит значение "S"

оператор **CIRCLE** оператор

Назначение: Оператор CIRCLE предназначен для рисования окружностей и дуг любого размера.

Формат:
```
CIRCLE [STEP] X,Y ,<радиус>[,<нач.угол>
  [ ,<кон. угол> ] ] [,<овал>]
```
X - координата по горизонтали от -32767 до 32768
Y - координата по вертикали от -32767 до 32768
STEP - указывает, что координаты точки X,Y вычисляются относительно координат графического курсора
<радиус> - радиус в диапазоне 0 - 255
<нач.угол> - начало дуги в радианной мере ; по умолчанию равно 0
<кон.угол> - конец дуги в радианной мере; по умолчанию равно 2π
<овал> - отношение осей; от 1/255 до 255; значение по умолчанию равно 1

Действие: Оператор CIRCLE рисует окружность или дугу окружности. Центр круга (X,Y) может быть расположен в любой точке координатной плоскости, даже за пределами экрана. При указании параметра STEP координаты центра вычисляются относительно текущего положения графического курсора. Начало окружности лежит слева от центра. Рисование выполняется в направлении против часовой стрелки. На экране изображаются только те точки, координаты которых лежат в пределах (0 - 255, 0 - 255), отсчитывая от нижнего левого угла экрана. При указании параметров <нач.угол> и <кон.угол> рисуется дуга окружности. Параметр <кон.угол> можно не задавать. Тогда берется значение <кон.угол> по умолчанию, равное 2π.
При указании параметра <овал> рисуется овал. При этом отношение горизонтальной и вертикальной оси равно значению параметра <овал>. Так при значении этого параметра, равном 2, рисуется овал с отношением горизонтальной и вертикальной осей 2:1. Ориентация овала всегда либо горизонтальная, либо вертикальная.

Применение: Оператор CIRCLE позволяет рисовать кривые без использования циклов. С его помощью окружности рисуются гораздо быстрее, чем другими способами (например, с помощью тригонометрических функций). Если круги на экране оказываются "недостаточно круглыми", необходимо применить параметр <овал>, выбирая его в диапазоне от 1 до 1.2.

оператор **CIRCLE** оператор

Пример:

```
CIRCLE 100,100,75
```

рисуется окружность радиусом 75, центр которой расположен в точке (100,100).

директива **CLEAR** директива

Назначение: Директива CLEAR предназначена для обнуления переменных и отведения места для строковых данных

Формат: CLEAR [<размер строки>]
<размер строки> - число байт, отводимых для строковой переменной.

Действие: Директива CLEAR обнуляет все переменные, это означает, что обнуляется все, кроме текста программы. Сюда входят массивы и переменные, определенные функции, строки символов и все содержимое аппаратного стека. Причем числовым данным присваивается значение 0 , а строковым - "" (пустая строка). Кроме того, директива CLEAR используется для определения объема памяти, отводимого для хранения строковых данных.

Применение: Директива CLEAR применяется для выделения памяти строковым переменным и очистки памяти от переменных и массивов.

Примеры:

```
 CLEAR
```
удаляются все переменные и для строковых переменных отводится 200 байт
```
 CLEAR 2000
```
удаляются все переменные и для строковых переменных отводится 2000 байт

директива **CLOAD** директива

Назначение: Директива CLOAD загружает с магнитной ленты программу, написанную на языке БЕЙСИК

Формат: CLOAD <имя файла>
<имя файла> - строковое выражение.

Действие: Директива CLOAD загружает программу с кассеты в оперативную память. <имя файла> - строковое выражение, состоящее не более чем из 6 символов, определяет имя программы на кассете. Поиск программы на кассете осуществляется по имени до конца ленты, пока программа с заданным именем не будет найдена. При отсутствии программы с заданным именем на кассете выдается сообщение об ошибке. В данной директиве можно задавать пустое <имя файла>. В этом случае загрузится первая встретившаяся на кассете программа, записанная в соответствующем формате.

Применение: Директива CLOAD является основным средством загрузки программ, написанных на языке БЕЙСИК, в оперативную память с магнитной ленты.

Примеры:

CLOAD "PROG"
загружается с магнитной ленты в ОЗУ программа с именем "PROG"

CLOAD ""
загружается с магнитной ленты в ОЗУ первая встретившаяся программа, записанная директивой CSAVE

оператор **CLS** оператор

Назначение: Оператор CLS предназначен для очистки экрана и установки курсора в левой верхней позиции.

Формат: CLS

Действие: Оператор CLS очищает экран и устанавливает курсор в левую верхнюю позицию.

Пример:

CLS
очищается экран и курсор устанавливается в левую верхнюю позицию.

оператор **COLOR** оператор

Назначение: Оператор COLOR предназначен для установки цветов изображения, фона и рамки.

Формат: COLOR<цвет изображения> [,<цвет фона>] [,<цвет рамки>]

<цвет изображения> - арифметическое выражение, определяющее цвет изображения диапазон от 0 до 255

<цвет фона> - арифметическое выражение, определяющее цвет фона, диапазон от 0 до 255

<цвет рамки> - арифметическое выражение, определяющее цвет рамки, диапазон от 0 до 15.

Действие: Оператор COLOR устанавливает цвета изображения, фона и рамки.
Если параметры <цвет фона> и <цвет рамки> не заданы, то сохраняются старые значения цвета фона и цвета рамки.
<цвет изображения> - это арифметическое выражение, определяющее математический цвет изображения и подходов под символами.
<цвет фона> - это арифметическое выражение, определяющее физический цвет фона.
<цвет рамки> - это арифметическое выражение, определяющее математический цвет рамки экрана.
Цвета изображения, фона и рамки определяются не только арифметическими выражениями, но и текущим режимом экрана.
Зависимость цвета от заданного выражения и установленного режима экрана описана в операторе SCREEN.

Применение: Оператор COLOR применяется для изменения цвета.

Примеры:

```
COLOR 15
COLOR 10, 6, 15
```

Директива **CONT** директива

Назначение: Директива CONT предназначена для продолжения выполнения программы после оператора STOP или приостановки выполнения программы одновременным нажатием клавиш УС и "E"

Формат: CONT

Действие: Директива CONT возобновляет выполнение программы с той точки, где она была прервана оператором STOP. При продолжении выполнения программы с помощью CONT выполняется оператор, следующий за оператором STOP.

Применение: Директива CONT совместно с оператором STOP используется для отладки программ. Если вставить операторы STOP в нескольких местах программы, то при остановах программы можно вывести значения переменных в непосредственном режиме, а затем набрать CONT. Во время прерывания можно также изменить значения различных переменных. Кроме того, связку STOP/CONT можно использовать для прерывания программы при проведении пользователем некоторых действий, требующихся для выполнения программы. Например, можно прервать программу и сообщить о необходимости установить кассету в магнитофон. После установки кассеты можно продолжить программу с помощью директивы CONT.

Функция **COS** функция

Назначение: Функция COS предназначена для вычисления косинуса угла

Формат: COS(X)
X -- арифметическое выражение, определяющее угол в радианах.

Действие: Функция COS вычисляет значение косинуса угла, заданное в радианах.

Применение: Функция COS может быть использована в вычислениях

Пример:

```
Z=COS(X^2+Y^2)
```

Z получает значение косинуса выражения X^2+Y^2

директива **CSAVE** директива

Назначение: Директива CSAVE предназначена для записи текста программы, написанной на языке БЕЙСИК, на магнитную ленту во внутреннем формате.

Формат. CSAVE <имя файла>
<имя файла> - строковое выражение.

Действие. Директива CSAVE записывает текст программы на магнитную ленту, используя внутренний формат. В качестве имени файла можно использовать любую строку символов. При отсутствии или неверном задании имени файла выдается сообщение об ошибке.

Применение: Директива CSAVE применяется для записи программ на магнитную ленту. Программы, записанные директивой CSAVE, могут быть в дальнейшем загружены директивой CLOAD.

Пример:

```
CSAVE "PROG"
```

запись программы с именем "PROG" на магнитную ленту

оператор **CUR** оператор

Назначение: Оператор CUR предназначен для установки курсора на экране в необходимую позицию.

Формат: CUR X,Y
X - координата курсора по горизонтали от 0 до 41
Y - координата курсора по вертикали от 0 до 24
Отсчет координат ведётся из нижнего левого угла

Действие: Оператор CUR перемещает курсор в позицию: столбец X по горизонтали, строка Y по вертикали Начало отсчета левый нижний угол экрана.Если после оператора CUR сразу выполняется оператор печати PRINT,то вывод информации на экран начинается с позиции столбец X ,строка Y

Применение: Оператор CUR позволяет создавать программы, реализующие так называемый экранный режим работы. К ним можно отнести разнообразные игровые программы, экранные редакторы текстов, программы обработки информации, представленной в табличной форме и многие другие.

Пример:

CUR 30,15
курсор устанавливается на экране с координатами X=30 и Y=15

оператор **DATA** оператор

Назначение. Оператор DATA предназначен для хранения констант в тексте программы и их использования по мере необходимости.

Формат. DATA <список значений>
<список значений> -последовательность числовых и символьных констант, разделенных запятыми

Действие. Оператор DATA в тексте программы используется для хранения констант. Оператор DATA во время выполнения программы пропускается. Он используется только в операторе READ. Перед выполнением программы БЕЙСИК просматривает все операторы DATA в порядке их появления и создает блок данных. Каждый раз, когда программе встречается оператор READ, из блока данных выбирается последовательно соответствующее значение для переменных этого оператора в том порядке, в котором они заданы в блоке. При "исчерпании" всех операторов DATA данные не могут считываться, пока не будет выполнен оператор RESTORE. RESTORE позволяет выбрать любой нумерованный оператор DATA в качестве "следующего" для чтения.

Применение: Оператор DATA - это самый краткий способ инициализации констант в программе. Он также является практичным способом инициализации массивов.

Пример:

```
DATA "1988", "ВЕКТОР-06Ц","БЕЙСИК",5.2,100
```

оператор **DEF FN** оператор

Назначение: Оператор DEF FN предназначен для определения функции пользователя

Формат: DEF FN <имя функции> [(<параметр>)] = выражение

<имя функции> - любое допустимое имя, которое в дальнейшем рассматривается как имя функции

<параметр> - любое допустимое имя, которое будем называть формальной переменной

выражение - любое выражение, имеющее тот же тип, что и параметры.

Действие: Оператор DEF FN описывает функцию пользователя. Описание функции должно всегда происходить перед ее первым вызовом. Список параметров содержит имена формальных переменных, которые используются при формировании выражения функции. При вызове функции формальные параметры заменяются фактическими переменными. Выражение функции вычисляется при вызове функции пользователя, и результат, после преобразования к типу, определяемому переменной <имя функции>, возвращается в точку вызова функции. Если выражение функции содержит имя переменной, описанной в качестве параметра, ее значение вычисляется исходя из фактического параметра оператора вызова функции FN

Применение: Оператор DEF FN используется для создания в программе собственных функций.

Пример:

```
DEF FN X(I)=C*D/I          - описание функции X
Y=FN X(5)                  - вызов функции X
DEF FN S$(A$)=A$+B$ - определение функции
                      S$ строкового типа.
```

директива **DELETE** директива

Назначение: Директива DELETE предназначена для удаления строк из текста программы в оперативной памяти.

Формат: DELETE [<начальный номер строки>]
[,<конечный номер строки>]

<начальный номер строки> - арифметическое выражение
<конечный номер строки> - арифметическое выражение.

Действие: Директива DELETE позволяет удалять часть текста программы или всю программу полностью. Если опущен аргумент <начальный номер строки>, то удаляются строки от начала программы и до строки, указанной в аргументе <конечный номер строки>. Если опущен аргумент <конечный номер строки>, то удаляются строки от начального номера строки и до конца текста программы.
Директива DELETE без параметров удаляет всю программу.

Применение: Директива DELETE применяется для удаления программных строк из оперативной памяти при отладке программы и компоновки программ с помощью директивы MERGE.

Примеры:

DELETE 100,500 - удаляются строки текста программы от 100 до 500

DELETE 200 - удаляются строки от 200 до конца программы

DELETE ,800 - удаляются строки от начала программы до 800

DELETE - удаляются все программные строки

оператор **DIM** оператор

Назначение: Оператор DIM предназначен для описания массивов, используемых в программе.

Формат: DIM <имя1> [,<имя2>] [,<имя3>] ...
<имя1> - имена массивов, за которыми в круглых
<имя2> скобках следует список максимальных
<имя3> значений индексов.

Действие: Оператор DIM описывает массивы арифметического или строкового типа. После имени массива следует знак "$" если массив строкового типа. За именем массива или знаком "$", если массив строкового типа, следует список максимальных значений индексов. Каждый созданный массив имеет количество элементов, равное произведению максимальных индексов. Массив можно не описывать, если его номера индексов не превышают 10.

Применение: Оператор DIM используется для описания массивов требуемого типа и размера.

Пример:

```
DIM M(20,40),S$(100)
```

директива **EDIT** директива

Назначение: Директива EDIT предназначена для редактирования программных строк.

Формат: EDIT <номер строки>

Действие: Директива EDIT вызывает появление на экране строки с указанным номером. Используя клавиши управления курсором, можно скорректировать вызванную строку, а потом записать ее, нажав клавишу ВК . При корректировке можно изменить номер строки. В этом случае программная строка записывается в ОЗУ в соответствии с присвоенным ей номером. В этом случае вызванная строка директивой EDIT останется в оперативной памяти без изменения.

Применение: Директива EDIT применяется для редактирования программных строк в оперативной памяти. Кроме того, она может быть использована для получения новых строк из уже существующих в тексте программы и выполнения отдельных строк в непосредственном режиме (коррекция и удаление номера строки).

Пример:

EDIT 220
на экране появится строка с номером 220.

функция **EXP** функция

Назначение: Функция EXP предназначена для вычисления e^x

Формат: EXP(X)
X - арифметическое выражение.

Действие: Функция EXP вычисляет значение основания натуральных логарифмов в степени, заданной аргументом X.

Применение: Функция EXP в основном применяется в расчетных задачах. Константа e (основание натуральных логарифмов) может быть получена с помощью EXP(1).

Пример: Z=EXP(X)
Z получает значение e^x

оператор **FOR/NEXT** оператор

Назначение: Операторы FOR и NEXT предназначены для организации циклов в программе.

Формат: FOR <перем. цикла>=<начало> TO <конец>[STEP<шаг>]
NEXT <перем. цикла>

<перем. цикла> - арифметическая переменная, хранящая текущее значение счетчика
<начало> - арифметическое выражение, задающее начальное значение счетчика
<конец> - арифметическое выражение, задающее конечное значение счетчика
<шаг> - арифметическое выражение, задающее величину, прибавляемую к счетчику при каждом повторении цикла; по умолчанию <шаг> равен 1

Действие: Операторы FOR и NEXT используются в программах для организации циклов. При выполнении оператора FOR первый раз вычисляется значение выражения <начало>, которое присваивается переменной цикла. Затем идет обычное выполнение операторов. При достижении парного оператора NEXT переменная цикла увеличивается на значение, заданное выражением <шаг> (или уменьшается, если <шаг> отрицателен). Значение переменной цикла сравнивается со значением выражения <конец> - если переменная цикла больше, выполнение нормально продолжается после оператора NEXT; если нет, то управление передается обратно оператору FOR и снова выполняются все операторы между FOR и NEXT.
Заметим, что при отрицательном значении шага идет уменьшение переменной цикла, и в этом случае значение <начало> должно быть больше значения <конец>.
Операторы FOR могут быть вложенными в том смысле, что самый внутренний цикл выполняется первым - основным правилом является различие переменных цикла. Таким образом, оператор NEXT внутреннего цикла должен стоять перед соответствующим NEXT внешнего цикла. Надо также отметить, что оператор FOR и операторы тела цикла всегда выполняются хотя бы один раз, даже если условия окончания цикла истинны в самом начале.

оператор **FOR/NEXT** оператор

Применение: Операторы FOR и NEXT используются для организации циклов в программе.

Примеры:

```
1.
FOR I = 1 TO 10
PRINT "ЭТОТ ОПЕРАТОР БУДЕТ ВЫПОЛНЕН 10 РАЗ"
NEXT I

2.
FOR I = 1 TO 10
FOR J = 1 TO 10
PRINT  "ЭТОТ ОПЕРАТОР БУДЕТ ВЫПОЛНЯТЬСЯ 100 РАЗ"
NEXT J
PRINT "А ЭТОТ - 10 РАЗ"
NEXT I
```

Функция **FRE** **Функция**

Назначение: Функция FRE предназначена для определения объема свободной оперативной памяти.

Формат: FRE (< аргумент>)
<аргумент> - любая переменная или константа строкового или числового типа.

Действие: Функция FRE имеет фиктивный аргумент, т.е. аргументом функции может быть любое число или строка символов. Функция FRE выдает количество свободной оперативной памяти в байтах, неиспользуемой в настоящее время, если аргументом является число.
Функция FRE со строковым аргументом выдает количество неиспользуемой памяти в байтах в строковом пространстве, определенном директивой CLEAR.

Применение: Функция FRE используется для определения объема незанятой оперативной памяти или незаиятой части строкового пространства.

Примеры:

FRE(0) - возвращает число байт, равное объему неиспользуемой оперативной памяти.
FRE("A") - возвращает число байт, равное объему неиспользуемого строкового пространства, определенного директивой CLEAR.

оператор **GET** оператор

Назначение: Оператор GET предназначен для запоминания графической информации с экрана в ОЗУ.

Формат: GET X,Y, <адрес массива>
X и Y - любые целые арифметические выражения в диапазоне 0 - 255
<адрес массива> - адрес массива, в котором запоминается графическая информация с экрана.

Действие: Оператор GET запоминает в массиве, адрес которого задан параметром <адрес массива>, графическую информацию с экрана, ограниченную прямоугольником, левый нижний угол которого определяется текущим положением графического курсора, а размеры сторон - арифметическими выражениями X и Y.
Массив, в котором запоминается графическая информация с экрана, должен быть числовым и обязательно описан оператором DIM. Размеры массива должны быть достаточными для размещения графической информации.
Минимальное число (N) элементов массива можно вычислить по формуле:
N >= INT(X*Y/8+3/4)+1
Например, если X=10 и Y=40, то число элементов в массиве должно быть не менее 51.

Применение: Оператор GET в сочетании с оператором PUT позволяет перемещать по экрану графическую информацию.

Пример:

```
10 DIM A(51)
20 PLOT 70,30,2
30 GET 40,10,ADDR(A(0))
```

оператор **GOSUB** оператор

Назначение: Оператор GOSUB совместно с оператором RETURN предназначен для организации подпрограмм.

Формат: GOSUB <номер строки>
<номер строки> - любой существующий номер строки программы.

Действие: Оператор GOSUB передает управление на строку с номером <номер строки>, с которой должна начинаться подпрограмма. Заканчиваться подпрограмма должна обязательно оператором RETURN. После его выполнения происходит возврат к оператору, следующему за оператором GOSUB. Допустима многократная вложенность подпрограмм.

Применение: Оператор GOSUB используется для организации подпрограмм.

Пример:

```
GOSUB 300
```

управление передается строке с номером 300, которая является началом подпрограммы.

оператор **GOTO** оператор

Назначение: Оператор GOTO предназначен для передачи управления оператору с заданным номером строки.

Формат: GOTO <номер строки>
<номер строки> - любой существующий номер строки программы.

Действие: Оператор GOTO непосредственно передает управление другому оператору программы. Управление передается заданной строке, даже если в ней нет выполняемого оператора; например, это могут быть операторы REM, DATA

Применение: Оператор GOTO используется для безусловной передачи управления на строку с номером <номер строки> программы

Пример:

```
GOTO 360
```

управление передается строке программы с номером 360

директива **HIMEM** директива

Назначение: Директива HIMEM предназначена для установки верхней границы ОЗУ под программу на языке БЕЙСИК.

Формат: HIMEM <адрес ОЗУ>
<адрес ОЗУ> - числовая константа или переменная.

Действие: Директива HIMEM устанавливает верхнюю границу оперативной памяти для программы на языке БЕЙСИК. Это означает, что программа не может занимать память, адреса которой превышают адрес, заданный в директиве HIMEM. Таким образом можно ограничить память на программу на языке БЕЙСИК. Недоступную для программы на БЕЙСИКе память можно использовать для помещения программ и данных в машинных кодах и т.п. Возможно расширение объема памяти для программ на БЕЙСИКе. Смотри SCREEN. При начальном запуске или повторном нажатии клавиш СБР и БЛ-верхняя граница устанавливается на адрес 7FFFH.

Применение: Директива HIMEM применяется для ограничения памяти для программ на языке БЕЙСИК.

Пример:

```
HIMEM 30000
```

программа на языке БЕЙСИК может занимать память только до адреса 30000.

оператор **HOME** оператор

Назначение: Оператор HOME предназначен для очистки экрана и установки курсора в левой верхней позиции.

Формат: HOME

Действие: Оператор HOME очищает экран телевизора и устанавливает курсор в левую верхнюю позицию.

Пример:

```
HOME
```

очищается экран и курсор устанавливается в левую верхнюю позицию

оператор **IF/THEN** оператор

Назначение: Оператор IF/THEN предназначен для организации ветвлений в программе.

Формат: IF<условие> THEN <список операторов>
<условие> - любое выражение
<список операторов> - несколько операторов языка БЕЙСИК, разделенных двоеточиями, или любой номер существующей строки.

Действие: Вычисляется значение <условие>. Если <условие> истинно, то выполняются операторы, стоящие в строке после слова THEN, если <условие> ложно, то управление будет передано следующей строке программы. Условие может включать проверку самых различных условий с использованием как операций отношения, так и логических операций. Операторы, стоящие после слова THEN, также могут быть различными. В частности, если необходимо выполнить оператор GOTO, то название оператора можно опустить и просто указать номер строки программы, которой следует передать управление.

Применение: Оператор IF/THEN является основным оператором для организации ветвлений в программе.

Примеры:

```
IF X = 10 THEN 550
IF A = 0 THEN B = 1
IF X = 20 AND Y = 30 THEN GOSUB 900
IF X = 0 OR Y = 0 THEN Z = 50 : GOTO 330
IF S$ = "ДА" THEN PRINT "ВЫПОЛНИТЬ"
IF K$ = "КОНЕЦ" OR K$ = "К" THEN STOP
```

Функция **INKEY$** Функция

Назначение: Функция INKEY$ предназначена для ввода одного символа с клавиатуры.

Формат: INKEY$

Действие: Функция INKEY$ пытается прочесть символ из буфера клавиатуры. Если символ отсутствует, INKEY$ возвращает пустую строку (""). Если символ есть, он передается из буфера в строковую переменную. Нажатие любой клавиши формирует код, который может быть прочитан этой функцией.
Главным преимуществом этой функции является то, что она не воспроизводит на экране символ прочитанного кода, не ожидает символа и может читать любую клавишу.

Применение: Наиболее общим случаем использования INKEY$ является ввод символа без использования стандартного режима INPUT; можно преобразовывать символ перед выводом на печать, проверить вводимую команду или отказаться зафиксировать неверно нажатую клавишу. Возможна также некоторая "параллельная обработка" - если клавиша не нажата, то вы можете выполнить какое-либо другое задание. Наконец, эту функцию можно использовать просто для ожидания нажатия клавиши.

Пример:

```
10 S$=INKEY$
20 IF S$ = "" THEN 10
30 PRINT "НАЧАЛО ОБРАБОТКИ ВВЕДЕННОГО СИМВОЛА"
```

Функция **INP** функция

Назначение: Функция INP предназначена для чтения одного байта из порта ввода-вывода.

Формат: INP(<номер порта>)
<номер порта> - любое целое выражение в диапазоне от 0 до 255.

Действие. Функция INP возвращает результат ввода в виде целого числа от 0 до 255, т.е. внутреннинный код символа.

Применение: Функция INP совместно с оператором OUT предназначена для программирования ввода-вывода с нестандартным устройством.

Пример:

```
X=INP(10)
```

оператор **INPUT** оператор

Назначение: Оператор INPUT предназначен для ввода информации с клавиатуры.

Формат: INPUT ["<подсказка>";] <список переменных>
<подсказка> - поясняющий текст для вывода на экран
<список переменных> - переменные или элементы массива, разделенные запятыми.

Действие: Оператор INPUT позволяет вводить данные (константы или выражения) с клавиатуры непосредственно при выполнении программы. Значение введенных данных присваивается переменным, имена которых указывают вслед за оператором INPUT. Это могут быть как числовые, так и символьные переменные.

При выполнении оператора INPUT на экране дисплея возникает символ "?" . В ответ на этот вопрос с клавиатуры вводят данные через запятую, которые воспроизводится на экране в строку сразу после этого символа. Ввод данных заканчивается нажатием на клавишу ВК .

Если в операторе задана <подсказка>, то в этом случае на экран будет выведен текст "подсказки", а затем знак "?" . Однако, если <подсказка> отделена от списка переменных запятой? а не точкой с запятой, то вывод "?" отменяется.

Применение: Оператор INPUT используется для ввода данных с клавиатуры. При этом данные автоматически преобразовываются к указанному типу.

Пример:

```
INPUT   "ВВЕДИТЕ ДАННЫЕ",A,B,C,S$
```

Функция **INT** Функция

Назначение: Функция INT предназначена для выделения наименьшей целой части числа.

Формат: INT(X)
X - любое число или арифметическое выражение

Действие: Функция INT вычисляет аргумент X и преобразует его значение в наименьшее целое число.

Применение: Функция INT может использоваться для определения делимости двух арифметических выражений нацело.

Примеры:

```
X=INT(5.72) - X принимает значение 5
X=INT(-7.2) - X принимает значение -8
```

Функция **LEFT$** **Функция**

Назначение: Функция LEFT$ предназначена для выделения подстроки из строковой переменной, начиная с крайнего левого символа.

Формат: LEFT$ (<строка>,<длина>)
<строка> - любая строковая константа или выражение
<длина> - количество символов в диапазоне 0 - 255

Действие: Функция LEFT$ возвращает строку, содержащую заданное число самых левых символов данной строки. Если <длина> больше длины <строка>, то возвращается вся строка. Если <длина> равна 0, или <строка> является пустой строкой, то возвращается пустая строка.

Применение: Функция LEFT$ используется для выделения подстроки из строковой переменной, начиная с крайнего левого символа.

Пример:

```
X$=LEFT$("ABCDEF",3)
X$ получает значение "ABC"
```

Функция **LEN** функция

Назначение: Функция LEN предназначена для определения длины строкового выражения

Формат: LEN(<строка>)
<строка> - любое строковое выражение.

Действие: Функция LEN возвращает число, равное количеству символов в строке. Это означает, что функция LEN может вернуть число от 0 (для пустой строки) до 255 (максимально возможной строки).

Применение: Функция LEN используется для определения длины строковых данных там, где это необходимо.

Пример:

```
A$="ABCD"
X=LEN(A$)
```

X получает значение 4 .

оператор = оператор

Назначение: Оператор = основной оператор присваивания

Формат: <переменная>=<выражение>
<переменная> - любая переменная или элемент массива
<выражение> - любое выражение, имеющее тот же тип, что и <переменная>.

Действие: Оператор = присваивает переменной значение <выражения>. Типы <переменной> и <выражения> должны быть одинаковыми.

Применение: Оператор = используется для присваивания значений переменным.

Примеры:

```
X=10
Y=20
A$=B$+C$
```

оператор **LINE** оператор

Назначение: Оператор LINE предназначен для рисования линий и прямоугольников на графическом экране.

Формат: LINE [STEP] X,Y, [<режим>]

X,Y - любые арифметические выражения в диапазоне от 0 до 255 и от 32768 до -32767 для относительных координат

STEP - указывает, что координаты точки X,Y вычисляются относительно координат графического курсора.

<режим> - если не задан, то рисуется прямая;
если задан B, то рисуется прямоугольник;
если задан BF, то рисуется закрашенный прямоугольник
если задан BS то устанавливается режим вывода на дисплей символов увеличенного размера.

Действие: Оператор LINE рисует линии, вычерчивает прямоугольники или закрашенные прямоугольники в зависимости от наличия или отсутствия режимов B или BF. Режим BS позволяет выводить оператором PRINT символы увеличенного размера.
LINE X,Y,B - рисует прямоугольник на координатах его диагонали.
LINE X,Y,BF - рисует закрашенный прямоугольник.
Если указано STEP, то отсчет координат X и Y берется от текущего положения графического курсора, иначе отсчет координат X и Y берется от начальной точки графического экрана (0,0 - нижний левый угол).
LINE X,Y,BS устанавливает режим вывода на дисплей символов увеличенного размера. Символы строятся относительно текущих координат графического курсора. X,Y определяют размеры символов. Символы выводятся оператором PRINT. При режиме BS STEP не задается.

Применение: Оператор LINE применяется для рисования прямых линий, прямоугольников и закрашенных прямоугольников. А использование STEP позволяет перемещать одну и ту же фигуру в разные участки экрана, меняя только положение графического курсора оператором PLOT.
Режим BS в операторе LINE применяется для рисования на экране текстов из символов различных размеров.

Примеры:

```
LINE 50,100   рисует линию
LINE STEP 50,50,B рисует квадрат
LINE STEP 50,  100,  BF рисует закрашенный прямо-
угольник
LINE  2,3,  BS  :  PRINT "ТЕКСТ"   рисует   ТЕКСТ
увеличенного размера.
```

директивы **LIST/LLIST** директивы

Назначение: Директива LIST предназначена для вывода текста программы на экран, а LLIST - на печатающее устройство.

Формат:
LIST [<нач. номер строки>] [,<кон. номер строки>]
LLIST [<нач. номер строки>] [,<кон. номер строки>]

Действие: Директивы LIST и LLIST выводят текст программы, находящейся в настоящий момент в памяти. Если указан LIST, то вывод происходит на экран. Если указан LLIST, то текст программы выводится на печатающее устройство. Если не указаны параметры, то выводится вся программа. Если заданы начальный и конечный номера строк, то распечатывается только текст программы в заданном диапазоне. Если задан начальный номер строки , то распечатывается текст программы начиная с указанной строки и до конца программы. Если заданы запятая и конечный номер строки, то распечатается текст с начала программы и до указанного номера.

Применение: Директивы LIST и LLIST используется для вывода текста программы или его части на экран или печатающее устройство.

Примеры:

LIST
на экран выводится вся программа

LLIST 100,500
на печатающее устройство выводится часть программы в диапазоне номеров от 100 и до 500

LIST , 600
на экран выводятся строки программы от ее начала и до 600 строки включительно

LIST 800
на экран выводятся строки программы от строки 800 и до конца включительно

функция **LG** **функция**

Назначение: Функция LG предназначена для вычисления десятичного логарифма.

Формат: LG(X)
X - любое положительное числовое выражение.

Действие: Функция LG вычисляет десятичный логарифм (по основанию десять). Аргумент X должен быть больше 0.

Применение: Функция LG используется для вычисления десятичных логарифмов.

Пример:

```
Y=LG(X)
```

Фуикция **LOG** Фуикция

Назначение: Функция LOG предназначена для вычисления натурального логарифма.

Формат: LOG(X)
X - любое положительное числовое выражение.

Действие: Функция LOG вычисляет натуральный логарифм (по основанию e). Аргумент X должен быть больше 0.

Применение: Функция LOG используется для вычисления логарифма по любому основанию. Например, логарифм по основанию 10 вычисляется следующим образом:
LOG(X)/LOG(10)

Пример:

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

Y получает значение натурального логарифма от X.

Директива **MERGE** Директива

Назначение: Директива MERGE предназначена для обьединения текста программы, находящейся на ленте, с программой, расположенной в памяти.

Формат: MERGE [<имя файла>]
<имя файла> - строковое выражение, содержащее до 6 символов, определяющее имя файла.

Действие: Директива MERGE загружает с магнитной ленты программу и обьединяет ее с программой в ОЗУ. Строки добавятся к программе в памяти с учетом номеров строк, если номера строк загружаемой программы больше последнего номера строки программы, имеющейся в памяти. Смотри операторы DELETE, RENUM, CSAVE. Если не задано имя файла, выбирается первый по формату файл.

Применение: Директива MERGE используется для обьединения нескольких фрагментов программ в одну программу, а также в случае модульного программирования или построения сложных программ.

Пример:

MERGE "MOD1" - в оперативную память с кассеты загружается файл с именем "MOD1", в котором записана программа, и она добавляется к программе, расположенной в ОЗУ.

Функция **MID$** Функция

Назначение: Функция MID$ предназначена для выделения подстроки из строкового выражения.

Формат: MID$ (<строка>,<начало>[,<длина>])
<строка> - любое строковое выражение
<начало> - любое числовое выражение (от 1 до 255), задающее начальное положение подстроки в строковом выражении
<длина> - любое числовое выражение (от 1 до 255), определяющее число выделяемых символов

Действие: Функция MID$ может использоваться в обеих частях оператора присваивания и позволяет выделить строку символов длиной, указанной в аргументе <длина>, начиная с позиции <начало>. Результатом является строковое данное. Пустая строка всегда выдает пустую строку. Если длина не указана, подразумевается остаток строки (независимо от его длины).

Применение: Функция MID$ используется при обработке строковых данных.

Пример:

```
XS=MID$("ABCDEF",3,2)
XS получает значение "CD"
```

Директива **NEW** **директиве**

Назначение: Директива NEW предназначена для удаления программ и всех переменных из оперативной памяти

Формат: NEW

Действие: Директива NEW уничтожает все строки программы в памяти, все переменные и массивы.

Применение: Директива NEW используется для уничтожения программ, прежде чем вводить новую программу.

Пример:

NEW - уничтожение программы и всех переменных

оператор **ON GOTO/GOSUB** оператор

Назначение: Оператор ON GOTO или ON GOSUB предназначен для организации вычисляемых переходов.

Формат: ON <выражение> GOTO <список номеров строк>
GOSUB

<выражение> - любое целое арифметическое выражение от 0 до 255

<список номеров строк> - один или несколько существующих номеров строк в программе. Номера строк разделяются запятыми.

Действие: Операторы ON GOTO и ON GOSUB в зависимости от результата вычисления выражения реализует условную передачу управления на одну из строк программы, номер которой указан в списке номеров строк. При выполнении оператора сначала вычисляется значение выражения, от которого берется целая часть, которая и указывает в списке на номер строки. Если результат выражения равен 1, то управление будет передано на первую строку в списке, двум - на вторую строку в списке, и т.д.
Если же результат выражения меньше единицы или больше, чем количество номеров строк в списке, то выполняется оператор, непосредственно следующий за оператором ON GOTO или ON GOSUB.
Оператор ON GOSUB передает управление на вычисленный номер строки с возвратом по оператору RETURN на следующий после ON GOSUB оператор.

Применение: Операторы ON GOTO и ON GOSUB используется при программировании в тех случаях, когда необходим анализ множества вариантов. Программисту же необходимо так "подобрать" выражение, чтобы по результату его вычисления происходило переключение на строку с нужным номером.

Примеры:

```
ON X GOTO 150,160,170
ON X GOSUB 210,220,230
```

оператор **OUT** оператор

Назначение: Оператор OUT предназначен для записи одного байта в порт ввода-вывода.

Формат: OUT <номер порта>,<данные>
<номер порта> - любое целое выражение в диапазоне 0 - 255
<данные> - любое целое выражение в диапазоне 0 - 255.

Действие: Оператор OUT позволяет выдать в порт с номером <номер порта> результат выражения <данные> от 0 до 255.

Применение: Оператор OUT совместно с функцией INP предназначен для программирования ввода-вывода с нестандартным устройством.

Пример:

```
OUT 3,83
```

оператор **PAINT** оператор

Назначение: Оператор PAINT предназначен для закрашивания замкнутых областей.

Формат: PAINT [STEP] <X>,<Y> [,<цвет>] [,<граница>]
STEP - указывает, что координаты точки X,Y вычисляются относительно координат графического курсора
<X>,<Y> - определяет точку внутри графического экрана, с которой начинается закраска
<цвет> - арифметическое выражение от 0 до 15, определяющее цвет закраски
<граница> - арифметическое выражение от 0 до 15, определяющее цвет границы.

Действие: Оператор PAINT закрашивает области на экране. Область для закрашивания должна быть ограничена одним цветом. Если цвет закрашивания не совпадает с цветом границы, цвет границы должен быть указан. Если цвет закрашивания совпадает с текущим цветом изображения, установленным оператором COLOR, то можно опустить и атрибут цвета. Точка начала закраски может быть любой точкой внутри области (но не на границе).
Если указано STEP, то отсчет координат X и Y берется от текущего положения графического курсора, иначе отсчет координат X и Y берется от начальной точки графического экрана (0,0 - нижний левый угол).

Применение: Оператор PAINT позволяет закрашивать замкнутые области любым цветом. Если область незамкнутая, то PAINT "вытекает" через разрыв и может закрасить весь экран.

Пример:

```
PAINT 85,85,2
```

функция **PI** функция

Назначение: Функция PI возвращает значение константы π, равное 3.14159.

Пример:

```
X=A+PI/2
```

оператор **PAUSE** оператор

Назначение: Оператор PAUSE предназначен для задержки выполнения программы на указанное время.

Формат: PAUSE <N>
<N> - время в секундах от 0.001 до 65 сек.

Действие: Оператор PAUSE позволяет организовать паузу, т е прекратить выполнение программы на указанное время

Применение: Оператор PAUSE используется для задержки рисунков на экране телевизора, различных текстов и т.д.

Пример:

```
PAUSE 0.5
```

Функция

PEEK

Функция

Назначение: Функция PEEK предназначена для чтения байта данных из любой ячейки памяти.

Формат: PEEK (<адрес ОЗУ>)
(<адрес ОЗУ>)-любое целое арифметическое выражение в диапазоне от 0 до 1023 и от 17840 до 65535.

Действие: Функция PEEK возвращает число, равное содержимому байта памяти, расположенного по адресу <адрес ОЗУ>.

Применение: Функция PEEK может быть использована для получения результата работы программ на машинном языке.

Пример:

```
X=PEEK(A)
```

оператор **PLAY** оператор

Назначение: Оператор PLAY предназначен для того, чтобы исполнять музыку и создавать звуковые эффекты параллельно с выполнением программы.

Формат: PLAY [A1$] [,A2$] [,A3$]

A1$ A2$,A3$ строковые константы или переменные, содержащие музыкальные предложения на музыкальном макроязыке (MML) для каждого из трех каналов аппаратного звукового синтезатора. Каждая из них может быть опущена, но наличие хотя бы одной обязательно

Действие: Строковые выражения A1$,A2$,A3$ являются программой на MML для любого из трех независимых звуковых каналов и состоят из символов, после которых может следовать числовой параметр

C D E F G A B
до ре ми фа соль ля си

Эти символы вызывают исполнение соответствующей ноты в текущей октаве и текущей длительности.
Ноты можно модифицировать добавлением:
, + , - , . , <длительность>.
Знаки "#" и "+" указывают диез, знак "-" обозначает бемоль.
После этих добавлений может быть указана длительность ноты, если она отличается от стандартной (заданной "L") по следующему правилу: 1 - целая, 2 - половинная, 4 - четвертная и т.д. Далее можно поставить одну или несколько точек, каждая из которых увеличит длительность ноты на половину. O<число> определяет выбор октавы от 0 до 8. Октава остается неизменной до следующего "O". Если в начале строки не задано "O", то при запуске оператора PLAY по умолчанию устанавливается O4. O4 соответствует первой октаве
L<длительность> устанавливает стандартную длительность, т.е. длительность тех нот, для которых она не указана явно. Длительность задается от 1 до 64, а действительная длительность определяется величиной "1/<длительность>". При каждом запуске оператора PLAY устанавливается по умолчанию L4.
P<длительность> - специальная молчащая нота (пауза).
B задается в конце строки, если необходимо запустить мелодию снова.

При запуске оператора PLAY по указанным в нем каналам звучание от предыдущих операторов PLAY прекращается.

Примечание: Оператор PLAY применяется для музыкального оформления программы.

Пример:

```
PLAY "GG2P2O4B2O5C1R","CFDEEEFFFFE1R",
"FGGGGGGGGO4C1R"
```

оператор **PLOT** оператор

Назначение Оператор PLOT предназначен для установки в цвет фона или текущий цвет рисования точки на экране и для перемещения графического курсора.

Формат: PLOT [STEP] <X>,<Y>,<режим>

<X> - координата по горизонтали
<Y> - координата по вертикали
STEP - указывает, что координаты точки X,Y вычисляются относительно координат графического курсора
<режим> - если равен 1, то на экране точка высвечивается, 0 - гаснет, 2 - устанавливается графический курсор

Действие. Оператор PLOT позволяет погасить или высветить точку на экране с координатами X и Y. Если <режим> равен 1, то на экране точка высвечивается. 0 гаснет.
Если <режим> равен 2, то происходит установка графического курсора в указанную точку без каких-либо действий над точкой на экране.
Если указано STEP, то отсчет координат X и Y берется от текущего положения графического курсора, иначе отсчет координат X,Y берется от начальной точки графического экрана (0,0 - нижний левый угол).

Применение: Оператор PLOT используют для установки графического курсора, а также для "рисования" различных фигур.

Пример:

```
PLOT 50,25,1
```

на экране высвечивается точка с координатами X=50 и Y=25

Функция **POINT** Функция

Назначение: Функция POINT предназначена для определения кода цвета текущей точки на экране.

Формат: POINT (1)

1 - фиктивный аргумент.

Действие: Функция POINT выдает значение от 0 до 15, равное цвету точки изображения с текущими координатами графического курсора.

Применение: Одно из назначений POINT - определить, принадлежит ли точка изображению. Другое - это определение цвета рамки для PAINT.

Пример:

```
X=POINT(1)
```

оператор **POKE** оператор

Назначение: Оператор POKE предназначен для записи данных в любые последовательные ячейки памяти.

Формат: POKE <адрес ОЗУ>,<данные>[<данные>]....
<адрес ОЗУ>-целое арифметическое выражение в диапазоне от 0 до 255.

Действие: Оператор POKE позволяет записать в память, начиная с <адреса ОЗУ> значение <данные>
<адрес ОЗУ>-может принимать значения от 0 до 1023 и от 16640 до 65535.

Применение: Оператор POKE может быть использован для обработки внутримашинных данных в оперативной памяти

Пример:

```
POKE 6500, 112
```

Функция **POS** **Функция**

Назначение: Функция POS предиазначеиа для получения позиции курсора в строке.

Формат: POS (<аргумеит>)
<аргумент> - любое число, функцией не обрабатывается, т.е аргумент является фиктивным.

Действие: Результатом работы фуикцни POS является целое число, равиое номеру позиции последиего отпечатанного симвоза в текущей строке.

Применение: В иекоторых случаях функция POS может быть использована для определеиия текущего положения курсора на экране

Пример:

```
PRINT "ABCDEF"
X=POS(10)
```

X получает зиачение 6.

оператор **PRINT/LPRINT** оператор

Назначение: Оператор PRINT предназначен для вывода данных на экран, а LPRINT - на печатающее устройство.

Формат:

PRINT <список вывода>
LPRINT
<список вывода> - список выражений любого типа, разделенных запятыми или точкой с запятой; список может быть пустым.

Действие: Оператор PRINT выводит данные на экран, а LPRINT - на печатающее устройство.
Выводимые данные могут быть представлены выражениями любого типа или отсутствовать совсем. Выражения могут быть переменными или константами. Для вывода чисел в шестнадцатиричном виде используется знак "@". Разделителями между данными являются запятая или точка с запятой. Если в качестве разделителя используется "," , то под каждое выводимое значение отводится 14 позиций в строке, а если ";" - то отводится столько позиций, сколько необходимо.
Если после последнего операнда в операторе PRINT/LPRINT стоит разделитель, то при выполнении следующего оператора PRINT/LPRINT печать будет продолжена в той же строке. Если же разделителя нет, то печать начнется с новой строки.

Применение: Операторы PRINT и LPRINT используются для вывода данных на экран или печатающее устройство.

Примеры:

```
PRINT A$,B$,X,Y
LPRINT A$,B$,X,Y
```

оператор **PUT** оператор

Назначение: Оператор PUT предназначен для вывода на экран изображения, запомненного оператором GET.

Формат: PUT X,Y, <адрес массива> [,<режим>]
X и Y - любые целые арифметические выражения в диапазоне 0 - 255
<адрес массива> - адрес массива, из которого выбирается изображение
<режим> - целое арифметическое выражение, имеющее значение 0, 1 или 2.

Действие: Оператор PUT выводит на экран в точку с координатами X, Y изображение, запомненное оператором GET в массиве, адрес которого определяется параметром <адрес массива>. X,Y - координаты левого нижнего угла прямоугольника, размеры которого определены оператором GET при занесении графического изображения в массив.
<режим> - режим наложения изображений.
<режим> = 0 - сброс точек. На экран выводится изображение, цвет которого определяется цветом фона.
<режим> = 1 - установка точек. На экран выводится изображение с тем же цветом, с которым оно было запомнено оператором GET.
<режим> = 2 - абсолютная копия. На экран выводится копия прямоугольника, запомненного оператором GET.

Применение: Оператор PUT в сочетании с оператором GET позволяет перемещать по экрану графическую информацию.

Примеры:

```
PUT 50,50,ADDR(A(0)),0
PUT 100,50,ADDR(A(0)),1
PUT 50,100,ADDR(A(0)),2
```

оператор **READ** оператор

Назначение: Оператор READ предназначен для чтения данных из блока и присвоения конкретных значений переменным программы.

Формат: READ <перем1> [,<перем2>] ...
<перем1> - имена переменных или элементов
<перем2> массива.

Действие: Оператор READ читает данные, заданные оператором DATA, в очередную переменную. Этот процесс повторяется до исчерпания всех переменных списка в операторе READ. Данные из операторов DATA выбираются последовательно.

Применение: Оператор READ является единственным методом доступа к данным, заданным в программе с помощью оператора DATA. Оператор READ просматривает данные, заданные оператором DATA за один проход от начала текста программы к концу. Порядок считывания операторов DATA может быть изменен с помощью оператора RESTORE, этот оператор позволяет выбрать любой нумерованный оператор DATA в качестве "следующего" для чтения.

Пример:

```
10  DATA "БЕЙСИК", 1.5
20  READ A$,B
```

оператор **REM** оператор

Назначение: Оператор REM предназначен для размещения комментариев в тексте программы.

Формат: REM комментарий

Действие: После ключевого слова REM может следовать любой текст. Этот оператор должен быть единственным или завершающим в строке.

Применение: Оператор REM используется для документирования программы.

Пример:

```
10 REM ПРОГРАММА РАСЧЕТА ЛИНЕЙНЫХ УРАВНЕНИЙ
```

директива **RENUM** директива

Назначение: Директива RENUM предназначена для переиумерации всех строк программы.

Формат: RENUM [<номер строки>] [,<шаг>]
<номер строки> - числовая константа, по умолчанию значение номера строки равно 10
<шаг> - числовая константа, по умолчанию значение шага равно 10.

Действие: Директива RENUM используется для перенумерации всех строк программы. Первому оператору программы присваивается номер, равный <номеру строки> или 10, если <номер строки> не задан. Всем остальным операторам программы присваиваются номера в порядке возрастания с учетом значения шага переиумерации. Если шаг переиумерации не задан, то берется значение шага, равное 10.

Применение: Директива RENUM используется
- для получения более аккуратного текста программы,
- для изменения номеров строк программы и последующего слияния с другой программой;
- если недостаточно номеров строк для вставки новых программных строк.

Примеры:

RENUM 100,20
Перенумерация строк программы с шагом 20. Первой строке присваивается номер 100.
RENUM 200
Перенумерация строк программы с шагом 10. Первой строке присваивается номер 200.
RENUM ,100
Перенумерация строк программы с шагом 100. Первой строке присваивается номер 10.
RENUM
Перенумерация строк программы с шагом 10. Первой строке присваивается номер 10.

оператор **RESTORE** оператор

Назначение: Оператор RESTORE предназначен для указания номера строки, начиная с которой последующие операторы READ будут искать операторы DATA

Формат: RESTORE [<номер строки>]
<номер строки> любой существующий номер строки с оператором DATA.

Действие: Оператор RESTORE перемещает указатель в блоке данных, который сообщает оператору READ, какой из операторов DATA должен обрабатываться следующим. Оператор READ перемещает указатель вниз по мере ввода данных, пока не будут исчерпаны все операторы DATA. Оператором RESTORE можно установить указатель на любой номер строки с оператором DATA и последующие операторы READ начнут считывать константы с этого места. Если <номер строки> не задан, то оператор RESTORE устанавливает указатель на первый оператор DATA в программе.

Применение: Оператор RESTORE используется в случаях восстановления указателя DATA для повторного чтения данных.

Пример:

```
RESTORE
```

оператор **RETURN** оператор

Назначение: Оператор RETURN предназначен для возврата из подпрограмм.

Формат: RETURN

Действие: Оператор RETURN передает управление оператору, следующему за оператором, вызвавшим эту подпрограмму (GOSUB)

Применение: Оператор RETURN используется для окончания работы подпрограммы и возврата в основную (вызывающую) программу.

Пример:

```
RETURN
```

функция **RIGHT$** функция

Назначение: Функция RIGHT$ предназначена для выделения подстроки из строковой переменной, начиная с крайнего правого символа.

Формат: RIGHT$ (<строка>,<длина>)
<строка> - любая строковая константа или выражение
<длина> - количество символов в диапазоне 0 255.

Действие: Функция RIGHT$ возвращает строку содержащую заданное число самых правых символов данной строки. Если <длина> больше длины <строка>, то возвращается вся строка. Если <длина> равна 0, или <строка> является пустой строкой, то возвращается пустая строка.

Применение: Функция RIGHT$ используется для выделения подстроки из строковой переменной, начиная с крайнего правого символа. Эта функция может быть заменена функциями LEN и MID$, но использование функции RIGHT$ удобнее.

Пример:

```
X$=RIGHT$("БПЭВМ ВЕКТОР-06Ц",10)
X$ получает значение "ВЕКТОР-06Ц"
```

Функция **RND** Функция

Назначение: Функция RND предназначена для генерации псевдослучайных чисел

Формат: RND(1)

Действие: Результат работы функции - случайное число в диапазоне от 0 до 1.

Применение: С помощью функции RND можно писать на языке БЕЙСИК разнообразные программы моделирования и, как частный случай таких программ, игровые.

Пример:

```
X=RND(1)
```

X получает случайное значение от 0 до 1.

директива **RUN** директива

Назначение: Директива RUN предназначена для запуска программы на выполненне.

Формат: RUN [<номер строки>]
<номер строки> - любой реально существующий номер строки в программе

Действие: Директива RUN позволяет запустить программу на выполнение с первой программной строки, если не задан номер строки, или, если задан параметр <номер строки>, то выполнение программы начинается со строки с указанным номером. Надо иметь ввиду, что по директиве RUN перед запуском программы происходит инициализация переменных и массивов. Поэтому, если необходимо начать выполнение программы с заданного номера строки и при этом сохранить все переменные, не следует использовать директиву RUN. Для этого лучше использовать операторы GOTO и GOSUB.

Применение: Директива RUN используется для запуска программ на выполнеиие со стиранием значений всех массивов и переменных.

Примеры:

RUN
Запуск на выполнение программы, находящейся в памяти, с первой строки программы.
RUN 300
Запуск на выполнение программы, находящейся в памяти, с 300-ой строки программы.

оператор **SCREEN** оператор

Назначение: Оператор SCREEN предназначен для установки текущего режима экрана и других параметров ввода-вывода.

Формат: SCREEN <режим>,<список параметров>
<режим> - арифметическое выражение, определяющее режим работы оператора
<параметры> - список параметров, разделенных запятыми; для каждого режима свои параметры.

Действие: Действие оператора SCREEN определяется параметром <режим>:
1. SCREEN 0,N,C1, [,C2] [,C3] ... [,C16]
Оператор SCREEN с режимом "0" присваивает значения физических цветов (из 255 цветовой палитры), одному или нескольким математическим цветам (до 16 цветов), где
N - код математического цвета (от 0 до 15), начиная с которого производится изменение цвета (0 - цвет фона).
C1,C2...C16 - коды физических цветов (от 0 до 255), присваиваиваемые математическим цветам.
Коды физических цветов определяют цвет в соответствии с таблицей:

Первичные цвета	синий		зеленый			красный		
вес разряда	1/2	1/4	1/2	1/4	1/8	1/2	1/4	1/8
Код физич. цвета	128	64	32	16	8	4	2	1

Например, желтый цвет средней яркости получается из 1/2 красного и 1/2 зеленого цветов. Тогда код этого цвета равен 4+32=36.
Оператор SCREEN 0,3,36,16 присвоит математическим цветам 3 и 4 физические коды цвета 36 и 16 соответственно, что соответствует желтому цвету средней и малой интенсивности.
При начальной загрузке интерпретатора устанавливается следующее соответствие физических и математических цветов в таблице цветности:

Код мат. (дес.)	Код физ. (дес.)	табл.цветн. (двоичн)	Цвет
0	64	01000000	темно-синий
1	128	10000000	синий
2	16	00010000	зеленый
3	208	11010000	голубой
4	6	00000110	красный
5	134	10000110	малиновый
6	22	00010110	кирпичный
7	54	00110110	желтый
8	0	00000000	черный
9	197	11000101	фиолетовый
10	34	00100010	ярко-зеленый
11	192	11000000	ярко-синий
12	2	00000010	темно-красный
13	152	10011000	бирюзовый
14	82	01010010	серый
15	173	10101101	белый

Таблицу можно переустановить, выполнив следующий оператор:

```
SCREEN 0,0,64,128,16,208,6,134,22,54,0,197,34,192,2,152,82,173
```

2. SCREEN 2,N

Оператор SCREEN с режимом "2" устанавливает режимы обращения к ВИДЕО ОЗУ.
N - код режима в диапазоне от 0 до 15.
Если рассматривать код режима как двоичное число, то его можно представить в следующем виде:

```
X   X   X   X
!   !   !   !
!   !   !   !       управление плоскостью 1
!   !   !   ---------------------------------
!   !   !           управление плоскостью 2
!   !   -------------------------------------
!   !               управление плоскостью 3
!   -----------------------------------------
!                   управление плоскостью 4
```

Причем 0 - запрещает обращение к плоскости экрана
1 - разрешает обращение к плоскости экрана.
Например, SCREEN 2,N позволяет запретить изменения в выбранных экранных областях. Это дает возможность наложения одних изображений на другие и при соответствующем задании таблицы цветов оператором SCREEN 0,N,C1,C2,C3 ... C16 задать приоритет наложения одного изображения на другое, т.е. создавать до 4 независимых планов изображения.

3. SCREEN 3,M1 [,M2] [,M3] ... [,M8]
M1 ... M8 -целые числа, переменные или выражения, принимающие значения в диапазоне от 0 до 255.
Оператор SCREEN с режимом "3" устанавливает маску, которая влияет на выполнение оператора LINE с параметрами BF и BS (закрашенные прямоугольники и символы произвольного размера).
Весь экран условно разбивается на прямоугольники 8*8 точек. Каждый прямоугольник - это 8 байт. Нижний байт маскируется параметром M1, второй снизу маскируется параметром M2 и т.д. до восьмого.
Каждый параметр это десятичное представление одного байта. Причем "1" в байте разрешает менять цвет в этой точке, "0" - запрещает

4. SCREEN 4,N
Оператор SCREEN с режимом "4" устанавливает скорость записи и считывания на магнитную ленту.
N - скорость записи и считывания. 40(= N (=460
Скорость в БОД равна N*10.

5. SCREEN 5,N
Оператор SCREEN с режимом "5" устанавливает расположение клавиш латинского регистра.
N = 0 - "JCUKEN"
N = 1 - "QWERTY"

6. SCREEN 6,N
Оператор SCREEN с режимом "6" устанавливает тип принтера.
N = 0 - типа "ROBOTRON - CM 6329"
N = 1 - типа "EPSON FX - 85", "RAVI - 8010 M"

7. SCREEN 7,N
Оператор SCREEN с режимом "7" распечатывает содержимое экрана на принтере в одном из возможных графических режимов.
N = номер графического режима принтера (0(= [illegible]
Печать производится в формате 512*256 точек. При этом для получения различной плотности [illegible] изображения позиции, имеющие нулевой математический цвет (цвет фона) на печати не изображаются, [illegible] цвет от единицы до семи печатаются одной [illegible] ные (т.е. больше восьми) двумя последовательными точками

оператор **SCREEN** оператор

Применение: Оператор SCREEN применяется для установки соответствия математических и физических кодов цветов; для установки различных (до 4-х) плоскостей экрана; для ускорения ввода-вывода информации на магнитную ленту; для установки типа принтера; для вывода на печать экранного образа, а также позволяет использовать память, выделенную для ВИДЕО ОЗУ (2, 3, 4 экранные плоскости) для хранения программ и данных.

Примеры: SCREEN 0,5,2,152
Устанавливает математическим кодам цвета 5 и 6 физические коды цвета 2 и 152.

Закрашивание прямоугольника косыми линиями с помощью SCREEN 3:

```
 1 0 0 0 1 0 0 0  -  M8=136
 0 1 0 0 0 1 0 0  -  M7=68
 0 0 1 0 0 0 1 0  -  M6=34
 0 0 0 1 0 0 0 1  -  M5=17
 1 0 0 0 1 0 0 0  -  M4=136
 0 1 0 0 0 1 0 0  -  M3=68
 0 0 1 0 0 0 1 0  -  M2=34
 0 0 0 1 0 0 0 1  -  M1=17
SCREEN 3,17,34,68,136,17,34,68,136:PLOT 28,28
LINE 228,228,B:LINE 28,28,BF
```

```
SCREEN 2,15
```
Разрешает работать со всеми 4-мя плоскостями экрана

```
SCREEN 4,100
```
Устанавливает скорость обмена данными с магнитофоном 1000 бод.

```
SCREEN 7,1
```
Выводит образ экрана на печать.
Пример использования ВИДЕО ОЗУ для хранения программы или данных:

```
10 CLS
20 SCREEN 0,0,64,54,64,54,64,54,64,54,64,54,64,54,64,54,64,54
30 SCREEN 2,1:COLOR 1:HIMEM &E000
```
установится синий цвет фона, желтый цвет изображения, и память под программу или данные увеличится на 24 Кбайта и составит около 40 Кбайт.

Функция **SGN** Функция

Назначение: Функция SGN предназначена для определения знака числа

Формат: SGN (<выражение>)
<выражение> - любое числовое выражение.

Действие: Функция SGN возвращает значение -1, если <выражение> меньше нуля, 0 - если <выражение> равно нулю и 1 - если <выражение> больше нуля.

Применение: В основном функция SGN применяется при умножении одного числа на знак другого числа. Она позволяет также приравнять нулю результат, если один из операндов равен нулю (без использования оператора IF).

Пример:

```
PRINT 2*X*Y*SGN(Y)
```

Функция **SIN** Функция

Назначение: Функция SIN предназначена для вычисления синуса угла, заданного в радианах.

Формат: SIN (X)
X - любое арифметическое выражение.

Действие: Функция SIN преобразовывает аргумент и рассматривает его как угол в радианах. Используя это значение, она возвращает синус угла.

Применение: Функция SIN используется в расчетах.

Пример:

```
Y=SIN(X)
```

Функция **SPC** функция

Назначение: Функция SPC предназначена для печати пробелов и используется только в операторах PRINT или LPRINT.

Формат: PRINT
SPC (<длина>) ...
LPRINT
<длина> - арифметическое выражение в диапазоне от 0 до 255.

Действие: Функция SPC используется только в операторах PRINT или LPRINT и печатает указанное количество пробелов.

Применение: С помощью функции SPC производится вывод пробелов.

Пример:

```
PRINT SPC(10);A;SPC(5);B$;SPC(5);C
```

Функция **SQR** функция

Назначение: Функция SQR предназначена для вычисления квадратного корня.

Формат: SQR (<выражение>)
<выражение> - любое неотрицательное арифметическое выражение.

Действие: Функция SQR вычисляет квадратный корень заданного арифметического выражения.

Применение: Функция SQR используется в расчетах.

Пример:

```
Y=SQR(X)
```

оператор **STOP** оператор

Назначение: Оператор STOP предназначен для прекращения выполнения программы и перевода интерпретатора в непосредственный режим.

Формат: STOP

Действие: Оператор STOP прекращает выполнение программы и переводит интерпретатор в непосредственный режим. При этом на экран телевизора выводится сообщение: "СТОП в строке XX" , где XX - номер строки, в которой произошел останов. Работу прерванной программы можно продолжить, выполнив директиву CONT.

Применение: Оператор STOP обычно применяют для останова в программе и выдачи сообщения о том, что пользователю надо что-то сделать. Кроме того, оператор STOP очень удобен при отладке программ.

Пример:

```
STOP
```

Функция **STR$** **Функция**

Назначение: Функция STR$ предназначена для преобразования числа в символьную строку, содержащую это число.

Формат: STR$(X)
X - арифметическое выражение.

Действие: Функция STR$ возвращает значение строковой переменной, которое представляет собой число в символьном формате. Другими словами, функция STR$ служит для преобразования числовых данных в символьные. Совместно с функцией VAL функция STR$ осуществляет преобразование числовых данных в символьный формат и обратно в числовой.

Применение: Функция STR$ используется для преобразования числовых данных в символьные.

Пример:

```
S$=STR$(A)
```

Функция **TAB** Функция

Назначение: Функция TAB предназначена для перемещения курсора в строке или указания позиции, с которой необходимо начать печать.

Формат: TAB(X)
X - арифметическое выражение в диапазоне 0 - 255.

Действие: Функция TAB - это функция горизонтальной табуляции, которая позволяет переместить курсор в заданную позицию в строке. Аргумент X не должен превышать значение 255. Он указывает, сколько позиций необходимо отступить от левого края строки.

Применение: Функция TAB обычно применяется в операторах PRINT и LPRINT для установки позиции для печати.

Пример:

```
PRINT TAB(5);A$
```

Функция **TAN** **Функция**

Назначение: Функция TAN предназначена для вычисления тангенса угла, заданного в радианах.

Формат: TAN (X)
X - любое арифметическое выражение.

Действие: Функция TAN преобразовывает аргумент и рассматривает его как угол в радианах. Используя это значение, она возвращает тангенс угла.

Применение: Функция TAN используется в расчетах.

Пример:

```
Y=TAN(X)
```

Функция **USR** Функция

Назначение: Функция USR предназначена для обеспечения доступа к подпрограммам на машинном языке.

Формат: X=USR(<адрес>)
<адрес> указывает точку входа в программу.

Действие: Функция USR передает управление программе на машинном языке, предварительно записанной в ОЗУ с помощью оператора POKE или директивы LOAD. Результатом функции является байт данных, присваиваемый переменной X, который был в регистре A микропроцессора при выходе из подпрограммы по команде RET (C9H). Содержимое регистров в подпрограмме сохранять не требуется.

Применение: Функция USR позволяет выполнять из БЕЙСИКа программы в машинных кодах.

Пример:

```
Y=USR(X)
```

функция VAL функция

Назначение: Функция VAL предназначена для преобразования строкового данного в число. Строковое данное должно иметь формат числа.

Формат: VAL(X$)
X$ - строковая переменная или константа, содержащая число.

Действие: Функция VAL возвращает численное значение строкового данного. Другими словами, функция VAL служит для преобразования символьных данных в числовые. Совместно с функцией STR$ функция VAL осуществляет преобразование числовых данных в символьный формат и обратно в числовой.

Применение: Функция VAL используется для преобразования символьных данных в числовые.

Пример:

```
X=VAL(S$)
```

директива VERIFY директива

Назначение: Директива VERIFY предназначена для проверки правильности записи программы на магнитную ленту.

Формат: VERIFY <имя файла>
<имя файла> - строковое выражение, определяющее имя файла на магнитной ленте. Имя файла не должно превышать 6 символов.

Действие: По директиве VERIFY осуществляется поиск файла с указанным именем на магнитной ленте. Если файл с указанным именем есть, то осуществляется сравнение программы в ОЗУ с программой на ленте. При обнаружении различий выдается сообщение об ошибке.

Применение: Директива VERIFY используется для проверки программных файлов на магнитной ленте.

Пример:

```
VERIFY "PROG"
```

6. ПРИЕМЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ БЕЙСИК

6.1. Разработка программы

Разработку программы всегда нужно начинать с цели, т.е. определить, что необходимо в конце концов получить. Результаты выполнения программы называют выходными данными и они могут быть выведены на экран , печатающее устройство или магнитофон. Выходными данными могут быть расчетные таблицы, графики, рисуики, музыка и т.п.

После того, как определена цель, необходимо определить данные для решения поставленной цели. Эти данные называют входными данными. Входными данными могут быть числа, строки, ноты для музыки и т.д.

Если известны входные и выходные данные, необходимо описать процесс получения результата в виде выходных данных из входных. Такое описание называется алгоритмом решения задачи.

Разработка алгоритма - сложный процесс и сводится обычно к тому, чтобы разбить поставленную задачу, которая может быть достаточно сложной, на части, которые чаще всего описывают законченный этап алгоритма. После этого достаточно детализировать части, чтобы окончательно решить алгоритм поставленной задачи.

На основании разработанного алгоритма можно начинать разработку программы. Разработка программы заключается в том, чтобы описать алгоритм задачи на языке БЕЙСИК. Лучше всего начать программирование поставленной задачи с программирования каждой части алгоритма, которая в программе может быть описана как подпрограмма. После этого достаточно описать алгоритм на языке БЕЙСИК в виде вызова необходимых подпрограмм (частей алгоритма).

Такой подход позволяет отдельно отладить подпрограммы (части), а затем уже сконцентрировать свое внимание на решении алгоритма в целом. Кроме того, при таком подходе намного легче локализовать ошибки в алгоритме программы и можно достаточно быстро отладить программу. И еще одно важное удобство при таком подходе : при внесении изменений в программу достаточно определить, в какую подпрограмму надо внести изменения и изменить эту подпрограмму.

Важным моментом в программировании является документирование текста программы, т.е. внесение в текст программы комментариев (оператор REM). Хорошее документирование текста программы помогает не только более быстро и качественно отлаживать программы, но позволяет еще и довольно легко вносить изменения в уже давно разработанные программы.

Можно определить три уровня документирования, которых следует придерживаться.

Первый уровень - это общее описание программы в начальном блоке комментариев, в котором обычно описывают: имя программы, даты создания и последующих модификаций, номер версии, имя автора, входные и выходные данные.

Второй уровень документирования заключен в каждой

подпрограмме, если алгоритм решения был разбит на части (подпрограммы), в котором необходимо описать назначение подпрограмм и какие переменные вводятся в подпрограмму и какие выводятся.

Третий уровень документирования - комментирование программных строк. Далеко не каждая программная строка требует комментария. Однако при использовании в строке необычной комбинации операторов или какого-то хитроумного приема программирования желательно снабдить эту строку комментарием.

Описанный метод разработки программ позволит получить качественные программы, легко читаемые и легко модифицируемые.

6.2. Использование массивов и циклические вычисления

В программах часто возникает необходимость выполнять операции над множеством определенных данных, которые объединяются в массив. В языке БЕЙСИК под массивом понимается набор переменных, которые имеют общее обозначение и называются элементами массива.

Для использования массивов в программе их необходимо описать в операторе DIM. При описании массивов указывают количество элементов каждой размерности. Это позволяет БЕЙСИКу определить необходимую оперативную память для хранения значений элементов массива.

Использование массивов позволяет выбирать его элементы посредством задания порядкового номера в виде выражения. Каждый раз, когда переменные участвуют в одних и тех же операциях, следует обдумать возможности использования массивов таким образом, чтобы операцию обработки можно было записать один раз, задав номер обрабатываемого элемента в виде выражения.

Массивы переменных позволяют эффективно использовать циклы FOR/NEXT для обработки элементов массивов.

Пример: Вычислить сумму 20 чисел (массив К) и полученный результат (S) вывести на экран.

```
10   DIM K(20)
20   S=0
30   FOR I=1 TO 20
40   S=S+K(I)
50   NEXT I
60   PRINT "РЕЗУЛЬТАТ="; S
```

В строке 10 описывается массив, состоящий из 20 элементов. В строке 20 переменной S присваивается значение 0. Строки 30 - 50 организуют цикл, который выполняется 20 раз, причем переменная цикла I увеличивается на единицу при каждом прохождении цикла. В результате при первом прохождении цикла к переменной S добавится первый элемент массива К, при втором - второй элемент массива К и т.д. По окончании цикла переменная S будет содержать сумму 20 элементов массива К.

Использование массивов позволяет динамически обращаться к элементам массивов, указывая номер элемента массива в виде арифметического выражения. Это, а также использование операторов

цикла (FOR/NEXT), позволяет применять в программах разнообразные алгоритмы для решения самых различных задач.

6.3. Вычисляемые переходы

Для выполнения перехода в программах используются операторы условного перехода IF/THEN или безусловного перехода GOTO. В ряде случаев, однако, возникают ситуации, когда переход определяется арифметическим выражением, получающимся в результате решения алгоритма.

Оператор ON GOTO позволет определить строку, на которую необходимо осуществить переход, в зависимости от значения арифметического выражения.

ON A GOTO 100, 200, 300

Если A=1 то переход будет осуществлен на строку с меткой 100, если A=2, то на строку с меткой 200, если A=3, то на строку с меткой 300. При любом другом значении A будет выполняться следующий после ON GOTO оператор.

Оператор ON GOTO часто используют для организации ветвлений в программе в зависимостк от вычисленного арифметического выражения.

Пример: Необходимо определить строку программы, на которую нужно осуществить переход по введенному режиму. Если режим задан неверно, выдать сообщение об ошибке и повторить ввод режима.

```
10   INPUT "ВВЕДИТЕ РЕЖИМ"; A
20   ON A GOTO 40, 50, 60
30   PRINT "НЕВЕРНЫЙ РЕЖИМ": GOTO 10
40   B=X^2+Y^2 : GOTO 70
50   B=X^2-Y^2 : GOTO 70
60   B=2*X : GOTO 70
```

6.4. Обработка строк

Значениями символьных переменных являются последовательности символов. В ряде операторов возникает необходимость оперировать какой-либо частью значения символьной переменной, т.е. ее подстрокой. Для этого в БЕЙСИКе имеются специальные функции выделения подстрок. Функции LEFT$(), RIGHT$() и MID$() позволяют выделять подстроки из строковых переменных. Причем эти функции могут располагаться слева и справа от знака присваивания, а также участвовать в сцеплении строк (операция "+").

Во многих случаях при обработке строк необходимо знать их длину. Для этого используют функцию LEN.

Надо иметь ввиду, что при обработке строк они перемещаются в строковое пространство. Строковое пространство является областью памяти , резервируемой для хранения строковых данных. Область памяти для строковых данных выделяется оператором CLEAR. В любой момент сформированная новая строка данных размещается на части строкового пространства. Если объем оставшегося строкового пространства недостаточен, то начинается автоматический процесс очистки строкового пространства и работа программы приостанавливается до завершения процесса очистки. Время очистки

строкового пространства зависит от размера строкового пространства и размеров строковых данных. При программировании часто возникает необходимость преобразования символьных данных в числовые и наоборот. Функция VAL() осуществляет преобразование символьных данных в числовые, а функция STR$() преобразовывает числа в строку символов.

Часто бывает необходимым получить значение кода какого-либо знака. Это можно сделать с помощью функции ASC(). Функция CHR$() выполняет действие, обратное действию функции ASC().

6.5. Организация подпрограмм

При составлении программ очень часто возникает необходимость многократного выполнения некоторых операций. Кроме того, при составлении программ удобно разбить программу на части (модули), которые выполняют какой-то законченный раздел алгоритма. Средством программирования повторяющихся операций, а также обращение к группе операторов, выполняющих какую-то часть алгоритма, является использование подпрограмм.

Если известно, что в различных местах программы понадобится исполнять одну и ту же последовательность операторов, то эта последовательность оформляется как отдельная вспомогательная программа, называемая подпрограммой. В тех местах основной программы, где необходимо выполнение этой последовательности, помещается лишь обращение к подпрограмме, обеспечивающее ее выполнение и возврат к продолжению основной программы.

Подпрограммой в БЕЙСИКе может являться любая последовательность строк программы, завершающаяся специальным оператором конца подпрограммы RETURN. Обращение к подпрограмме может осуществляться из любого места программы с помощью оператора вызова подпрограммы GOSUB или по оператору вычисляемого перехода на подпрограмму ON GOSUB.

После выполнения подпрограммы управление передается на оператор, следующий после операторов GOSUB или ON GOSUB.

Подпрограммы могут быть вложенными, т.е. из одной подпрограммы возможно обращение к другой. При этом сохраняются все адреса возврата. При выполнении оператора RETURN список этих адресов уменьшается на один адрес и происходит возврат к оператору, следовавшему за последним исполнявшимся оператором GOSUB.

6.6. Применение графических операторов

Одной из ярких особенностей интерпретатора БЕЙСИК и БПЭВМ "ВЕКТОР-06Ц" является широкий набор графических операторов.

В графическом режиме пользователю доступны 256 x 256 графических точек. Движение по графическим точкам экрана осуществляется с помощью графического курсора, координаты которого по оси X и по оси Y могут изменяться от 0 до 255.

Установка графического курсора может быть осуществлена с помощью оператора PLOT. С помощью оператора LINE можно построить прямую из текущей точки, в которой находится графический курсор, до точки, определенной в операторе LINE. С помощью оператора LINE

можно построить прямоугольник или закрашенный прямоугольник, а также указать размеры шрифта для выводимого текста с помощью операторов PRINT.

Для управления цветом используется оператор COLOR, с помощью которого можно задать цвет изображения, цвет фона и цвет рамки. Любые графические операторы будут использовать цвета изображения и фона, заданные в последнем исполненном операторе COLOR.

Для закраски замкнутых фигур используется оператор PAINT. Надо только иметь ввиду, что в случае незамкнутой фигуры оператор PAINT может закрасить весь экран, т.е. цвет через разрыв начнет "вытекать" за границы фигуры.

Применение оператора SCREEN позволяет получать дополнительные возможности обработки графики и цвета. SCREEN 0 позволяет изменить соответствие математических и физических кодов цвета, что дает возможность при одних и тех же математических кодах цвета менять цвет участков экрана. SCREEN 2 позволяет строить изображения в различных плоскостях экрана (до 4-х), что позволяет получать объемные изображения различной цветовой гаммы. SCREEN 7 позволяет отображать содержимое экрана на печатающее устройство.

Применение оператора CIRCLE позволяет рисовать окружности и дуги.

Использование оператора GET позволяет запоминать в массиве графическую информацию с экрана, а оператором PUT вывести графическую информацию из массива на экран. Оператор GET в сочетании с оператором PUT позволяет перемещать по экрану графическую информацию, что позволяет программировать красочные игры.

6.7. Программирование музыки

Наиболее простым способом вывода звука является сигнал BEEP. На это время прекращается выполнение программы.

Наиболее эффективным способом получения музыки является музыкальный макроязык, используемый оператором PLAY, который позволяет одновременно описать до трех исполняемых "голосов" ,так что музыка будет автоматически исполняться параллельно с выполнением программы.

Каждый из трех "голосов" имеет свои независимо вводимые характеристики, которые используются в синтезируемой музыке. Они включают ноту и ее длительность, длительность пауз, октаву. Комбинирование всех этих возможностей позволяет получить желаемую музыку.

6.6. Программирование ввода-вывода

При разработке программ всегда необходимо осуществлять обмен данными между ОЗУ и другими устройствами машины. Для БПЭВМ "ВЕКТОР-06Ц" основными операциями ввода-вывода являются:

- ввод данных с клавиатуры;
- запись и чтение программ с магнитной ленты;
- запись и чтение данных в ОЗУ с магнитной ленты;
- вывод данных на экран и печатающее устройство;
- вывод программ на экран и печатающее устройство.

Для работы программы часто требуется информация, которую пользователю необходимо ввести с клавиатуры с помощью оператора INPUT или функции INKEY$. При вводе информации с помощью INPUT на экран выводится запрос на ввод информации. После чего пользователь начинает ввод информации. Такой ввод очень удобен и поэтому оператор INPUT часто применяется при разработке программ.

Функция INKEY$ осуществляет ввод одного символа из буфера клавиатуры и передает его в строковую переменную. Если буфер пуст, INKEY$ возвращает пустую строку (""). Нажатие любой клавиши формирует код, который может быть прочитан этой функцией. Основным достоинством INKEY$ является то, что символ не высвечивается на экране и нет ожидания нажатия клавиши. Эту функцию часто используют для управления программой не останавливая ее работу.

Программа может выполняться только в оперативной памяти. В оперативную память программа попадает в результате ввода ее с клавиатуры. После того, как она введена, ее можно исполнить. Однако, после набора программы ее желательно было бы сохранить, чтобы и в дальнейшем ее можно было бы использовать. Для этого существует директива записи программы на магнитную ленту CSAVE и директива чтения программы с магнитной ленты в ОЗУ CLOAD.

В языке БЕЙСИК есть директива записи области памяти на магнитную ленту (BSAVE) и директива чтения данных с магнитной ленты в ОЗУ (BLOAD), записанных директивой BSAVE. Используя директивы BSAVE и BLOAD можно осуществлять обмен данными между ОЗУ и магнитной лентой. Это позволяет сохранить для дальнейшего использования данные, полученные в результате решения задачи.

Операторы PRINT и LPRINT позволяют выводить данные соответственно на экран телевизора и печатающее устройство.

Директивы LIST и LLIST позволяют выводить текст программы, находящейся в ОЗУ соответственно на экран и печатающее устройство.

7. СООБЩЕНИЯ ОБ ОШИБКАХ

При обнаружении ошибок во время выполнения программы на экран телевизора выводятся сообщения в виде:

<текст ошибки> ошибка в строке <номер строки>: <номер оператора>,

где <текст ошибки> - текст, поясняющий характер ошибки,
<номер строки> - номер строки программы, в которой обнаружена ошибка
<номер оператора> - номер ошибочного оператора в строке.
Например:
синтаксическая ошибка в строке 200:3
В третьем операторе 200-й строки обнаружена синтаксическая ошибка

При выполнении программы могут быть обнаружены следующие ошибки.

Сообщение	Пояснение
NEXT без FOR	Обнаружен оператор конца цикла NEXT без оператора начала цикла FOR.
синтаксическая	Обнаружена синтаксическая ошибка.
RETURN без GOSUB	Оператор RETURN не соответствует никакому предварительно выполненному GOSUB.
мало данных в DATA	Операторам READ требуется больше данных, чем это было предусмотрено в операторах DATA.
неверный аргумент	Аргумент находится вне области допустимых значений.
переполнение	Результат вышел за допустимые пределы.
нет памяти	Нет свободной памяти для размещения текста программы или ее переменных.
нет строки	Номер строки в GOTO, GOSUB, IF/THEN и т.д. не соответствует никакой существующей строке программы.
неверный индекс	На элемент массива ссылаются с индексом, превышающим размерность массива, или с неправильным количеством индексов.
повторное описание	Повторно описан массив оператором DIM.
деление на ноль	Обнаружено деление на 0 или возведение 0 в отрицательную степень.

Сообщение	Пояснение
только в программе	Использование функции (оператора) в режиме непосредственного счета недопустимо
несоотв.данных	В выражении обнаружено несоответствие типов данных
мал буфер	Выделено недостаточно памяти для символьных данных
X$>255	Предпринята попытка создать строчное значение длиннее 255 символов
сложно	Выражение слишкои сложное, рекомендуется разделить его на несколько более простых
нельзя	Применение директив, операторов функций в такой интерпретации запрещено
нет DEF	Предпринято обращение к функции пользователя до ее определения с помощью DEF FN.

ПРИЛОЖЕНИЕ 1

Список зарезервированных слов
и кодов их внутреннего представления.

ABS	B0	GET	DE	PLAY	E2
ACS	D8	GOSUB	8C	PLOT	92
ADDR	D5	GOTO	88	POINT	C6
AND	A9	HIMEM	D2	POKE	94
ASC	C1	HOME	CD	POS	B4
ASN	D4	IF	8A	PRINT	95
ATN	BC	INKEY$	C7	PUT	DF
AUTO	D1	INP	B3	READ	86
BEEP	CA	INPUT	84	REM	8E
BLOAD	E1	INT	AF	RENUM	D7
BSAVE	E0	LEFT$	C3	RESTORE	8B
CHR$	C2	LEN	BE	RETURN	8D
CIRCLE	E4	LINE	93	RIGHT$	C4
CLEAR	99	LIST	98	RND	B6
CLOAD	9A	LG	D9	RUN	89
CLS	80	LLIST	D8	SCREEN	DC
COLOR	DD	LOG	B7	SGN	AE
CONT	97	LPRINT	DA	SIN	BA
COS	B9	MERGE	D0	SPC	8F
CSAVE	9B	MID$	C5	SQR	B5
CUR	87	NEW	9C	STEP	A3
DATA	83	NEXT	82	STOP	8F
DEF	96	NOT	A2	STR$	BF
DELETE	CF	ON	91	TAB	9D
DIM	85	OR	AA	TAN	BB
EDIT	CE	OUT	90	THEN	A1
EXP	B8	PAINT	E3	TO	9E
FN	A0	PAUSE	CB	USR	B1
FOR	B1	PEEK	BD	VAL	C0
FRE	B2	PI	D6	VERIFY	CC

Коды даны в шестнадцатеричном представлении.

ПРИЛОЖЕНИЕ 2

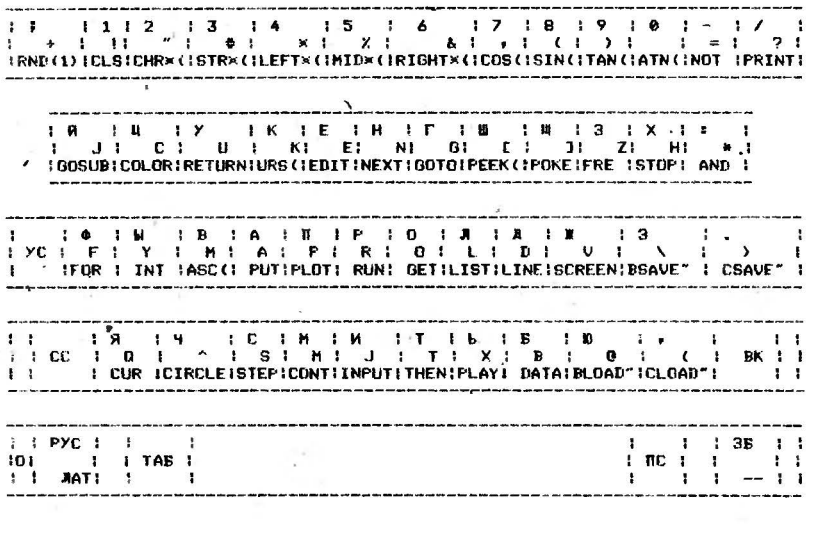
Соответствие клавиш служебным словам, вызываемым последовательным нажатием клавиши АР2 и одной из клавиш.

Ир ABS	5 HIMEM	PLAY
; ACS	GET	Sл PLOT
8 ADDR	Мл GOSUB	POINT
Ер AND	Iл GOTO	Uл POKE
Зр ASC	O HOME	Мр POS
7 ASN	Кл IF	Vл PRINT
Ур ATN	* INKEY$	PUT
4 AUTO	Лр INP	Gл READ
- BEEP	Ел INPUT	Oл REM
BLOAD	Хр INT	: RENUM
BSAVE	Эр LEFT$	Lл RESTORE
Шр CHR$	Вр LEN	Nл RETURN
CIRCLE	Тл LINE	Щр RIGHT$
Zл CLEAR	Yл LIST	Oр RND
[CLOAD	< LG -	Jл RUN
Áл CLS	> LLIST	? SCREEN
@ COLOR	Пр LOG	Гр SGN
Xл CONT	= LPRINT	Cр SIN
Рр COS	3 MERGE	Юр SPC(
\ CSAVE	Чр MID$	Нр SQR
Нл CUR	] NEW	Др STEP
Dл DATA	Cл NEXT	Рл STOP
Wл DEF	Цр NOT	Ьр STR$
2 DELETE	Rл ON	` TAB(
Fл DIM	Фр OR	Тр TAN
1 EDIT	Qл OUT	Ер THEN
Яр EXP	PAINT	TO
Ар FN	. PAUSE	Йр USR
Вл FOR	Кр PEEK	Мр VAL
Кр FRE	9 PI	/ VERIFY

(Индекс р означает русский регистр, индекс л означает латинский регистр, без индекса - знаки и цифры)

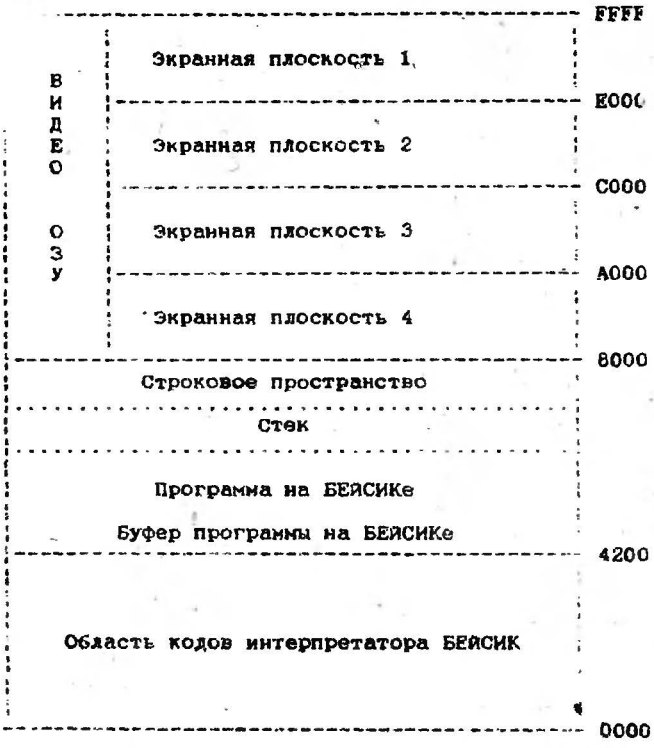
Продолжение

Соответствие клавиш служебным словам вызываемым одновременным нажатием клавиш УС , ОС и клавиши основного поля клавиатуры.



ПРИЛОЖЕНИЕ 3

Распределение памяти



Интерпретатор БЕЙСИК занимает около 16 Кбайт памяти. Под ВИДЕО ОЗУ выделено 32 Кбайта памяти для управления четырьмя экранными плоскостями. Под программу на БЕЙСИКЕ выделяется 16 Кбайт памяти. В случае необходимости можно увеличить память под программу за счет памяти экранных плоскостей 2,3,4, т.е получить дополнительно еще 24 Кбайта памяти.

ПРИЛОЖЕНИЕ 4.

Таблица кодов

Обозначение клавиши	Десятичный код	Шестнадцатиричный код	Обозначение клавиши	Десятичный код	Шестнадцатиричный код
F1	00	00	пробел	32	20
F2	01	01	!	33	21
F3	02	02	"	34	22
F4	03	03	#	35	23
	04	04	$	36	24
	05	05	%	37	25
	06	06	&	38	26
	07	07	'	39	27
←	08	08	(	40	28
TAB	09	09	)	41	29
ПС	10	0A	*	42	2A
	11	0B	+	43	2B
К	12	0C	,	44	2C
ВК	13	0D	-	45	2D
	14	0E	.	46	2E
	15	0F	/	47	2F
	16	10	0	48	30
	17	11	1	49	31
	18	12	2	50	32
	19	13	3	51	33
	20	14	4	52	34
	21	15	5	53	35
	22	16	6	54	36
	23	17	7	55	37
→	24	18	8	56	38
↑	25	19	9	57	39
↓	26	1A	:	58	3A
АР2	27	1B	;	59	3B
	28	1C	<	60	3C
	29	1D	=	61	3D
	30	1E	>	62	3E
СТР	31	1F	?	63	3F

Продолжение

Обозначение клавиши	Десятичный код	Шестнадцатиричный код	Обозначение клавиши	Десятичный код	Шестнадцатиричный код
@	64	40	Ю	96	60
A	65	41	А	97	61
B	66	42	Б	98	62
C	67	43	Ц	99	63
D	68	44	Д	100	64
E	69	45	Е	101	65
F	70	46	Ф	102	66
G	71	47	Г	103	67
H	72	48	Х	104	68
I	73	49	И	105	69
J	74	4A	Й	106	6A
K	75	4B	К	107	6B
L	76	4C	Л	108	6C
M	77	4D	М	109	6D
N	78	4E	Н	110	6E
O	79	4F	О	111	6F
P	80	50	П	112	70
Q	81	51	Я	113	71
R	82	52	Р	114	72
S	83	53	С	115	73
T	84	54	Т	116	74
U	85	55	У	117	75
V	86	56	Ж	118	76
W	87	57	В	119	77
X	88	58	Ь	120	78
Y	89	59	Ы	121	79
Z	90	5A	З	122	7A
[	91	5B	Ш	123	7B
\	92	5C	Э	124	7C
]	93	5D	Щ	125	7D
^	94	5E	Ч	126	7E
	95	5F	ЗБ	127	7F