I. ОБЩИЕ СВЕДЕНИЯ

a_l_e_x

I.I. Введение в язык БЭЙСИК 02.

I.I.I. Сравнение языка БЭЙСИК 02 с БЭЙСИК 01.

Язык БЭЙСИК 02 разработан на основе опыта работы с языком БЭЙСИК 01.

Особое внимание было обращено на усовершенствование синтаксиса многих операторов, а также разработку новых операторов, способствующих наилучшему составлению, оформлению и отладке программ на языке БЭЙСИК 02.

Несмотря на то, что некоторые операторы БЭЙСИК 01 претерпели синтаксические изменения в БЭЙСИК 02, данный язык программирования почти полностью сохраняет черты предыдущего варианта.

Программы, составленные на языке БЭЙСИК 01, могут быть введены и использованы в системе с интерпретатором БЭЙСИК 02 почти без модификаций. Существует лишь несколько исключений. Сравнение языка БЭЙСИК 02 и языка БЭЙСИК 01 приведено в приложении.

I.I.2. Типы элементов в языке БЭЙСИК 02.

Язык программирования БЭЙСИК 02 состоит из элементов различных типов и включает в себя операторы, команды, знаки операций и набор функций, входящих в систему. Наиболее важными элементами являются операторы и команды. Операторы являются программируемыми элементами для составления программ на языке БЭЙСИК 02. Команды используются для управления системой непосредственно с клавиатуры и не программируются. Функции, входящие в систему, и знаки операций используются для составления цифровых и символьных выражений, включенных в оператор.

Язык БЭЙСИК 02 разрешает вводить более одного оператора в строке программы. В этом случае операторы в одной и той же строке должны быть разделены двоеточием.

Строки, содержащие группу операторов, способствуют более эффективному использованию памяти и более быстрому выполнению программы.

В строку программы для визуального удобства могут быть введены пробелы, но при выполнении программы они не принимаются во внимание.

Например, строка

```
: I0 PRINTA%  ,B
```

Эквивалентна строке

```
: I0 PRINT A% , B
```

Все пробелы в строке программы выбрасываются при запоминании, кроме:

I) пробелов, которые вводятся в кавычках или апострофах ("Т ЕСТ", "Т Е К С Т");

2) пробелов в операторах REM и %

При выводе текста программы на дисплей или печать пробелы вставляются автоматически в текст для удобства чтения.

I.I.5. Максимальная длина строки программы.

Одна строка программы на дисплее может занимать несколько строк. Максимальная длина строки программы определяется максимальной длиной строки программы, которая может быть записана в сектор диска, и равна 253 байтам.

При вводе строки программы с клавиатуры после набора 253 символа, система не воспринимает нажатия клавиш, кроме CR/LF, для совершения нормального ввода. При нажатии клавиш команд ошибка не выдается и ввод строки программы не прекращается.

I.I.6. Выполнимые и невыполнимые операторы.

Кроме операторов, которые дают указания системе выполнить определенные задания во время их исполнения ("выполнимые операторы") в систему входят операторы, описывающие данные программы:

I) обеспечение системы информации (DIM, COM, DEFFN, DEFFNP, DATA, %);

2) оформление программы (REM).

Эти операторы называются "невыполнимыми". Они не дают системе команды действовать, когда встречаются во время выполнения программы. Невыполнимые операторы могут быть помещены в любом месте программы (при условии некоторых ограничений).

I.I.7. Команды.

Команды в системе обеспечивают управление основными функциями системы непосредственно с клавиатуры. Команды вводятся и немедленно выполняются и в память, как часть программы, не записываются. Команды дают возможность выполнить следующие функции:

I) останов программы во время выполнения (HALT/ STEP);

2) продолжение выполнения программы после останова (CONTINUE);

3) прекращение выполнения программы (RESET).

Все команды языка имеют на клавиатуре свои собственные клавиши и вводятся нажатием соответствующей клавиши. Индикация команд на дисплее отсутствует. Для окончания ввода с клавиатуры должна быть нажата клавиша CR/LF. После набора строки программы и нажатия клавиши CR/LF строка программы обрабатывается системой. При нажатии клавиши CP/LP непосредственно после ввода оператора без номера строки, система немедленно выполняет данный оператор.

I.I.8. Режим непосредственного счета.

В режиме непосредственного счета операторы выполняются как только осуществляется их ввод и не запоминаются в памяти как часть программы, предназначенная для дальнейшего выполнения. Все команды системы выполняются в режиме непосредственного счета и в программу включены быть не могут.

Несмотря на то, что согласно определению операторы являются программируемыми элементами, большая их часть может без ограничений выполняться в режиме непосредственного счета, если их ввести без номера строки. Чаще всего в режиме непосредственного счета выполняется оператор печати, который используется для того, чтобы показать результаты вычисления выражения.

В режиме непосредственного счета могут выполняться строки с несколькими операторами, например:

FOR I=I TO IO: PRINT LOG(I): NEXTI

Надо иметь ввиду, что выполнение оператора PRINT в режиме непосредственного счета не влияет на содержащиеся в памяти переменные, даже если в операторе есть ссылка на эти переменные. Однако другие операторы, выполняющиеся в режиме непосредственного счета, могут изменить содержащиеся в памяти переменные. Поэтому следует быть внимательными при выборе тех переменных, на которые есть ссылки в операторах, если изменение этих переменных может повлиять на выполнение программы, находящейся в текущий момент в памяти.

Некоторые операторы языка (DATA, DEFFN , DEFFN' , ONERROR, %)

при использовании их в режиме непосредственного счета не имеют смыслового значения и системой не воспринимаются. Операторы READ , GOSUB ,GOSUB', ON GOSUB могут использоваться в режиме непосредственного счета только при выполнении некоторых условий:

I) оператор READ может быть выполнен, если перед его вводом программа, находящаяся в памяти, была запущена по оператору RUN;

2) операторы GOSUB, GOSUB' , ONGOSUB могут быть выполнены только при отсутствии в подпрограмме, к которой производится обращение, оператора RETURN , если оператор RETURN есть, то при попытке осуществить возврат в строку непосредственного счета выдается сообщение об ошибке (ERROR 25).

Хотя строка, выполненная в режиме непосредственного счета, не записывается в зоне программы, она временно сохраняется в памяти в специально отведенной зоне и может быть вновь вызвана для редактирования непосредственно после ввода. Строка непосредственного счета остается з памяти до тех пор пока не выполнится одно из следующих условий:

I) вводится новая строка непосредственного счета (в этом случае новая строка замещает предыдущую в памяти и эта строка утрачивается);

2) по оператору CLEAR (в этом случае память очищается).

I.2. Организация памяти.

В настоящем подразделе описывается логическая структура внутренней памяти БЭЙСИК- системы. Целью настоящего подраздела является разъяснение понятий, касающихся хранения текста программы и переменных в памяти пользователя.

I.2.I. Этапы выполнения программы.

Ввод и выполнение программы на языке БЭЙСИК состоит с точки зрения системы из трех этапов: ввод, просмотр программы и выполнение.Во время каждого этапа в системе выполняется определенный ряд действий.

Этап ввода включает в себя ввод текста программы на языке БЭЙСИК в память и обычно осуществляется с клавиатуры или с диска. В большинстве случаев программа первоначально вводится с клавиатуры.

После нажатия клавиши CR/LF в процессе ввода производится разбиение строки на операторы и проверка синтаксиса (если ошибка обнаруживается, то выдается соответствующее сообщение). Если в строке нет ошибок и перед ней нет номера строки, строка немедленно выполняется. Если перед строкой есть номер, она записывается в память, (даже в том случае, если в ней есть синтаксические ошибки).

Ввод строк в соответствии с последовательностью номеров строк данной программы БЭЙСИК-система осуществляет следующим образом. Номера введенных строк программы хранятся в таблице строк. Сами строки программы хранятся в зоне программы. Таблица строк представляет собой упорядоченный список элементов, каждый из которых содержит номер строки (2 байта) и адрес расположения строки в зоне программы (2 байта). С помощью таблицы строк производится упорядочивание вводимых строк в порядке возрастания номеров.

Строки программы могут вводиться в любом порядке. Когда вводится новая строка со своим индивидуальным номером, оттранслированный текст строки переписывается (в процессе трансляции) в конец зоны программы, а в таблицу строк добавляется элемент, содержащий номер введенной строки и адрес расположения в зоне программы. Размещение нового элемента в таблице строк зависит от номера строки и от номеров уже введенных строк программы.

Когда вводится строка с номером, идентичным номеру строки уже находящейся в памяти, то первоначальная строка уничтожается следующим образом: текст новой строки переписывается в конец зоны программы, в таблице строк производится замена адреса расположения строки в зоне программы, а по первоначальному адресу в зоне программы записывается признак вычеркнутой строки.

Когда вводится номер строки, за которым следуют только CR/LF, сброс указанной строки производится следующим образом: по введенному номеру строки из таблицы строк выбирается адрес расположения строки в программе, в зоне программы по этому адресу записывается признак вычеркнутой строки, из таблицы строк выбрасывается элемент, соответствующий указанному номеру строки.

При вводе строки без номера, в процессе трансляции строка перестраивается в зону непосредственного счета, расположенную за зоной программы. При вводе следующей строки без номера, первоначальная строка уничтожается и ее место в зоне непосредственного счета занимает вновь введенная строка.

В зоне программы вводимые строки располагаются друг за другом по мере ввода. В процессе просмотра операторов производится преобразование текста вводимой строки. Каждый оператор строки в зоне программы состоит из кода оператора (один или два байта), длины оператора (один байт) и значения оттранслированного оператора. Каждая строка заканчивается байтом нуля (00). В процессе трансляции каждой строки пропускаются все пробелы (кроме пробелов в символьных константах и операторах REM и '). Идентификаторы переменных заменяются на порядковые номера, числовые константы

к символьные. Числовые данные могут участвовать в арифметических операциях (сложение, вычитание и т.п.) и математических функциях. Каждое числовое значение запоминается в соответствии с форматом БЭЙСИК-системы. Символьные значения не могут участвовать в стандартных арифметических операциях. Символьные значения записываются в память символ за символом и занимают объем памяти, определяемый их размерами. Числовые и символьные данные в программе могут использоваться как константы или как переменные. Числовые значения могут быть представлены как числа с фиксированной запятой или в экспоненциальной форме. Символьные значения могут быть представлены в виде последовательности символов, заключенных в кавычки, апострофы. Символьные значения, не имеющие графического обозначения могут быть представлены в шестнадцатиричном виде с помощью функции HEX. Например, HEX(000102).

I.2.2.2. Числовые и символьные переменные.

Наряду с числовыми и символьными константами в памяти могут храниться такие данные, как значения переменных. Переменные - это определенные блоки данных в памяти, значения которых могут быть изменены в процессе выполнения программы.

В БЭЙСИК-системе выделяется два класса переменных: числовые и символьные. Числовые переменные могут принимать только числовые значения, а символьные переменные - символьные значения. Класс числовых переменных делится на два типа: действительные переменные и целые переменные. Целые переменные отличаются от действительных переменных диапазоном допустимых значений (от - 7999 до + 7999) и внутренним форматом представления.

Каждая переменная должна быть описана индивидуальным именем переменной. В языке БЭЙСИК имена переменных состоят из одной буквы латинского алфавита (A - Z) или из одной буквы, за которой следует одна цифра (0 - 9). Имена целых переменных отличаются от имен действительных переменных наличием знака %, который следует непосредственно за именем переменной. Примеры правильных имен:

Действительные	Целые	Символьные
A	A%	A$
A1	A1%	A1$
B0	B0%	B0$
Z9	Z9%	Z9$

Имена действительных, целых и символьных переменных всегда описывают различные переменные, даже если имена (т.е. буквы или буквы и цифры) идентичны и не вносят двусмысленности в программу.

I.2.2.3. Скалярные переменные и массивы.

Внутри каждого типа переменных (действительных, целых и символьных) выделяются две различные группы переменных: скалярные переменные и массивы переменных. Действительная скалярная переменная всегда имеет длину 8 байтов, целая скалярная величина - 2 байта. Символьная скалярная переменная может содержать одну непрерывную строку символов. Эта переменная может иметь длину от I до 253 байтов. Если пользователь не определяет длину символьной скалярной переменной (оператором DIM или COM, то система определяет длину переменной I6 байтов).

Массив переменных - это упорядоченное множество скалярных переменных, обозначенных общим именем. Каждая из скалярных переменных в массиве называется элементом массива и может быть обозначена при помощи имени массива, за которым следует индекс или двойной индекс, которые определяют номер элемента массива. Например, пятый элемент в массиве N ( ) может быть определен как N(5).
Имя массива переменных образуется точно так же, как и имя скалярной переменной, причем для скалярных переменных и массивов могут использоваться одни и те же имена.

Любая ссылка на массив переменных должна состоять из имени массива, непосредственно за которым следуют круглые скобки. Если в круглые скобки заключено выражение или два выражения, то эти выражения интерпретируются как индексы конкретного элемента массива. В приведенном выше примере N(5) обозначает пятый элемент массива N( ). Если должна быть сделана ссылка на весь массив, а не на отдельный элемент массива, то за именем массива должны следовать пустые круглые скобки, например: N( ), N%( ) или A$( ). Массив может иметь то же название, что и скалярная переменная, но (за некоторым исключением) в ссылке на массив должен обязательно присутствовать указатель массива ( ), который показывал бы, что имеется ввиду массив, а не скалярная переменная.
Например:

N6 обозначает действительную скалярную переменную
N6% обозначает целую скалярную переменную
N6$ обозначает символьную скалярную переменную
N6( ) обозначает действительный массив
N6%( ) обозначает целый массив
N6$( ) обозначает символьный массив

I.2.2.4. Одномерные и двумерные массивы

Массивы переменных могут быть двух типов: одномерные и двумерные. Одномерный массив представляет собой список переменных, каждая из которых обозначена одним и тем же именем. Одномерные массивы также называют векторами. Двумерный массив представляет собой таблицу переменных, где все переменные обозначены одним и тем же именем. Двумерные массивы также называют матрицами.

Одномерный массив представляет собой список или столбец переменных (элементов), каждый из которых занимает свою строку в столбце. Например, массив N( ):

Строка I	N(I)
Строка 2	N(2)
Строка 3	N(3)
Строка 4	N(4)
Строка 5	N(5)

Массив N( ) состоит из 5 элементов, каждый элемент обозначается при помощи строки в столбце (например, элемент N(3) расположен в строке 3).

Двумерные массивы называют таблицами или матрицами. Каждый элемент двумерного массива обозначается двумя индексами. Двумерный массив может представлять собой таблицу, состоящую из двух или более столбцов. Например массив M ( ):

	Столбец I	Столбец 2
Строка I	M(I,I)	M(I,2)
Строка 2	M(2,I)	M(2,2)
Строка 3	M(3,I)	M(3,2)
Строка 4	M(4,I)	M(4,2)
Строка 5	M(5,I)	M(5,2)

Массив M ( ) состоит из двух столбцов по пять строк каждый, всего I0 элементов. В данном случае недостаточно обозначить каждый элемент номером строки, поскольку строка имеет столбец I и столбец 2. Для обозначения столбца требуется еще один индекс. Обычно при обращении к конкретному элементу в двумерном массиве сначала указывается строка, а затем столбец. Таким образом, M(3,2) обозначает элемент в третьей строке. столбца 2.

I.2.2.5. Максимальный размер массива.

Допустимый размер массива имеет определенные ограничения. Первое ограничение определяется наличием свободного объема в памяти:массив не может содержать больше байтов, чем это позволяет свободное место в памяти. Для массивов имеющийся объем памяти является единственным ограничением, поскольку максимально возможный объем памяти составляет 64К или 65535 байтов, то максимально допустимое число элементов в одномерном массиве - 65535. Это теоретический максимум, который по разным причинам практически никогда не достигается. Для двумерных массивов это ограничение можно сформулировать следующим образом: произведение числа строк на число столбцов не должно превышать 65535.

Массивы N( ) и M( ), показанные выше, являются действительными массивами, в которых каждый элемент имеет постоянную длину восемь байтов. Таким образом, N( ), который состоит из пяти элементов занимает в общем 5*8 = 40 байтов. Аналогичным образом M( ) с I0 элементами занимает I0* 8 = 80 байтов. Отсюда ясно, что ограничения для массивов должны учитывать не только число элементов, но и количество байтов, отводимых под каждый элемент. Например, одномерный действительный массив с 65535 элементами займет в общей сложности 65535* 8 = 524280 байтов, т.е. число, которое выходит далеко за пределы всего объема памяти,(фактически, каждая переменная также требует несколько управляющих байтов в памяти, что еще больше уменьшает их допустимое количество).Решающим фактором при определении размера массива всегда должно быть наличие свободного места в памяти, поскольку это условие в большинстве случаев представляет собой единственное практическое ограничение при выборе размера массива.

Длина элемента символьного массива может быть установлена пользователем от I до 253 байтов включительно. Символьный массив, состоящий из пяти элементов длиной в I байт будет занимать 5 байтов, в то время как массив из пяти элементов длиной 253 байта будет занимать 5 * 253 = I265 байтов (не считая управляющие байты).Определяющим фактором в ограничении размера массива также как и для числовых массивов является наличие свободного места в памяти.

I.2.2.6. Управляющие байты для переменных.

Для того, чтобы обозначить каждую переменную в памяти,система автоматически вводит несколько байтов управляющей информации,которые используются системой и не доступны пользователю. Эти управляющие

байты представляют собой постоянные " накладные расходы" на хранение каждой переменной программы. Количество управляющих байтов для символьных, целых, действительных переменных, а также массивов разное:

Тип переменной	Количество управляющих байтов
Целая скалярная	2
Действительная скалярная	4
Символьная скалярная стандартной длины	4
Символьная скалярная объявленной длины	12
Массив	18

Управляющие байты не повторяются для каждого элемента массива, они определяют весь массив. Таким образом, целый массив из 5 элементов имеет такие же " накладные расходы" (18 байтов), что и целый массив из 500 элементов. Для того, чтобы вычислить общее число байтов T, которое фактически необходимо для переменной в памяти, можно использовать следующую формулу:

$$T = N * L + P$$

где:

N - общее число элементов в массиве (N = 1 для скалярных переменных)

L - длина скалярной переменной или каждого элемента массива (целого - 2, действительного 8, символьного- от 1 до 253 байтов)

P - накладные расходы системы.

1.2.2.7. Определение переменных и массивов.

В общем случае существует три способа определения переменной:

1) неявное определение при использовании в операторе на языка БЭЙСИК;

2) явное определение оператором DIM или COM (скалярные переменные и массивы переменных);

3) явное или неявное определение матричными операторами (только массивы переменных).

Для скалярных переменных явного определения не требуется. Используемая скалярная переменная автоматически определяется, когда она впервые появляется в программе. Подобным образом могут быть определены как числовые, так и символьные переменные стандартной длины (16 байтов).

Массивы переменных должны в общем случае определяться явно.

Исключением из общего правила являются матричные операторы, которые обеспечивают неявное определение массива. Явное определение переменной может быть выполнено при помощи операторов DIM и COM. Например, оператор:

: 20 DIM N(5), M (5,2), A$ (10,10) 100

определяет три массива. N( ) - это одномерный числовой массив из пяти элементов, M ( ) - это двумерный массив из пяти строк и двух столбцов (10 элементов), A$( )-двумерный символьный массив из 10 строк и 10 стобцов (100 элементов), причем каждый элемент имеет длину 100 байтов.

Если символьной скалярной переменной нужно задать другую длину, а не 16 байтов, эта переменная должна быть определена явно. Например, оператор:

: 30 DIM F$ 252, A4$ 3

Определяет символьные скалярные переменные F$ с длиной 252 байта и A4$ с длиной 3 байта. Действительные и целые скалярные переменные также могут быть обозначены в операторе DIM для удобства документации, но их длина всегда остается постоянной, она составляет 8 байтов для действительных и 2 байта для целых переменных и не может быть изменена пользователем.

Оператор COM может быть использован аналогичным образом, как и оператор DIM, для явного определения переменных. Например, оператор:

: 20 COM N(5), M(5,2), A$ (10,10) 100

определяет три массива N( ), M ( ) и A$ ( ) точно так же как и оператор 20 выше. Однако, имеется существенная разница между DIM и COM; переменные, определенные в DIM, не сохраняют своё значение при повторном запуске программ и подгрузке программных сегментов, в то время как переменные, определенные в COM сохраняют свое значение и называются общими переменными.

I.2.2.8. Таблица переменных.

Зона памяти, где хранятся описания переменных, называется таблицей переменных. Таблица переменных располагается в памяти перед таблицей строк и расширяется в сторону уменьшения адресов. В таблице переменных отводится место для всех определенных во время трансляции переменных.

Во время процесса трансляции строится таблица переменных путем отведения места для описания переменной в той последовательности, в которой появляются переменные. Как только появляется очередная

переменная, производится просмотр имеющейся таблицы и если такой переменной в таблице нет, производится подсоединение описания новой переменной к таблице и формирование номера переменной. Таким образом, первая определенная переменная занимает первое положение в таблице и имеет номер 0, следующая определенная переменная занимает положение непосредственно за ней и имеет номер 1 и так далее. Рассмотрим, например, следующие два оператора:

```
: 10 COM N, M (2,5), L¤
: 20 DIM F¤(10,10)100, R (3), B¤25
```

В таблице эти переменные будут располагаться следующим образом:

```
B¤ R( ) F¤( ) ! L¤ M( ) N
              ! Граница общих переменных.
```

1.2.2.9. Общие и необщие переменные.

Таблица переменных содержит два типа переменных - общие и необщие. Все общие переменные размещаются в таблице сначала, а затем размещаются необщие переменные. Общие переменные отделяются от необщих границей, определяющей расположение первой необщей переменной. Все переменные, расположенные справа от границы общих переменных, являются общими переменными, а все те, которые расположены слева, называются необщими переменными. Например, в рассматриваемом случае переменные N, M ( ) и L¤ являются общими переменными, A P¤ ( ), F ( ) и B¤ необщими.

Такой метод различения общих и необщих переменных устанавливает строгие ограничения порядка определения этих переменных. В частности, все общие переменные должны быть определены до того, как будет определена любая из необщих переменных. Как только появляется первая необщая переменная, система автоматически фиксирует указатель границы общих переменных и все последующие переменные определяет как необщие. Если после этого делается попытка определить общую переменную, система формирует сообщение об ошибке.

Когда очередной сегмент программы загружается в память по оператору LOAD , все переменные, находящиеся в слова за границы общих переменных сбрасываются (получают начальное значение), а общие переменные, расположенные справа от границы сохраняют свои значения. Поэтому общие переменные так нужны для хранения общих данных, которые используются для нескольких сегментов программы.

Граница общих переменных используется и при выполнении оператора CLEAR. Когда выполняется оператор CLEAR без параметров, вся память,

в том числе и зона программы, таблица строк и таблица переменных сбрасываются. Оператор CLEAR P сбрасывает зону программы и таблицу строк, не затрагивая таблицу переменных. Оператор CLEAR V сбрасывает зону значений переменных, перестраивает заново таблицу переменных исключая при этом переменные, которых нет в программе, отводит зону значений переменных по таблице переменных и устанавливает всем переменным начальные значения, не затрагивая при этом зону программы и таблицу строк. И, наконец, оператор CLEAR N сбрасывает зону значений необщих переменных, перестраивает таблицу переменных, отводит зону значений необщих переменных по таблице переменных, устанавливает начальные значения необщих переменным, не затрагивая при этом зону программы, таблицу строк и зону значений общих переменных. Обратите внимание на то, что при использовании оператора CLEAR N в программе происходит уничтожение в таблице переменных только тех необщих переменных, которые не содержатся в тексте программы в момент выполнения оператора CLEAR N , те же необщие переменные, которые содержатся в программе, лишь принимают начальные значения.

I.2.2.IO. Переопределение переменных.

Для того, чтобы переместить переменную из общих в необщие и наоборот, необходимо передвинуть границу общих переменных вправо, или влево по таблице переменных. Например, таблица переменных содержит следующие переменные:

B¤ R( ) F¤( ) I L¤ M( ) N( )

Предположим граница смещена на одну переменную вправо

B¤ R( ) F¤( ) L¤ I M( ) N( )

Последняя общая переменная L¤ теперь стала необщей в результате перемещения границы. Часто в больших системах необходимо, чтобы переменные, которые были необходимы в качестве общих данных в первоначальных сегментах, позднее не использовались как общие. В таких случаях очень удобно иметь способ нового определения ненужных общих переменных в качестве необщих переменных с тем, чтобы их можно было стереть при загрузке последующего сегмента.

Бывает необходимость наоборот превратить одну и более необщих переменных в общие. В любом случае процедура заключается в простом перемещении границы общих переменных вправо или влево по таблице

переменных.

В языке БЭЙСИК предусмотрен специальный оператор для изменения границы общих переменных: COM CLEAR . COM CLEAR имеет три формы: COM CLEAR с общей переменной, COM CLEAR с необщей переменной и COM CLEAR без указания параметров. Оператор COM CLEAR с общей переменной сдвигает границу общих переменных вправо таким образом, что указанная переменная и все общие переменные, расположенные левее ее превращаются в необщие переменные. Например, таблица содержит следующие переменные:

B¤ R( ) F¤( ) ↑L¤ M( ) N( )

В результате выполнения оператора 100 COM CLEAR M ( ) таблица будет иметь вид:

B¤ R( ) F¤( ) L¤ M( ) ↑ N( )

Обратите внимание на то, что переменные M ( ) и L¤ стали необщими переменными. Подгрузка следующего сегмента уничтожит эти переменные, если они будут не нужны.

Оператор COM CLEAR с необщей переменной перемещает границу общих переменных влево таким образом, что все переменные, расположенные справа от нее (переменные определенные до нее в программе) становятся общими. Сама обозначенная переменная и все переменные, определенные после этого в программе, остаются необщими.
Предложим, что после выполнения рассмотренной строки 100 выполняется строка 200 COM CLEAR B¤ .
В результате таблица переменных будет иметь вид:

B¤ ↑ R( ) F¤( ) L¤ M( ) N( )

Обратите внимание на то, что переменные M ( ),L¤, F¤( ) и R( ) стали общими переменными. Эти переменные не будут уничтожаться при подгрузке последующего сегмента.

Оператор COM CLEAR без параметров просто перемещает границу общих переменных в начало таблицы, в результате чего все общие переменные таблицы становятся необщими. При последующей подгрузке сегмента память очищается от всех переменных.

I.2.2.II. Внутренние магазины.

БЭЙСИК-система в своем составе имеет два внутренних магазина в памяти для хранения определенной управляющей информации: магазин циклов и подпрограмм RENEXT и магазин операндов. В процессе выполнения программы размеры магазинов увеличиваются или уменьшаются. Эти магазины располагаются в центральной части памяти и при увеличении стремятся навстречу друг другу. Магазин RENEXT располагается за зоной строки непосредственного счета. Магазин операндов располагается перед таблицей переменных (непосредственно перед динамической зоной заданий на ввод вывод, которая может отсутствовать).

Магазин RENEXT предназначен для хранения адресов возврата и другой управляющей информации для подпрограмм и циклов. Когда осуществляется вызов подпрограммы (GOSUB, GOSUB', ON GOSUB и т.д.) система должна иметь возможность запомнить расположение оператора, который следует за GOSUB с тем, чтобы затем был осуществлен возврат после того, как выполнение подпрограммы закончится. Поэтому, прежде чем обращаться к подпрограмме, система записывает адрес следующего оператора программы в магазин RENEXT. Подобным же образом, когда выполняется оператор FOR... TO, адрес оператора, который следует за FOR ... TO и другая необходимая информация записывается в магазин RENEXT. Когда выполняется соответствующий оператор NEXT, система обращается к соответствующему оператору цикла с тем, чтобы начать следующую итерацию цикла.

Информация записывается в магазин RENEXT последовательно. Например, рассмотрим программу:

```
вызов          : 100 GOSUB  500
подпрограммы   : 110
                 .
                 .
                 .
подпрограмма   : 500 FOR  I=1  TO 10
               : 510 FOR  J=1  TO 20
                 .
                 .
                 .
               : 600 NEXT J
               : 610 NEXT I
                 .
                 .
                 .
               : 700  RETURN
```

Данная программа содержит три оператора требующих использование магазина: оператор GOSUB в строке 100 и два оператора FOR... TO в строках 500 и 510. Расположение информации в магазине после выполнения строки 510 следующее:

информация о возврате
оператора GOSUB
в строке 100

информация цикла
оператора FOR
в строке 500

информация цикла
оператора FOR
в строке 510

Обычно информация стирается в магазине при следующих условиях:

1) выполнение оператора RETURN ;
2) выполнение оператора NEXT ;
3) выполнение оператора RETURN CLEAR.

Оператор RETURN обеспечивает переход к оператору, который следует за последним выполненным оператором GOSUB и т.п. и автоматически стирает информацию о возврате из магазина. Оператор NEXT при условии окончания цикла стирает информацию цикла из магазина. Информация в магазин добавляется в конец, стирание информации из магазина производится с конца. Поэтому когда из магазина стирается элемент (информация возврата или цикла), то вся информация расположенная за элементом (т.е. записанная в магазин после этого элемента) также стирается. Обычно последний добавленный в магазин элемент является первым элементом, который стирается. В рассмотренной программе самый внутренний цикл (начиная со строки 510 и кончая строкой 640) будет естественно закончен первым и его информация будет стираться из магазина первой. Затем будет закончен следующий цикл (строки 500 - 650) и его информация будет стираться из магазина. После этого будет выполнен оператор RETURN, который сотрет информацию возврата подпрограммы.

Однако нужно обратить внимание на то, что могут возникать осложнения, если не придерживаться нормального порядка выполнения програм-

мы. Например, предположим, что оператор RETURN в строке 700 перенесен в строку 530. Возврат из подпрограммы будет осуществлен до нормального завершения обоих циклов. В этом случае информация циклов будет стерта из магазина при стирании из магазина информации возврата подпрограммы. Любая попытка вновь выполнить оператор NEXT любого из циклов приведет теперь к ошибке.

Магазин операндов предназначен для хранения операндов, адресов операндов и результатов вычисления арифметических выражений. Информация в магазин операндов добавляется в конец. При выполнении операции из магазина выбирается последний элемент (или элементы). Результат операции, как правило, заносится в магазин и становится последним элементом. Информация в магазин операндов заносится при расшифровке операторов, содержащих арифметические выражения. К концу выполнения оператора магазин операндов всегда пуст.

Если магазины непрерывно пополняются информацией без соответствующего стирания, это в конечном итоге приведет к переполнению памяти и будет выдано сообщение об ошибке. При нормальном выполнении программы такая ситуация возникает, если пользователь пытается некорректно использовать циклы и подпрограммы. Например, неоднократный выход из подпрограмм без выполнения операторов RETURN или выходы из циклов до нормального окончания цикла приводит к накоплению информации в магазине RENEXT, когда соответствующее стирание не происходит. Если любую из этих операций повторять часто, в результате наступит переполнение памяти, при этом система автоматически сбрасывает магазин RENEXT.

I.2.2.I2. Предупреждение переполнения памяти при помощи RETURN CLEAR.

Обычный метод программирования предполагает использования клавиш специальных функций клавиатуры для запуска выполнения подпрограммы в памяти. Если эти подпрограммы завершаются обычными операторами RETURN, возврат останавливает выполнение программы и передает управление пользователю, что может быть неудобно. В другом случае, если операторы RETURN не используются для завершения подпрограмм, может возникнуть переполнение памяти. Эту проблему можно решить при помощи оператора RETURN CLEAR.

Оператор RETURN CLEAR является оператором, который стирает информацию о подпрограмме в магазине RENEXT не передавая управление оператору, который следует за последним оператором GOSUB и т.п. (или в случае подпрограмм, вызванных при помощи клавиш специальных функций - на клавиатуру). Когда подпрограмма завершается

оператором RETUR CLEAR , управляющая информация для данной подпрограммы стирается в магазине и выполнение программы продолжается со следующего оператора. Если используется параметр ALL, происходит сброс всего магазина RENEXT.

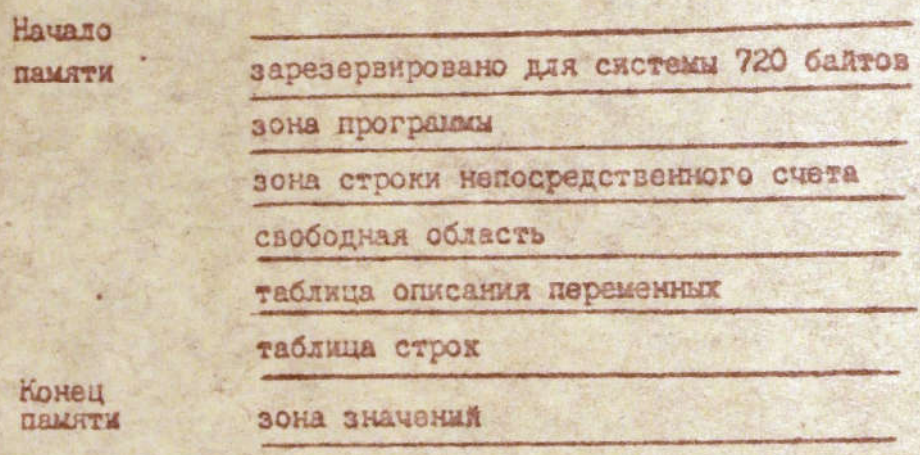
I.2.2.I3. Автоматический сброс магазинов .

Магазин RENEXT и магазин операндов автоматически сбрасываются системой в следующих случаях:

I) запуск выполнения программы по оператору RUN ;
2) выполнение оператора CLEAR;
3) выполнение оператора LOAD любого вида;
4) нажатие клавиши RESET;
5) возникновение переполнения памяти.

I.2.2.I4. Организация памяти и понятие свободного места.

Величина имеющегося свободного места в памяти определяет возможность пользователя добавить новые строки программы и переменные, и успешно выполнить программу после того, как она была введена. Организацию памяти можно представить следующим образом:



Зона программы расширяется в сторону увеличения адресов по мере выполнения программы новыми строками. Зона строки непосредственного счета и магазин RENEXT перемещаются по мере увеличения или уменьшения зоны программы.

Зона значений расширяется в сторону уменьшения адресов по мере увеличения числа переменных. До тех пор пока идет ввод программы зона значений может отсутствовать. Перед зоной значений располагается таблица строк, которая расширяется в сторону уменьшения адресов при вводе каждой программной строки. Перед таблицей строк располагается таблица описания переменных.

которая расширяется в сторону уменьшения адресов при появлении очередной переменной.

В свободной области могут формироваться в процессе счета по программе:

1) магазин RENEXT за зоной непосредственного счета;

2) зона заданий на ввод/вывод перед таблицей описания переменных;

3) магазин операндов перед зоной заданий на ввод-вывод.

Все эти зоны динамические, они возникают в процессе счета по программе и тогда свободная область уменьшается. Обычно к завершению программы все динамические зоны уничтожаются.

Кроме того, в этой свободной области в режиме ввода располагается рабочий буфер ввода с клавиатуры, который уничтожается при счете по программе.

Таким образом, понятие свободного места в памяти определяется размером свободной области, ограниченной в общем случае сверху - зоной программы, снизу - таблицей описания переменных этой программы.

I.2.2.I5. Определение свободного места при помощи END.

Для того, чтобы обеспечить пользователя сведениями о наличии свободного места в памяти в любой момент до начала выполнения программы или в процессе выполнения программы, система располагает оператором END , с помощью которого сообщает пользователю объем свободной области в байтах.

При использовании оператора END в режиме ввода следует учесть, что зона значений в этот момент может отсутствовать. Оператор END используемый в программе, автоматически прекращает выполнение программы и дает величину текущего объема памяти, не занятой текстом программы, переменными и зоной значений. Таким образом оператор END в программе будет показывать меньший объем, чем сразу после ввода одной и той же программы.

Важно, чтобы пользователь понимал, что может возникнуть такое положение, когда есть достаточно свободного места для ввода исходной программы, но нет достаточного места для выполнения программы, так как при вводе может отсутствовать зона значений. Для того, чтобы определить фактический объем свободной памяти, его следует проверять во время выполнения программы, когда отведена зона значений и уже может быть сформирован магазин RENEXT.

I.2.2.I6. Переполнение памяти .

Переполнение памяти наступает тогда, когда все имеющееся свободное место в памяти исчерпано. Такое положение может сложиться тогда, когда между границей зоны строки непосредственного счета (или магазина RENEXT) и границей таблицы строк (или магазина операндов) имеется менее 500 байтов.

В этом случае сравниваются размер зоны программы и размер таблиц переменных, строк и зоны значений. Если размер зоны программы больше - выдается ошибка ERR 01, в противном случае - ERR 02. Ошибки переполнения памяти характерны для всех этапов работы. На этапе ввода может не хватать памяти при отведении рабочего буфера клавиатуры, зоны строки непосредственного счета, при записи номера строки в таблицу строк и описаний переменных в таблицу переменных. На этапе просмотра программы может не хватать места на построение второй временной таблицы переменных или отведения зоны значений для переменных программы. На этапе счета по программе может не хватать места на запись очередной информации в магазин RENEXT или магазин операндов, на формирование задания на ввод-вывод. В общем случае эти ошибки не разрушают существующий текст программы, но сбрасывают значения всех переменных и внутренние магазины.

I.3. Возможности редактирования и отладки.

Для того, чтобы ввести и/или отредактировать строку текста, пользователь имеет возможность ввести информацию нажатием клавиш клавиатуры. Существуют два основных режима работы, когда управление передается на клавиатуру:

I) режим ввода текста;

2) режим редактирования.

I.3.I. Режим ввода текста.

В режиме ввода текста система готова к принятию строк программы и строк непосредственного счета, когда в начале строки набирается двоеточие (:).

Кроме того, система приступает к режиму ввода текста при выполнении оператора INPUT. В этом случае на экране появляется вопросительный знак (?), который указывает на необходимость ввода информации пользователем.

Во всех этих случаях пользователь может ввести с клавиатуры любой символ.

В режиме ввода текста нажатием клавиш вводятся символы, которые появляются на экране и одновременно помещаются во временную зону хранения, называемую буфером ввода. Это особая зона, которая используется для записи строки текста, по мере ее ввода. После того, как строчка набрана, по нажатии клавиши CR/LF она помещается в память машины. Если введенная строка является строкой программы, то в процессе трансляции она переписывается в зону программы, но не выполняется. Если это была строка непосредственного счета, то в процессе трансляции она переписывается в зону непосредственного счета, расположенную сразу за зоной программы, и немедленно выполняется.

Если строка состоит из одного или более значения данных, введенных в ответ на запрос INPUT, то каждое значение, в процессе трансляции, переписывается в свободную зону, расположенную сразу за зоной непосредственного счета и присваивается очередной переменной из списка переменных INPUT.

В режиме ввода текста системета обеспечивает возможность для редактирования строки текста, которая все еще находится в буфере ввода (т.е. до того, как нажата клавиша CR/LF). Как только клавиша CR/LF нажата, строка переписывается в память и должна быть повторно введена и отредактирована в режиме редактирования.

В режиме ввода текста будут рассмотрены следующие возможности редактирования:

1) стирание символов по клавише BACKSPACE;
2) стирание строки по клавише LINE ERASE;
3) стирание строки программы;
4) замена строки программы.

I.3.I.I. Стирание символов клавишей BACKSPACE.

Символ из вводимой строки текста может быть стерт при нажатии клавиши BACKSPACE. Каждое нажатие клавиши BACKSPACE смещает курсор на одну позицию влево, стирает символ с этой позиции и из буфера ввода. Например, предположим, что пользователь сделал ошибку во время ввода следующей строки:

```
: I00 PRINT  TAB(20); "ERROR) _
```

В этой строка последний символ ")" введен ошибочно. Одно нажатие на клавишу BACKSPACE выполнит перевод курсора на одну позицию влево и сотрет символ ")", строка будет иметь вид:

```
: I00 PRINT  TAB (20); "ERROR _
```

Теперь пользователь может ввести правильный символ (двойные кавычки) и в результате получаем:

```
: I00 PRINT  TAB (20); "ERROR"_
```

I.3.I.2. Стирание строки клавишей LINE ERASE.

Клавиша LINE EPASE может быть использована для стирания вводимой строки.

Например, предположим, что следующий текст был введен ошибочно:

```
: 20 PRINT TAB(20), "ERROR"_
```

При нажатии на клавишу LINE ERASE эта строка будет стерта с экрана и из буфера ввода

```
:_
```

I.3.I.3. Стирание строки программы в памяти.

Хранящаяся в памяти строка программы может быть стерта путем ввода номера строки, непосредственно за которым нажимается клавиша CR/LF. Например, ввод

```
: 20_
: _
```

стирает строку 20 из памяти.

I.3.I.4. Замена строки программы в памяти.

Строка программы в памяти может быть заменена на новую строку путем присвоения номера исходной строки новой строке. Новая строка в памяти автоматически замещает старую строку. Например, в памяти находятся следующие строки:

```
: I0 INPUT  A,B
: 20 PRINT SQR (A^2+B^2)
:_
```

Введя новую строку 20, мы заменяем ею существующую 20 строку:

```
: 20 PRINT  A,B
:_
```

Теперь в памяти машины имеем:

```
: I0 INPUT   A,B
: 20 PRINT   A,B
: _
```

I.3.2. Режим редактирования.

В режиме редактирования перед пользователем раскрываются широкие возможности системы с точки зрения редактирования. Система переходит в режим редактирования:

1) при нажатии клавиши EDIT;
2) при выполнении оператора LINPUT.

Когда система переходит в режим редактирования, то символ двоеточия, стоящий в начале строки, заменяется символом звездочка (*), а если была строка с вопросительным знаком, то символ звездочка (*) появляется вслед за вопросительным знаком. Нажатие клавиши CR/LF прекращает режим редактирования и одновременно помещает отредактированную строку в память. В режиме редактирования допустимы следующие операции:

1) редактирование строк программы, строк непосредственного счета и значений данных во время ввода;
2) вызов строки программы из памяти по клавише RECALL ;
3) вызов строки непосредственного счета по клавише RECALL ;
4) повторный вызов введенных данных по клавише RECALL;
5) использование клавиш перемещения курсора;
6) использование клавиш BACKSPACE и LINE ERASE в режиме редактирования;
7) стирание символов в строке текста клавишами DELETE и ERASE;
8) вставка символов в строку текста при помощи клавиши INSERT;
9) изменение номера строки программы.

I.3.2.I. Редактирование строк программы, строк непосредственного счета и значений данных во время ввода .

Если в процессе ввода строки программы строки непосредственного счета или строки со значениями данных допущена ошибка, то пользователь может переключиться на режим редактирования до того, как будет нажата клавиша CR/LF. Для этого требуется нажать клавишу EDIT и сделать необходимые изменения (смотрите следующие пункты, посвященные перемещению курсора, стиранию символов и вводу символов). После того, как все необходимые исправления внесены, можно продолжить ввод в режиме редактирования. По нажатии клавиши CR/LF ввод и редактирование строки заканчивается.

I.3.2.2. Вызов строки программы по клавише RECALL .

Если требуется отредактировать строку программы, ранее записанную в памяти машины, то необходимо вновь вызвать данную строку в буфер ввода. Это делается путем ввода номера нужной строки, затем нажимаются клавиши EDIT и RECALL . По клавише RECALL обозначенная строка из зоны программы переводится в буфер ввода.

Следует отметить, что все изменения вносятся в копию строки программы, вызванную в буфер ввода, а не в саму строку программы, записанную в памяти, пока не будет нажата клавиша CR/LF. По нажатии клавиши CR/LF отредактированная строка замещает исходную строку в памяти.

I.3.2.3. Вызов строки непосредственного счета по клавише RECALL.

В отличие от строки программы, строка непосредственного счета хранится в памяти машины до ввода очередной (программной, непосредственного счета или пустой) строки. Как только строка непосредственного счета введена, по нажатии клавиши CR/LF происходит ее выполнение. Для каждой последующей строки непосредственного счета используется та же зона непосредственного счета, что и для предыдущей, поэтому в зоне непосредственного счета может находиться только выполненная строка непосредственного счета.

Последняя выполненная строка непосредственного счета может быть повторно вызвана в буфер ввода для редактирования путем нажатия клавиши EDIT, затем RECALL без обозначения номера строки.

По клавише RECALL строка непосредственного счета из зоны непосредственного счета переводится в буфер ввода. Затем строка редактируется и по нажатии клавиши CR/LF выполняется вновь. Если вслед за выполнением строки непосредственного счета в буфер ввода вводится один или более символов, то строка непосредственного счета не может быть вызвана повторно до тех пор, пока введенные символы не будут убраны из буфера ввода нажатием клавиши LINE ERASE до нажатия клавиши CR/LF.

Как только нажимается клавиша CR/LF, содержимое буфера ввода в процессе трансляции переписывается в зону непосредственного счета и предыдущая строка исчезает. Если на экране была изображена пустая строка, когда была нажата клавиша CR/LF, то зона строки непосредственного счета уничтожается.

I.3.2.4. Клавиши перемещения курсора.

Шесть клавиш перемещения курсора используются для установки курсора в желаемое положение при редактировании текста.

Клавиши перемещения курсора имеют следующие назначения:

-> перемещает курсор на одну позицию вправо,
<- перемещает курсор на одну позицию влево,
----> перемещает курсор на пять позиций вправо,
<---- перемещает курсор на пять позиций влево,
^ перемещает курсор на строку экрана вверх,
V перемещает курсор на строку экрана вниз.

С помощью клавиш ->, ----> и V производится перемещение курсора вправо, с помощью клавиш <-, <---- и ^ производится перемещение курсора влево по всей длине редактируемой строки.

I.3.2.5. Клавиши BACKSPACE и LINE ERASE в режиме редактирования.

В режиме редактирования по клавише BACKSPACE производится только перемещение курсора влево на одну позицию, без стирания символа, а при нажатии клавиши LINE EPASE, независимо от местоположения курсора, производится стирание всей редактируемой строки из буфера ввода и символ звездочка, стоящий в начале редактируемой строки, заменяется на символ двоеточие, т.е. система переводится в режим ввода.

I.3.2.6. Стирание символов в строке клавишами DELETE и ERASE.

В режиме редактирования символы в строке текста могут быть стерты путем нажатия клавиш DELETE и ERASE. Каждый раз при нажатии клавиши DELETE стирается символ, находящийся над курсором и оставшаяся часть информации перемещается на одну позицию влево. В последнее место символа строки вставляется пробел.

Например, есть ошибочная строка:

*A = 2++B_

Если установить курсор под символом "+" и нажать клавишу DELETE один раз, то получим,

*A= 2+B

По нажатии клавиши ERASE производится стирание символов, находящихся между курсором и концом редактируемой строки. Например, имеем строку:

*R = 5+C+F/2+B_

Если установить курсор под символом "/" и нажать на клавишу ERASE, то получим:

*R=5+C+F_

I.3.2.7. Вставка символов в строку клавишей INSERT.

В режиме редактирования символы могут быть вставлены в строку текста при помощи клавиши INSERT. При каждом нажатии клавиши INSEPT происходит раздвижка текста, т.е. информация, расположенная от курсора до конца текста сдвигается на одну позицию вправо, а на месте курсора появляется пробел. Это происходит в том случае, если редактируемая строка меньше 253 символов.

Если редактируемая строка занимает 253 символа, то по нажатии клавиши INSERT раздвижки не происходит. Чтобы отредактировать такую строку, нужно сначала удалить символ или несколько символов,

с помощью клавиши DELETE, а затем производить вставку символов с помощью клавиши INSERT.

Для ввода каждого символа в строку должен быть вставлен один пробел. Когда вставлено достаточное количество пробелов для размещения новых символов, эти символы вводятся нажатием клавиш. Например, имеется строка:

```
*20  PRINT_" ошибка"
```

Нажатие клавиши INSERT 8 раз дает:

```
*20PRINT_        "ошибка"
```

теперь, путем ввода нужной информации, получаем:

```
*20  PRINTTAB(25);_ "ошибка"
```

исправленная строка может быть записана в память по нажатии клавиши CR/LF.

I.3.2.8. Изменение номера строки программы.

В режиме редактирования можно изменить номер строки программы путем повторного вызова строки, переместив курсор в начало строки и изменив номер строки. По нажатии клавиши CR/LF эта строка записывается в зону программы.

Следует отметить, что отредактированная строка запоминается как новая строка (поскольку она имеет новый номер). Исходная строка со старым номером не стирается автоматически. Например, если повторно вызывается 20 строка, которая получает новый номер 30 и заносится в зону программы, то обе строки 20 и 30 (идентичные по содержанию, но имеющие разные номера) будут находиться в памяти машины. Исходная строка 20 может быть уничтожена путем стирания ее. Для этого нужно вызвать строку 20 и нажать клавишу CR/LF или заменить ее новой строкой 20.

I.3.2.9. Соединение строк программы.

Две или более строки программы в памяти могут быть соединены в одну строку программы в процессе редактирования. Эта процедура заключается в следующем:

Вызов первой строки программы, предназначенной для соединения, в буфер ввода осуществляется вводом номера строки и нажатием клавиш EDIT и RECALL.

В конце первой строки программы следует ввести двоеточие (:), затем номер строки программы, которая должна быть соединена со строкой, показанной в текущий момент на экране.

Нажать клавишу RECALL . Обозначенная строка вызывается из памяти и соединяется со строкой, которая в текущий момент находится в буфере ввода. Обратите внимание на то, что до этого момента ничего в памяти не менялось. Эта процедура может быть повторена для соединения нескольких строк в одну строку.

Нажатием клавиши CR/LF занесите эту новую строку в память. Новая строка замещает в памяти только первую вызванную строку (если она имеет номер первой вызванной строки). Последующие строки программы, вызванные и соединенные с исходной строкой, не стираются в памяти, когда новая строка заносится в память. Каждая строка стирается обычным способом при вводе номера строки и нажатием клавиши CR/LF.

Предположим, например, что в памяти находятся следующие две строки:

```
:10 INPUT  A,B
:20 PRINT   A+D^5
:_
```

Строка 20 может быть соединена со строкой 10, если вызвать строку 10 и в конце ввести двоеточие и 20. В результате получим:

```
*10  INPUT   A,B:20_
```

Если в этот момент нажать клавишу RECALL , то произойдет соединение строки 20 со строкой 10.

```
*10  INPUT  A,B:PRINT  A+D^5_
```

При нажатии клавиши CR/LF новый расширенный вариант строки 10 замещает исходную строку 10 в памяти. Однако, строка 20 остается в памяти нетронутой. Этот факт становится очевидным при листинге.

```
: LIST
: 10 INPUT  A,B:PRINT  A+D^5
: 20  PRINT  A+D^5
: _
```

Строка 20 должна быть стерта обычным способом.

I.3.3. Особенности отладки программ.

Процесс поиска и устранения ошибок в программе называется отладкой программы. Система предусматривает целый ряд средств отладки программы:

1) оператор STOP и команда CONTINUE;
2) команда HALT/STEP;
3) оператор TRACE;
4) оператор LIST;
5) оператор RENUMBER.

I.3.3.I. Останов и возобновление программы.

При отладке программы бывает целесообразно остановить выполнение программы на определенном месте для того, чтобы дать возможность пользователю изучить и изменить значения критических переменных. Одним из таких способов является оператор STOP. Выполнение оператора STOP приостанавливает выполнение программы, вызывает печать слова STOP, за которым следует произвольное сообщение, определенное в операторе, и/или номер строки оператора STOP. При этом пользователь получает возможность вывести текущие значения переменных при помощи оператора PRINT в режиме непосредственного счета, изменить их или выполнить другие нужные действия в режиме непосредственного счета. Количество операторов STOP в программе неограничено. Оператор STOP дает возможность пользователю контролировать выполнение программы в любых местах программы.

Выполнение программы может быть возобновлено после оператора STOP при нажатии клавиши CONTINUE. Команда CONTINUE возвращает систему к продолжению выполнения программы с оператора, следующего за оператором STOP, при условии, что пользователь не внес каких-либо изменений в текст программы.

I.3.3.2. Останов и пошаговое выполнение программы.

По нажатии клавиши HALT/STEP происходит остановка выполнения программы и пошаговое выполнение операторов программы с индикацией их на экране. Нажатие клавиши HALT/STEP останавливает выполнение программы после завершения исполняемого в данный момент оператора. Затем пользователь имеет возможность выполнить необходимые операции в режиме непосредственного счета. Каждое следующее нажатие клавиши HALT/STEP вызывает выполнение следующего оператора и индикацию оставшихся операторов в строке, затем снова производится останов. Часто целесообразно исполнять программу по шагам, наблюдая последовательность выполнения операторов в программе.

I.3.3.3. Трассировка.

Если программа выполняется в режиме трассировки (слежения), то пользователь имеет возможность просмотреть на экране присвоение значений переменным и переходы в программе. В режиме слежения система каждый раз автоматически при присвоении выводит имя и значение переменной и каждый раз при переходе в программе выводит сообщение с номером строки перехода, на которую передается управле-

ние. Режим слежения устанавливается оператором TRACE и снимается после выполнения оператора TRACE OFF.

Пользователь имеет возможность получить более полную картину того, что происходит в программе при пошаговом режиме выполнения программы по HALT/STEP, выполняемой в режиме слежения. Печатная копия результатов слежения может быть получена при назначении печатающего устройства, в качестве устройства консольного вывода до выполнения оператора TRACE.

Если TRACE имеет вывод на экран, то этот процесс может быть замедлен до скорости, удобной для чтения, при помощи организации пауз оператором SELECT P.

I.3.3.4. Вывод текста программы.

Оператор LIST имеет множество форм, которые вооружают пользователя целым набором возможностей, связанных с листингом и выдаче. перекрестных обращений в программе.

LIST - обеспечивает вывод всей или избранной части текста программы, находящейся в памяти машины.

LIST# - обеспечивает вывод перекрестного обращения к номерам строк программы.

LIST V - обеспечивает вывод имен переменных, определенных в программе, с указанием номеров строк, где они используются.

LIST' - обеспечивает вывод перекрестного обращения к помеченным подпрограммам (DEFFN'), которые были определены в программе и на которые в ней есть ссылки.

LIST % - обеспечивает вывод содержимого таблицы файлов.

LIST * - обеспечивает поиск по всей или избранной части программы указанного контекста и вывод содержимого строк с найденным контекстом.

I.3.3.5. Изменение номеров строк в программе.

Если для вставки новых строк необходимо освободить место, то нумерация строк программы может быть изменена при помощи оператора RENUMBER. Данный оператор дает возможность пользователю обеспечить вставку между последовательно идущими номерами строк программы. Для этого между последовательно идущими номерами строк следует оставить соответствующее количество номеров для вставки необходимого числа строк. При выполнении оператора RENUMBER автоматически меняются все обращения к номерам строк в программе (в операторах GOTO, GOSUB, IF THEN и т.д.) в соответствии с новой нумерацией.

I.4. Операции с числами.

I.4.1. Числовые значения.

Числовое значение может быть записано в памяти в виде константы или в виде значения числовой переменной. Оно может быть введено пользователем или получено как результат вычисления числовых выражений. Допустимый диапазон числовых значений для данной системы от 1E-99 до 9.99999999E+99.

Числовые значения, введенные с клавиатуры или с других носителей, могут быть представлены в формате числа без показателя или числа с показателем. Существуют следующие правила ввода числовых значений:

Формат числа без показателя - числовое значение, введенное в формате числа, может содержать максимум 13 цифр, знак и десятичную точку. Пробелы, вставленные в значение, игнорируются. Знак значения должен предшествовать цифре. Значение без знака считается положительным. Если десятичная точка не вводится, дробная часть числа считается равной нулю.

Примеры:

12.004
+17400
-.00275

Формат числа с показателем - числовое значение, введенное в формат числа с показателем, может содержать максимум 13 цифр, знак, десятичную точку и показатель десятичной степени, состоящий из буквы E и одной или двух цифр со знаком или без знака.

Значение равно введенному значению до показателя, возведенному в степень 10. Число должно соответствовать правилам описанным для формата числа без показателя. Показатель должен быть целым числом и содержать максимум две цифры и знак, перед которым ставится буква "E".

Если указан знак, то он должен находиться перед цифрами, обозначающими показатель. Показатель без знака рассматривается как положительное число.

Примеры:

5.65E5
-125012E+10
+24.0068-07
-1.028E-08

Если вводится числовая величина, состоящая более чем из 13 цифр, система выдает сообщение об ошибке. Впереди стоящие нули, до цифр

в целой части, в числовой величине игнорируются.

I.4.2. Числовые константы.

Числовая константа должна соответствовать форматам, описанным в предыдущем пункте. Константа является числовым значением, допустимым для операторов языка БЭЙСИК. Значение константы из изменяется во время выполнения программы.

Примеры:

```
: I0  N =I.256
: 30 PRINT  I65*N
: 50 K=5.700IE-03
```

I.4.3. Числовые переменные.

Числовые переменные используются для записи числовых данных в памяти. В отличие от констант, значения которых постоянны, переменным могут присваиваться новые значения во время выполнения операторов.

Поскольку числовые значения заносятся в память в виде специального формата числа с плавающей запятой, то для них используются специальные переменные, которые отличаются от переменных, используемых для хранения символьных данных. Числовые переменные бывают двух типов: действительные переменные и целые переменные. Целые переменные отличаются от действительных переменных диапазоном допустимых значений (от -7999 до + 7999) и внутренним форматом представления.

Внутри каждого типа переменных (действительных, целых и символьных) выделяется две различные группы переменных: скалярные переменные и массивы переменных. Скалярные числовые переменные используются для занесения в память одного числового значения и обозначаются буквой (A-Z) или буквой, за которой следует цифра (0-9). Существует всего 286 допустимых имен скалярных переменных (каждого типа). Имена целых переменных отличаются от имен действительных переменных наличием знака %, который следует непосредственно за именем переменной. Примеры правильных имен:

A, N , BI, F9, A% , N6%, BI%, F9%

Массив числовых переменных представляет собой группу элементов массива, обозначенных одним именем массива. Каждому элементу массива может быть присвоено одно числовое значение. Таким образом, массив может быть использован для хранения и обработки множества числовых значений. Для числовых массивов переменных используются те же имена, что и для скалярных числовых переменных.

Массив переменных может быть одномерным и двумерным. Конкретный элемент в массиве обозначается путем указания индекса (индексов), заключенного в круглые скобки, который ставится непосредственно за именем массива.

Примеры:

A(3), N(I,2), F7(6), BI(9,9)

Скалярные числовые переменные и массивы переменных могут иметь одно и то же имя, поскольку они являются независимыми переменными, но одномерный массив переменных и двумерный массив переменных одного типа не могут иметь одинаковые имена в пределах программы.

Прежде чем числовому массиву переменных будут присвоены данные, необходимо зарезервировать место в памяти. Для этой цели должны быть использованы операторы DIM и COM.

Пример:

: 20 DIM A(I0), NI(5,5)

Оператор DIM резервирует место в памяти для одномерного массива переменных A, содержащего I0 элементов и для двумерного массива NI, содержащего 25 элементов. Числа, заключенные в скобки в данном случае определяют количество строк и столбов в массиве и называются размером массива. При резервировании места для массива переменных нужно придерживаться следующих правил:

I) нумерация элементов массива начинается с единицы, а не с нуля;

2) размерность массива переменных (одномерного и двумерного) не может быть более 65535.

I.4.4. Числовые выражения

Числовое выражение может состоять из переменной или константы, последовательности переменных и констант, разделенных знаками арифметических операций и числовыми функциями. Числовые выражения могут быть вычислены при помощи целого ряда операторов языка БЭЙСИК.

Чаще всего выражения вычисляются, и их значения присваиваются переменным при выполнении оператора присвоения (LET) или выражения вычисляются и их значения печатаются или выводятся на дисплей при выполнении оператора PRINT.

Примеры:

I) :I0 A=B

2) :20 PRINT 4*A+B

3) :30 N=N+SQR (A+B^2)

4) : 40 BI, B, B3=50

I.4.5. Знаки арифметических операций.

Для выполнения математических операций в языке БЭЙСИК используются следующие знаки операций:

+ сложение,
- вычитание,
* умножение,
/ деление,
^ возведение в степень.

I.4.6. Порядок вычисления.

Вычисление выражения производится слева направо. Когда в выражении используются знаки различных операций, соблюдается следующий приоритет вычисления:

I - выполняется возведение в степень (^) слева направо,
2 - выполняются умножение и деление (* ,/) слева направо,
3 - выполняются сложение и вычитание (+,-) слева направо.

Нормальный порядок вычислений может быть изменен при помощи круглых скобок. Если круглые скобки включены в выражение, то часть выражения, заключенная в скобки, вычисляется первой. При составлении выражений круглые скобки могут быть заключены в другие скобки (число пар круглых скобок практически неограничено).

I.4.7. Округление.

Результаты выполнения всех арифметических операций (+, -, * , /, ^) , а также функций, всегда округляются до I3 значимой цифры.

Если арифметическое выражение состоит из нескольких арифметических операций, то округление производится после каждой операции, а не результата в целом.

I.4.8. Числовые функции.

В арифметических выражениях могут быть использованы математические и тригонометрические функции. Для записи функции используется несколько букв имени функции и заключенные в круглые скобки один или несколько аргументов функции. Большинство функций являются обычными математическими или тригонометрическими функциями и не требуют дальнейшего разъяснения. Те же функции, которые требуют дополнительного обсуждения , будут описаны в следующих подпунктах.

Математические функции включают в себя:

1) ABS (аргумент) - абсолютное значение аргумента

Примеры:

ABS (7^2) =49

ABS (-4.32)=4.32

2) EXP (аргумент) - число, равное числу E в степени аргумента

Примеры:

EXP (.33*(5-6)=.71892373343I9

EXP (I)=2.7I828I82846

3) INT (аргумент) - наибольшее целое число, ближайшее к аргументу, но не более его

Примеры:

INT (I0.4)=I0

INT (-6.37)=-7

4) LOG (аргумент) - натуральный логарифм аргумента

Пример:

LOG (3052)=8.023552392404

5) RND(аргумент) - случайное число в диапазоне между 0 и I

Пример:

RND (X)=.73954

6) ROUND (X,N) - величина X, округленная до N десятичного знака, если N > 0; округленную до ближайшего целого, если N = 0; округленную до /N/ +I знаков влево от десятичной запятой, если N< 0.

Примеры:

ROUND (I.234,2)= I.23

ROUND (5.06,I)=5.I

ROUND (5.5,0)=6

ROUND (-3.8,0)=-4

ROUND (-3.2,0)=-3

ROUND (I6,-I)=20

ROUND (-I5I,-2) =-200

7) SGN (агрумент) - знак аргумента, равный:

I при аргументе > 0

0 при аргументе = 0

-I при аргументе < 0

Примеры:

SGN (7.I2)=I

SGN (0) = 0

SGN (-I.I7)=-I

8) SQR (аргумент) - квадратный корень от значения аргумента

Пример:

SQR (25)=5

I.4.8.I. Функция INT.

Функция INT является функцией наибольшего целого числа. Для значения целого числа, функция INT тождественна значению исходной величины (например, INT(4)=4, INT (-4)=-4). Для нецелых чисел, функция INT дает наибольшее целое число.

Примеры:

INT(3.8)=3
INT(-2.6)=-3
INT(30)=30

I.4.8.2. Функция RND.

Функция RND дает случайные числа со значениями от 0 до I. RND можно рассматривать как средство извлечения случайных чисел между 0 и I из "списка" случайных чисел. RND различает только два типа аргументов: равен нулю и не равен нулю. Если аргумент равен нулю, RND извлекает из "списка" случайных чисел первое число, если нет - очередное случайное число. Величина аргумента RND, не считая того, что он не равен нулю, не имеет отношения к случайно полученному числу. Когда RND исполняется в первый раз, она дает первое число из случайного "списка" если аргумент не равен нулю, и первое число из постоянного списка если аргумент равен нулю. Во второй раз она дает следующее число из "списка" и т.д. каждый раз, когда выполняется функция RND с аргументом, не равным нулю, она дает новое случайное число. Если необходимо повторно использовать ту же последовательность случайных чисел (например, во время отладки), функция RND с аргументом, равным нулю, может быть использована для "возврата" списка к первому случайному числу.

Пример:

```
:I0 PRINT  RND (0)
:20  FOR  N = I  TO 99
:30 PRINT  RND (I)
:40  NEXT N
```

В данном примере программа выдает на печать первые I00 случайных чисел из "списка" случайных чисел каждый раз, когда программа выполняется. Если строка I0 отсутствует, то программа выдает различные ряды случайных чисел при каждом прогоне программы.

Первое применение функции RND с аргументом не равным нулю сразу после пуска системы или выполнение команды CLEAR дает первое число из случайного "списка". При последующем втором применении функции RNDс аргументом не равным нулю система дает следующее число из "списка" и т.д.

I.4.8.3. Функция SGN .

Функция SGN выполняет сравнение аргумента с нулем. Если аргумент отрицательный, значение функции SGN равно -I, если аргумент равен нулю, значение функции равно нулю, если аргумент положительный, значение функции равно +I.

Примеры:

I) SGN (.000I)=I
2) SGN (-9.86)=-I
3) SGN(0)=0

I.4.8.4. Функция ROUND .

Функция ROUND используется для округления значения с заданной точностью. Функция ROUND имеет два аргумента: первый аргумент - это выражение, значение которого должно быть округлено, второй аргумент - это параметр округления. Если параметр округления не является целым числом, его дробная часть автоматически отбрасывается.

Пример:

ROUND (5.375,2)
/ /
округля - параметр
емое значение округления

Если параметр округления (N) > 0, то производится округление до N - цифры после точки.

Примеры:

I) ROUND (3.6789,I)=3.7
2) ROUND (3.6789,2)=3,68
3) ROUND (3,6789,3)=3.679

Если параметр округления равен нулю, то значение округляется до ближайшего целого числа.

Примеры:

I)ROUND (3.25,0)=3
2) ROUND (5.5,0)=6

3) ROUND (-3.2,0)=-3

Если параметр округления меньше нуля, то округление производится до (/N/+1) цифры влево от десятичной точки.

Примеры:

1) ROUND (651,-1)=650
2) ROUND (1256,-2)=1300
3) ROUND (-1601,-3)=-2000

I.4.8.5. Тригонометрические функции.

Тригонометрические функции SIN , COS , TAN и их обратные функции ARCSIN , ARCCOS , ARCTAN могут быть вычислены в радианах,градусах или градах (360 градусов равны 400 градам).Обычно все тригонометрические функции выполняются в радианах. Если вычисления должны быть сделаны в градусах или градах, то это должно быть задано в операторе SELECT, до выполнения тригонометрических функций. Для этого используются следующие операторы:

SELECT D - все последующие тригонометрические вычисления должны быть выполнены в градусах;

SELECT G - все последующие тригонометрические вычисления должны быть выполнены в градах.

Радианы автоматически выбираются системой для вычислений тригонометрических функций при инициации системы или по оператору CLEAR. Радианы также могут быть выбраны для вычислений при выполнении оператора SELECT R.

Тригонометрические функции включают в себя:

1) SIN (аргумент) - найти синус аргумента

Пример:

SIN (# PI/3)=.8660254037848

2) COS (аргумент) - найти косинус аргумента

Пример:

COS (.693 ^ 2)=.8868799122688

3) TAN (аргумент) - найти тангенс аргумента

Пример:

TAN (12)=-.6358599286616

4) ARCSIN (аргумент) - найти арксинус аргумента

Пример:

ARCSIN (.003)=3.000004502-03

5) ARCCOS (аргумент) - найти арккосинус аргумента

Пример:

ARCCOS (.578)=.943480794406

6) ARCTAN (аргумент) - найти арктангенс аргумента

Пример:

ARCTAN (3.2)=I.2679II45842

I.4.9. Ошибки в вычислениях.

Ошибки в вычислениях могут быть совершены во время выполнения арифметических операций или вычисления числовых функций. Обычно, когда в вычислениях встречается ошибка, система выдает сообщение об ошибке и прекращает выполнение программы. Если в программе используется оператор ONERROR, сообщение об ошибке не выдается, выполнение программы при ошибке не прекращается, а осуществляется переход в программе, заданный оператором ON ERROR. Таким образом производится программная обработка ошибок.

I.5. Операции с текстами.

I.5.I. Символьная строка .

Язык БЭЙСИК 02 обеспечивает широкие возможности по обработке информации в виде символьных строк. Символьная трока - это последовательность символов, которая рассматривается как одно целое.

Символьная строка может состоять из любых комбинаций символов на клавиатуре, включая буквы A - Z, числа 0 - 9 и специальные символы +, -, Д и т.д. символы, которых нет на клавиатуре, могут быть представлены в виде шестнадцатиричных кодов. Примерами символьных строк являются наименования, адреса и заголовки сообщений.

Символьные строки могут быть представлены в программе в двух основных формах:

I) в виде строк символьных констант;

2) в виде строк символьных переменных.

I.5.2. Символьная константа .

Символьная константа представляет собой строку символов, заключенную в кавычки (") или в апострофы (') или строку, состоящую из одного или более шестнадцатиричных кодов, предоставленных функцией HEX.

Примеры:

```
I) :IO X=9:   PRINT " VALUE OF  X=";  X
   : RUN
   VALUE OF X=9
2) : IO PRINT "BASIC 02"
   : RUN
```

BASIC 02

Символьная константа печатается точно в таком виде, в котором она задана в программе. Символьные константы могут быть присвоены символьным переменным.

Пример:

```
:10 A$ = "BASIC  02"
: 20 PRINT  A$
: RUN
BASIC 02
```

Символьная константа может иметь любую длину, не превышающую максимальную длину строки программы. Однако, когда символьная константа заносится в символьную переменную, то она обрезается по длине символьной переменной.

Пример:

```
:10 DIM  A$5
:20  A$ = "123456789"
:30 PRINT  A$
:RUN
12345
```

Минимальная длина символьной константы - один байт, но допустима и пустая символьная константа (" " или ' '). Символьная константа может содержать любые символы, включая двоеточие, но не может содержать кавычки в символьной константе " " и апострофы в символьной константе ' '.

Для представления специальных символов, которых нет на клавиатуре системы, используется шестнадцатиричная константа. Шестнадцатиричные коды состоят из пар шестнадцатиричных цифр (0 - 9 или A - F). Например, шестнадцатиричный код 03 является управляющим кодом, который очищает дисплей, так оператор;

```
: 50 PRINT  HEX (03)
```

при исполнении очистит дисплей. Шестнадцатиричные константы допустимы во всех случаях, когда допустимы символьные константы. В частности, они могут быть присвоены символьным переменным во время выполнения оператора присвоения.

Пример:

```
: 60  A$ =HEX(313233)
: 70 PRINT  A$
: RUN
123
```

напечатаны символы "123", так как они представлены шестнадцатиричными кодами (31, 32, 33).

I.5.3. Символьные переменные.

Символьные переменные отличаются от числовых переменных наличием знака $, который следует за именем переменной (A$).

Числовая переменная и символьная переменная являются отдельными независимыми переменными, даже, если они имеют одно и то же имя.

Данные, записанные в символьной переменной, могут участвовать в логических операциях, в выполнении функций символьных переменных, а также в двоичных математических операциях и в операциях с упакованными десятичными данными. Но они не могут быть использованы в стандартных десятичных арифметических операциях. В арифметических операциях могут использоваться только числовые данные.

Символьные переменные бывают двух типов: скалярные символьные переменные и символьные массивы. Символьная скалярная переменная может быть занесена в память в виде одной строки символов длиной от I до 253 символов.

Символьный массив состоит из одного или нескольких элементов массива. Каждый элемент имеет длину от I до 253 символов.

Массив переменных дает возможность пользователю обращаться к совокупности данных, объединенных одним именем массива (при определенных условиях строки отдельных символов, занесенные в элементы символьного массива, могут рассматриваться как единое целое, как одна строка смежных символов).

Массивы символьных переменных могут быть одномерными и двумерными. Одномерные массивы представляют собой аналоги списков с одним рядом элементов. Двумерные массивы являются аналогами таблиц, состоящих из строк и столбцов элементов.

Скалярные переменные и массивы переменных рассматриваются системой, как отдельные независимые типы переменных, в то время как переменные одномерных и двумерных массивов являются различными, но родственными видами массивов. Таким образом, в программе может быть использовано одно и то же имя для символьной скалярной переменной и для массива символьных переменных, но для одномерного символьного массива и двумерного символьного массива должны быть использованы разные имена.

Например, оба имени переменных B¤ и B¤(5) могут быть использованы в одной и той же программе, а B¤(5) и B¤(6,6) не могут быть использованы в одной и той же программе.

К символьному массиву можно обратиться как к скалярной переменной, содержащей одну строку символов, там где разрешены символьные переменные. Когда массив используется как скалярная переменная, то он обозначается наименованием символьного массива, который состоит из имени массива, за которым следуют круглые скобки. Например A¤( ), B4¤( ) и т.д.

Возможность обращения ко всему массиву, как к простой переменной, дает возможность программе легко производить действия с чрезвычайно длинными строками символов, поскольку размер символьного массива ограничен только размером памяти машины.

I.5.4. Длина символьной переменной.

Для каждой символьной переменной программы система резервирует место в памяти. Размер памяти, зарезервированный для каждой переменной, может быть задан при помощи операторов DIM или COM, в которых можно определить длину символьной переменной. Максимальная длина символьной переменной или одного элемента символьного массива составляет 253 символа, а минимальная длина в обоих случаях I символ. Если размер символьной переменной в операторах DIM или COM отсутствует, то система автоматически резервирует I6 байтов для хранения символьной переменной. Однако, в ряде случаев хвостовые пробелы не считаются частью значения символьной переменной. Например:

```
: I0  A¤ = "ABC":  B¤ = "B "
: 20 REPLACE  N, A¤ , B¤
: 40 PRINT  A¤
: RUN
AC
```

С помощью функции символьной переменной LEN всегда можно определить длину символьной переменной без хвостовых пробелов, например:

```
: I0 A¤ = "ABC "
: 20  PRINT  LEN  (A¤)
3
```

I.5.5. Функции символьных переменных LEN , NUM, POS , STR.

I.5.5.I. Функция LEN.

LEN (<символьный аргумент>).

Числовая функция LEN является функцией длины и определяет количество символов в значении символьной переменной или в определенной части символьной переменной.

Хвостовые пробелы функция LEN не считает частью значения символьной переменной. LEN просматривает переменную и дает число символов до первого хвостового пробела (пробелы внутри и в начале значения символьной переменной считаются частью значения). Если символьная переменная представляет собой одни пробелы, значение функции формируется равным единице.

Примеры:

```
1) : 10 A$ =" ABCD "
   : 20 PRINT LEN (A$)
   : RUN
   4
2) : 10 A$ = "A BCD"
   : 20 PRINT LEN (A$)
   : RUN
   5
3) : 10 A$ = " "
   : 20 PRINT LEN (A$)
   : RUN
   I
```

Функция LEN может быть использована во всех случаях, когда допустимы числовые функции.

Примеры:

```
1) : 10 X= LEN (A$) +2
2) : 50 IF LEN (A$(3)) < 8 THEN 100
3) : 20 X= LEN (A$() )
4) : 90 PRINT LEN (STR(A$,3,8))
```

I.5.5.2. Функция NUM.

NUM (<символьный аргумент>)

Числовая функция NUM определяет количество последовательных числовых символов в обозначенном символьном аргументе, которые представляют собой допустимые числовые символы языка БЕЙСИК (в строго определенной последовательности формата числа).

Числовой символ может быть представлен следующим образом: пробелы,

цифры 0 – 9, десятичная точка (.), знаки плюс и минус (+, –), буква Е.

Числовые символы считаются, начиная с первого символа заданного аргумента или функции STP. Подсчет символов заканчивается. Если встречается нечисловой символ, или когда последовательность числовых символов не соответствует стандартному числовому формату БЭЙСИК, или когда все символы аргумента просмотрены. Ведущие и хвостовые пробелы включаются в счет.

Таким образом, функция NUM может быть использована для проверки допустимого представления символьного значения на языке БЭЙСИК или для определения длины числовой части символьного аргумента (представление числа на языке БЭЙСИК не может иметь мантиссу более, чем из 13 цифр).

Функция NUM может быть использована во всех тех случаях, когда обычно используются числовые функции и особенно полезна для применения в тех случаях, когда желательно проверить числовую истинность данных, введенных во время выполнения программы.

Примеры:

```
1)  : 10 A¤ = "+24.37#L"
    : 20 X= NUM(A¤)
    : 30  PRINT "X="; X
    : RUN
    X=6
2)  : 10 A¤ ="98.7+53.6"
    : 20 X= NUM (A¤ )
    : 30 PRINT "X=" ; X
    : RUN
    X=4
```

I.5.5.3. Функция POS.

POS (<символьный аргумент> <операция сравнения> <символьный аргумент I >) <операция сравнения> :: = <меньше> / <меньше или равно> / <больше> / <больше или равно> /<равно> /<не равно>

Числовая функция положения POS сравнивает (в соответствии с условием) каждый символ символьного аргумента, начиная с начала, с заданным символом. Первый же символ символьного аргумента, удовлетворяющий заданному условию сравнения, завершает выполнение функции. Положение данного символа в виде числового значения по отношению к первому символу в символьном аргументе запоминается. Таким образом,

при функции POS просматривается вся длина символьного аргумента, включая хвостовые пробелы. Если ни один символ не удовлетворяет отношению, функция POS = 0. Функция POS может быть использована во всех случаях, где допустима числовая функция.

Примеры:

```
1) :10 X=POS (A$ = "S ")
2) : 20PRINT  POS  (STR (A$, I, J ) =HEX (0D ))
3) : 30 IF POS (A$( )= "T")=0 THEN 100
4) : 10 A$ =" ABCD "
   : 20 X=POS  (A$ = "D ")
   : 30 PRINT "X=" ; X
   : RUN
   X = 4
```

Переменная X принимает значение 4 (так как символ D является четвертым символом символьной переменной.)

I.5.5.4. Функция STR .

STR (<символьный элемент> , <AB>[, <AB>])

Символьная псевдофункция STR дает возможность выборки, проверки, сравнения или изменения любых символов в символьном элементе или в его части.

Первый параметр функции STR, заданный <AB> определяет номер символа, второй - количество символов, при определении параметра используется целая часть арифметического выражения. Если параметр <количество символов> опущен, то выбираются символы от символа с заданным номером до конца символьного элемента, включая хвостовые пробелы.

Пример:

```
: 10 A$ = "ABCDE"
: 20 PRINT STR (A$, 3)
: RUN
CD E
```

Функция STP может использоваться по обе стороны от знака равенства в операторе присвоения. Функция STP может быть использована во всех случаях, когда допустимы символьные переменные.

Примеры:

```
1) A$ = STP (B$, 2,4)
2)STP (A$, I,4)=B$
```

```
3) : 10 B¤ = "ABCDEF"
   : 20 A¤ = STR (B¤ ,1,4)
   : 30  PRINT "A ¤ =" ; A¤
   : RUN
   A¤ = ABCD
```

4) :10 STR (A¤,4)=C¤ в переменную A¤ с четвертого по шестнадцатый символ помещается цепочка символов со значением переменной C¤.

```
5) : 10  A¤ =" 0123456789ABCDEF "
   : 20 B¤= "IIIIIIIIII"
   : 30 STR  (B¤, 2,7) =STR  (A ¤ , 10, 7)
   : RUN
   B¤= "I9ABCDEFII"
```

2. ЭЛЕМЕНТЫ ЯЗЫКА

2.I. Логические операции и операции над двоичными числами.

2.I.I. Оператор двоичного сложения ADD.

ADD [C] <Логические аргументы>

<Логические аргументы> :: = (<Символьная переменная>, <Символьный аргумент I>)

<Символьный аргумент I> :: = <Символьный аргумент>/<Элемент HEX>

Оператор ADD используется для сложения двоичных значений логических аргументов.

Результат заносится в первую символьную переменную. Действие выполняется со всем значением символьной переменной, включая хвостовые пробелы. Действия могут производиться и над частью символьной переменной, когда часть переменной обозначается функцией STR.

Если за знаком операции ADD не следует "C", то сложение выполняется символ за символом справа налево, перенос между байтами отсутствует.

Если за ADD следует "C", то значение прибавляемого аргумента рассматривается как одно двоичное число и прибавляется к двоичному значению переменной с учетом переноса между байтами.

Если складываемые аргументы имеют разную длину, то производится выравнивание по правому разряду, для более короткого значения вводятся нули. Результат присваивается первой переменной и если он длиннее переменной, то запоминается допустимая часть младших байтов, остальные символы отсекаются.

Примеры:

```
I) :5  DIM  A$ 2
   :IO  A$ = HEX (OIA3)
   :20  ADD     (A$, 82)
```

При выполнении программы производятся действия:

```
A$ = 0000 0001 1010 0011 (01A3)
     1000 0010 1000 0010 (8282)
     1000 0011 0010 0101 (8325)
```

и получается результат: A$ = HEX (8325) - шестнадцатиричные цифры добавляются к каждому байту первой символьной переменной и перенос между байтами отсутствует.

```
2) :5 DIM  A$2
   :IO A$ = HEX(01A3)
   :20 ADD C  ( A$, 82)
```

При выполнении программы производятся действия:
A¤ = 0000 0001 1010 0011 (01A3)
82 = 0000 0000 1000 0010 (82)

0000 0010 0010 0101 (0225)

и получается результат A¤ = HEX (0225) - складываются два двоичных числа, перенос между байтами существует.

3):5 DIM A¤3, B¤1
:10 A¤ = HEX (012345): B¤ = HEX (FF)
:20 ADD(B¤, STR (A¤ 2,2,))

при выполнении программы производятся действия:
B¤ = 0000 0000 1111 1111 (00FF)
STR (A¤ , 2,2) = 0010 0011 0100 0101 (2345)

0010 0011 0100 0100 (2344)

и получается результат B¤ = HEX (44) - перенос между байтами отсутствует.

Так как B¤ и A¤ имеют разную длину, то недостающие позиции переменной B¤ заполняются нулями. Результат присваивается переменной B¤ , длина которой меньше длины полученного результата, поэтому запоминается только младший байт результата - HEX (44).

2.1.2. Преобразование двоичных чисел в десятичные VAL.

VAL (<Символьная переменная> [,2]) .

Функция VAL предназначена для преобразования двоичного значения первого символа (или первых двух символов, если есть параметр "2") символьной переменной в десятичное число. Эта функция обратна оператору BIN .

Примеры:

1) :10 A¤ = "ABC"
:20 X = VAL (A¤)
:30 PRINT "X ="; X
:RUN
X=65

2) :10 A¤= "ABCDEF"
:20 IF VAL (STR (A¤, 3, 1) <80 THEN 40
.
.
:40 PRINT VAL(STR (A¤ , 3, 1))
:RUN
67

Если функцией VAL задано преобразование двух символов символьной

переменной , а символьная переменная однобайтовая, то преобразование выполняется без учета параметра [,2].

2.I.3. Преобразование десятичных чисел в двоичные BIN.

BIN (<символьная переменная> [,2])= <AB>

Оператор BIN преобразует целую часть числового выражения (которая должна быть меньше 256 при однобайтовой форме и меньше 65535 при двухбайтовой форме оператора BIN) в двоичное число и присваивает это значение первому символу или первым двум символам символьной переменной.

Оператор BIN является обратным функции VAL .

Примеры:

```
I) :I0 BIN (A$) =40^2/I9.8
   :20 PRINT  A$
   :RUN
   P
2) :I0  A$= "ABCDEFG"
   :20 BIN (A$) =64
   :30 PRINT  A$
   :RUN
  @BCDEFG
```

2.I.4. Логические операции.

В языке БЭЙСИК 02 есть несколько операторов, которые позволяют изменять один или несколько битов в байте данных. Операторы AND, OR, XOR, BOOL и ADD выполняют различные логические операции над битовой структурой одного или нескольких символовсимвольной переменной.

При операциях с переменными учитывается полная длина переменных, включая и конечные пробелы. Для оперирования с частью переменной используется функция STR.

С помощью этих операторов производятся логические операции двух типов в зависимости от типа второго параметра оператора:

I) символьная переменная;

2) две символьные переменные.

Логические операции выполняются по символам слева направо, начиная с самого левого символа каждого поля. Если длина второго аргумента меньше длины первого аргумента, то остающиеся символы первого аргумента не изменяются.

Если длина второго аргумента больше длины первого аргумента, то операция заканчивается, когда над последним символом первого аргумента было выполнено действие.

2.I.5. AND .

AND<логические аргументы>

Оператор AND выполняет операцию логического умножения (логического И). Побитно сравнивая два аргумента, оператор присваивает значение единица результирующему биту, если соответствующие биты сравниваемых величин равны единице, в противном случае-нуль. Результат присваивается первому аргументу.

Примеры:

```
I) : I0  A%=HEX(0C):B%=HEX(08)
   : 20 AND  (A%, B% )
```

при выполнении программы производятся действия:

```
  A%= 0000 1100 (0C)
  B%= 0000 1000 (08)
      0000 1000
```

получается результат: A%=HEX(08)

```
2) :I0 DIM A%2
   :20  A%=HEX(25A4)
   :30 AND  (A%,F0)
```

при выполнении программы производятся действия:

```
 A% = 0010 0101 1010 0100    (25A4)
      1111 0000 1111 0000    (F0F0)
      0010 0000 1010 0000
```

получается результат: A% =HEX(20A0)

2.I.6. OR .

OR <логические аргументы>

Оператор OR выполняет операцию логического ИЛИ. Побитно сравнивая два аргумента, оператор присваивает значение нуль результирующему биту, если соответствующие биты аргументов имеют нулевое значение, в противном случае - единица. Результат присваивается первой переменной.

Примеры:

```
I) : I0 DIM  A%I,  B%I
   : 20   A%=HEX(0C): B%=HEX(08)
   : 30 OR ( A%, B%)
```

при выполнении программы производятся действия:

```
 A% = 0000 1100  (0C)
 B% = 0000 1000  (08)
      0000 1100
```

Получается результат: A¤ = HEX (OC)

2) :I0 DIM A¤2, B¤2
:20 A¤ = HEX (25A4)
:OR (A¤, F0)

При выполнении программы производятся действия:

A¤ = 0OI0 0I0I I0I0 0I00 (25A4)
I I I I 0 0 0 0 I I I I 0 0 0 0 (F0F0)
I I I I 0 I 0 I I I I 0 I 0 0

Получается результат: A¤ = HEX (F5F4)

2.I.7. XOR.

XOR <логические аргументы>

Оператор XOR выполняет операцию исключающего ИЛИ. Побитно сравнивая два аргумента, оператор присваивает значение единица результирующему биту, если соответствующие биты различны и значение нуль, если соответствующие биты совпадают. Результат присваивается первой переменной.

Примеры.

I) :I0 DIM A¤I, B¤I
:20 A¤= HEX (OC): B¤ =HEX (08)
:30 XOR (A¤, B¤)

При выполнении программы производятся действия:

A¤ =0000 II00 (OC)
B¤ =0000 I000 (08)
0000 0I00 (04)

и получается результат: A¤ = HEX (04)

2) :I0 DIM A¤2
:20 A¤ = HEX (25A4)
:30 XOR (A¤, F0).

При выполнении программы производятся действия:

A¤ = 0OI0 0I0I I0I0 0I00 (25A4)
I I I I 0 0 0 0 I I I I 0 0 0 0 (F0F0)
I I 0 I 0 I 0 I 0 I 0 0

и получается результат: A¤ =HEX (D554)

2.I.8. BOOL .

BOOL <шестнадцатиричная цифра> <логические аргументы>

Оператор BOOL дает возможность выполнить любую из I6 возможных логических операций (первая шестнадцатиричная цифра от нуля до F, которая следует сразу же за BOOL).

Логические операции выполняются над всеми символами символьной переменной слева направо. Результат логических операций (за исключением операции 5) присваивается первой символьной переменной. Для операций с частью символьной переменной можно использовать функцию STR.

Коды логических операций (см. табл. I) являются мнемоническим средством представления логических результатов выполнения операций над двумя величинами X и У (1100и 1010).

Таблица I

Коды	Значение		Результат
	X	У	
Двоич-	I	I	0 0 0 0 0 0 0 0 I I I I I I I I
ные	I	0	0 0 0 0 I I I I 0 0 0 0 I I I I
коды	0	I	0 0 I I 0 0 I I 0 0 I I 0 0 I I
	0	0	0 I 0 I 0 I 0 I 0 I 0 I 0 I 0 I
Шестнад - цатирич- ные коды			0 I 2 3 4 5 6 7 8 9 A B C D E F

0- нулевая;
I - отрицание НЕ ИЛИ;
2 - переменная X не имплицируется;
3 - дополнение X;
4 - переменная X не имплицирует У;
5 - дополнение У;
6 - исключающее ИЛИ;
7 не И;
8 - И;
9 - эквивалентность;
A - переменная X=У;
B - переменная X имплицирует У;
C - У равна переменной X;
D - У имплицирует переменную X;
E -ИЛИ;
F - тождество.

Примеры:

```
I) :5 DIM  A¤ 2
   :IO  A¤ = HEX (5432)
   :20 BOOL  3 ( A¤, 00)
```

При выполнении программы производятся действия согласно правилам, соответствующим операции 3;

```
A¤ = 0101 0100 0011 0010 (5432)
00 = 0000 0000 0000 0000 (0000)
     1010 1011 1100 1101
```

и получается результат A¤ = HEX (ABCD)

```
2):5 DIM  A¤2,  B¤2
   :IO  A¤= HEX (4145):B ¤ = HEX (2185)
   :20 BOOL  7  ( A¤, B¤)
```

при выполнении программы производятся действия согласно правилам, соответствующим операции 7;

```
A¤ = 0100 0001 0100 0101 (4145)
B¤ = 0010 0001 1000 0101 (2185)
     1111 1110 1111 1010
```

и получается результат A¤ = HEX (FEFA) .

2.2. Выбор устройств ввода-вывода.

2.2.1. SELECT.

SELECT <Список SELECT>

<Список SELECT> :: =<параметр SELECT> /<Список SELECT>,
<параметр SELECT>

<Параметр SELECT> :: = CI <Элемент HEX> / CO <Элемент HEX>
[(<Длина строки>)]
/TAPE <элемент HEX>[(<Длина строки>)] / LIST <элемент HEX>
[(<длина строки>)] / PRINT <элемент HEX> [(<длина строки>)]/
PLOT <элемент HEX> [(<длина строки>)] / DISK <элемент HEX> F
/ DISK <элемент HEX> R /# <цифра> <элемент HEX> [F] /# <цифра>
<элемент HEX> [R]/P [<цифра>]/D/ R / G

<Длина строки> :: = <цифра> / <цифра> <цифра> / <цифра>
<цифра> <цифра>/ <цифра> <цифра><цифра><цифра>

Оператор SELECT используется в трех случаях:

1) для выбора желаемого режима при вычислении тригонометрических функций:

2) для выбора параметров вывода при общении с выводимыми устройствами;

3) для выбора адресов устройств в операторах ввода-вывода.

2.2.2. Выбор градусов, радианов или градов .

При помощи соответствующего использования параметров D, R или G могут быть выбраны следующие единицы измерения для аргументов тригонометрических функций: градусы, радианы или грады.

Например: оператор SELECT D заставляет систему использовать градусы для тригонометрических функций. Единица измерения может быть изменена при выполнении другого оператора SELECT или при запуске системы заново, когда автоматически выбираются радианы.

2.2.3. Выбор паузы .

При выборе параметра P система приостанавливается на заданную паузу каждый раз, когда код возврата каретки выдается на дисплей. Такая особенность дает возможность пользователю замедлить последовательный вывод на экран при необходимости просмотра выполнения программы, например, в процессе трассировки.

Цифра, следующая за паузой, обозначает длину паузы с приращением на I/6 секунды. Например, следующие операторы выработали указанные паузы:

SELECT P1 пауза - I/6 секунды
SELECT P6 пауза - I секунда
SELECT P (или P0) пауза - нулевая (т.е. отсутствие паузы)

Обозначенная пауза остается до нового запуска системы или до выбора другой паузы. Выбор P или P0 уничтожает текущую паузу.

2.2.4. Выбор длины строки .

Для операций, связанных с печатью и воспроизведением символов на экране дисплея (PRINT, LIST, вывод на консольное устройство), система обычно использует автоматический перевод строки, когда текст превышает длину строки выводного устройства.

Длина строки выбирается после адреса устройства при выполнении операторов LIST, PRINT и выводе на консольное устройство (CO):

2.2.5. Выбор адресов устройств .

Каждое вводно-выводное устройство, связанное с системой, имеет свой индивидуальный адрес устройства. Адрес устройства состоит из двух шестнадцатиричных цифр.

При выполнении операторов ввода-вывода соответствующее устройство может быть выбрано одним из трех способов:

I) отсутствие адреса. Если адрес устройства не обозначен или не выбран, то система автоматически определяет адрес устройства,

который употребляется для данной конкретной операции по умолчанию;

2) выполнен оператор SELECT. При этом в операторах ввода-вывода используется устройство, заданное оператором SELECT;

3) адрес устройства задан оператором ввода-вывода.

2.2.6. Отсутствие задания адреса устройства.

Система имеет пять вводно-выводных устройств, предназначенных для основных устройств ввода-вывода. Адреса этих устройств используются по умолчанию каждый раз, когда система запускается вновь. Эти адреса автоматически выбираются для операций ввода-вывода без заданного адреса устройства. Основные устройства ввода-вывода следующие:

1) консольное вводное устройство (CI) - клавиатура (адрес 01)
2) консольное выводное устройство (CO) - дисплей (адрес 05)
3) консольное устройство DISK - накопитель на гибком магнитном диске (адрес 18)Г
4) консольное устройство PLOT - графический дисплей (адрес 10)
5) консольное устройство TAPE - кассетный накопитель на магнитной ленте (адрес 08)

Если в системе нет дополнительных вводно-выводных устройств, тогда обозначать адреса устройств или выбирать их в операторах БЭЙСИК для выполнения вводно-выводных операций не нужно. Таким образом, при выполнении дисковых операторов происходит обращение к диску с адресом 18Г, операторы PRINT, LIST обращаются к дисплею с адресом 05, оператор PLOT обращается к графическому дисплею с адресом 10. Однако, если в системе используются другие периферийные устройства, то в таком случае указывать адрес устройства обязательно.

2.2.7. Таблица устройств.

Когда система получает инструкцию выполнить операцию ввод-вывод в виде оператора ввода-вывода, то она должна определить адрес устройства, который используется для этой операции. Если адрес устройства непосредственно не обозначен в самой инструкции (в некоторых группах операторов ввода-вывода непосредственно указывать адреса недопустимо), то этот адрес можно получить в специальной области памяти, которая называется таблицей устройств. Таблица устройств состоит из строк, каждая из которых может содержать адрес устройства (а в некоторых случаях и другую информацию) для конкретных операторов ввода-вывода. Когда система запускается вновь, то таблица устройств заполняется адресами консольных устройств, перечисленными для каждого оператора ввода-вывода

в предыдущем разделе, и имеет следующий вид:

```
   CI = 01 CO = 05 (80) PRINT=05(80) LIST = 05(80) TAPE=08(256)
    PLOT = 10
#0-18F
#1-
#2-
#3-
#4-
#5-
#6-
#7-
```

Адреса устройств могут быть явно занесены в таблицу устройств при помощи оператора SELECT. Например, оператор

```
   : SELECTPRINT  0C, # 31B
```

заносит адрес 0C в строку PRINT, а адрес 1B в строку # 3 таблицы устройств, и таблица устройств имеет следующий вид:

```
CI =01 CO=05(80) PRINT=0C(80) LIST =05(80) TAPE=08(256) PLOT=10
#0-18F
#1-
#2-
#3-1B
#4-
#5-
#6-
#7-
```

Использование каждой строки таблицы устройств описывается ниже.

CI - устройство, с которого производится ввод программ и операторов.

CO - устройство, на которое выводится изображение символов, введенных с клавиатуры;
распечатки об ошибках;
результат STOP и END;
результат TRACE;
вывод на экран при нажатии клавиши HALT/STEP;
результат LIST.

PRINT - устройство, на которое выводятся результаты операторов PRINT, PRINTUSING и HEXPRINT;

LIST - устройство, на которое выводится результат оператора LIST;

PLOT - устройство, на которое выводятся результаты графических операторов.

2.2.8. Изменение таблицы устройств.

Параметры в таблице устройств могут быть явно изменены при помощи оператора SELECT или неявно оператором CLEAR, нажатием клавиши RESET или при запуске БЭЙСИК-системы заново. Поэтому параметры в таблице устройств могут оставаться действительными до выполнения одной из следующих операций:

1) выполнение оператора SELECT, когда один или более из обозначенных параметров явно получают новое определение;

2) выполнение оператора CLEAR без указания параметров;

3) нажатие клавиши RESET;

4) запуск БЭЙСИК-системы заново.

Параметры в таблице устройств могут быть явно изменены при выполнении оператора SELECT с одним или более параметрами SELECT. Например, для того, чтобы изменить консольное выводное устройство (адрес дисплея 05) на другое консольное выводное устройство, используется оператор следующей формы:

SELECT CO<Элемент HEX> [(<Длина строки>)]

примером такого оператора является следующий оператор

: SELECT CO IB (377)

Этот оператор выбирает устройство, адрес которого IB, в качестве нового консольного выводного устройства. Длина строки, которая может быть использована на выбранном устройстве, равна 377 символам.

Дисплей может быть выбран вновь в качестве консольного выводного устройства при помощи оператора SELECT следующей формы:

:SELECT CO 05 (80)

Этот оператор вновь выбирает дисплей с адресом, равным 05, в качестве консольного выводного устройства, и вновь устанавливает длину строки 80 символов.

Если длина строки для консольного вывода операторами PRINT или LIST не обозначена, то длина строки остается прежней (377). Запуск системы вновь устанавливает длину строки из 80 символов для дисплея.

Дисковые операторы могут обращаться к другому дисковому устройству, а не на консольный диск при выполнении оператора:

SELECT DISK IC R

Этот оператор выбирает дисковое устройство с адресом IC в качестве консольного дискового устройства. После выполнения данного оператора все дисковые операторы выбирают диск с адресом IC, если в операторе прямо не обозначен номер файла или адрес

устройства.

Система обеспечивает также еще два способа выбора устройств для операторов ввода-вывода.

Многие дисковые операторы БЭЙСИК (LOAD, DATASAVE и т.д) могут содержать параметр, обозначенный #<AB> или <символ /> <элемент HEX>.

Параметр <символ /><элемент HEX>позволяет прямо включить фактический адрес дискового устройства в оператор. Этот метод выбора дисковых устройств не зависит от оператора SELECT. Например, оператор:

```
:LIST DCF/IC
```

печатает индекс каталога на диске, адрес которого IC. Обратите внимание на то, что указание адреса устройств в вводно-выводной инструкции не изменяет содержимое таблицы устройств.

Параметр #<AB> , который позволяет косвенно присвоить адреса устройств при помощи оператора SELECT, называется номером файла и должен иметь одну из следующих форм: #0, #1,...#7. Конкретный адрес может быть присвоен номеру файла одним из операторов SELECT в программе. (обратите внимание на то, что оператор формы

SELECT #0 <элемент HEX> эквивалентен оператору.

SELECT DISK <элемент HEX> .

После этого в программе все операторы ввода-вывода, содержащие обращение к данному номеру файла, автоматически используют уже присвоенный адрес устройства. Например, оператор

```
: SELECT #2IC,#3I8
```

присваивает адрес дискового устройства IC файлу #2 и адрес диска I8 файлу #3.

В последующих операторах программы, когда выполняются операторы ввода-вывода, номера файлов 2 и 3 могут использоваться для косвенного обращения к адресам IC и I8. Например, оператор:

```
: 90 DATALOAD DC OPEN F #3,"ДИСК"
```

автоматически открывает файл под названием "ДИСК" на дисковом устройстве с адресом I8.

Косвенное присвоение адресов устройств в программе при помощи номеров файлов имеет некоторые преимущества. Подпрограммы могут быть записаны так, чтобы выполнялась последовательность вводно-выводных операций несколькими устройствами и все присвоения адресов устройств в программе могут быть изменены путем модификации одного оператора. Например, в следующей программе адреса могут быть присвоены при изменении оператора I0.

```
:10 SELECT #218, # 310
:20 DATA LOADDC OPEN F# 2, "ЗАДАЧА"
.
.
.
:110 DATALOAD DC OPEN R#3, "ПРИМЕР"
```

Когда выполняется команда CLEAR без параметров, то для всех выводных операций выбирается консольное выводное устройство, находящееся в действии в данный момент, а для всех операций ввода выбирается консольное вводное устройство, которое используется в данный момент.

В таблице устройств делаются следующие изменения:

1) строки PRINT и LIST устанавливаются на консольный вывод CO;

2) строки #1-#7 устанавливаются на нули, включая присвоенные адреса.

Когда выполняется команда RESET, то в строках CO и CI в таблице устройств автоматически формируются адреса консольных устройств.

В таблицу устройств вносятся следующие изменения:

1) строка CO - адрес консольного устройства CO (05). Длина строки равна 80;

2) строка CI - адрес консольного устройства CI (01);

3) строка #0 имеет нулевые значения за исключением адреса диска, который не изменяется;

4) строки #1 - #7 имеют нулевые значения.

При запуске система присваивает адреса консольных устройств всем строкам в таблице устройств и устанавливает длину строк CO, PRINT, LIST равной 80.

Кроме того, строки #1 - #7 имеют нулевые значения.

Обратите внимание на то, что при операторе CLEAR без параметров, RESET и запуске системы стираются все присвоенные номера файлов. Затем все номера файлов должны быть установлены в необходимое положение при новом выполнении оператора SELECT. Обращение к неприсвоенному или стертому номеру файла вызывает ошибку.

2.2.9. Параметр PRINT.

Параметр PRINT обозначает выводное устройство: на которое выводятся результаты выполнения операторов PRINT, PRINTUSING и HEXPRINT. Данный параметр также может определять длину строки, которая должна быть использована для этого устройства. Например,

```
:100 SELECT PRINT  OC (100)
:110 PRINT "X = "; X, "NAME ="
:120 PRINTUSING 121, X
:121 % ##
```

Оператор SELECT PRINT в строке 100 направляет всю печать на выводное печатающее устройство с адресом устройства OC. Длина строки ограничена 100 символами.

```
:SELECT PRINT 05 (80)
```

Данный оператор вновь выбирает дисплей с адресом 05 в качестве уст-ва, куда направляются результаты выполнения операторов PRINT, PRINTUSING и HEXPRINT, а также устанавливает длину строки в 80 символов.

Результаты выполнения операторов PRINT, PRINTUSING и HEXPRINT при запуске системы появляются на консольном выводном устройстве.

2.2.10. Параметр LIST .

Параметр LIST обозначает выводное устройств, используемое для листинга всей программы и перекрестных ссылок. Данный параметр также определяет длину строки, которая должна использоваться для операций LIST на этом устройстве. Например,

```
: SELECT LIST OC (132)
```

означает использование печатающего устройства, адрес которого OC, для всех листингов. Длина строки равна 132 символам.

2.2.11. LIST % .

Оператор LIST % воспроизводит на дисплее содержимое таблицы устройств в шестнадцатеричном представлении. Данный оператор используется для отладки программ с обращением к таблице устройств.

Таблица воспроизводится в следующем формате:

```
CI =AA CO=AA(SS) PRINT=AA (SS) LIST=AA (SS) TAPE=AA (SS)
        PLOT =AA

#0-AAT
#1-
#2-
#3-
#4-
#5-
#6-
#7-
```

где: T- тип диска, AA - адрес устройства, SS - длина строки.

2.3. Обработка ошибок .

2.3.I. Типы ошибок .

Язык БЭЙСИК-02 обеспечивает обширный набор диагностируемых ошибок, разработанный для автоматического определения и регистрации условий ошибок. Выявление ошибок осуществляется при трансляции программы и во время ее выполнения. Ошибки, диагностируемые системой, можно разделить на следующие группы:

I) системные ошибки;
2) ошибки переполнения;
3) синтаксические ошибки;
4) ошибки счета по программе;
5) ошибки оператора ASMB;
6) ошибки ввода-вывода.

Системные ошибки возникают как правило в результате сбоя в работе процессора. При возникновении этих ошибок происходит останов системы и выдача сообщения об ошибке. В этом случае пользователь может попытаться запустить программу снова. При этом необходимо иметь ввиду, что сбой процессора может привести к произвольному искажению версии системы и программы пользователя. В этом случае следует перегрузить версию, затем загрузить программу и попытаться продолжить работу.

К системным ошибкам относятся следующие ошибки:

I) сбой системы;
2) сбой МО;
3) сбой ОЗУ;
4) сбой УП.

Сообщение "Сбой системы" вырабатывается при переходе системы через нулевой адрес, управляющей памяти. Сообщения "Сбой ОЗУ" и "Сбой УП" вырабатываются при сбое оперативной или управляющей памяти. Сообщение "Сбой МО" вырабатывается при сбое процессора и при использовании неотлаженных ассемблерных программ.

Ошибки переполнения возникают при отсутствии свободного места в памяти. При возникновении ошибок переполнения происходит останов системы и выдача сообщения об ошибке. При этом сбрасывается зона значений, а программа сохраняется.

Ошибки переполнения памяти могут возникнуть в следующих ситуациях:

I) в процессе трансляции, если не хватает места для расположения служебной информации или нет места для записи транслированной строки;

2) при просмотре программы, по клавише RUN, если не хватает места для построения временной таблицы описания переменных или не хватает места для отведения зоны значений;

3) в процессе счета по программе, если не хватает места на организацию внутренних магазинов или заданий на ввод-вывод.

При возникновении таких ситуаций выдается сообщение : "ERR 01" или "ERR 02". Сообщение "ERR 01" вырабатывается, когда переполнение памяти происходит из-за большого объема текста программы, а сообщение "ERR 02" вырабатывается при переполнении памяти переменными и их значениями.

К синтаксическим ошибкам относятся ошибки, выявляемые в процессе трансляции программы при вводе с клавиатуры, диска или из переменной и трансляции при вводе данных с клавиатуры.

Если в процессе ввода строки программы с клавиатуры обнаруживается синтаксическая ошибка, система повторно инициирует введенную строку, выдает сообщение "ERR 06" и стрелкой ^ указывает конструкцию, в которой обнаружена ошибка (приблизительно). При этом введенная строка запоминается в памяти (с признаком ошибочной строки) также как и строка без ошибок. Пользователь может исправить ошибку до начала ввода очередной строки.

Если такая строка не исправлена до момента выполнения программы, то попытка выполнить в программе такой оператор неизбежно приведет к формированию ошибки ERR 12 - неправильный оператор.

Если в процессе ввода данных с клавиатуры обнаруживается синтаксическая ошибка, система выдает сообщение "ERR 29" и требует от пользователя правильного ввода данных. На экране повторно появляется вопросительный знак, который указывает на необходимость повторного ввода.

При вводе программы с диска или из переменной обнаружение синтаксической ошибки приводит к появлению сообщения "ERR 07" или "ERR 10" соответственно и продолжению трансляции со следующей строки программы. При обнаружении синтаксической ошибки, трансляция данной строки прекращается, хотя за ошибочным оператором в этой строке могут следовать операторы без ошибок.

К ошибкам счета относятся ошибки, которые выявляются в процессе счета по программе. Эти ошибки можно сгруппировать следующим образом:

1) недопустимое использование операторов - ERR 08, ERR 09, ERR 13, ERR 25, ERR 27, ERR 37, ERR 44, ERR 48, ERR 58 и ERR 70;

2) неправильные ссылки на номера строк - ERR 11, ERR 15, ERR 31 и ERR 46;

3) параметр оператора превышает допустимое значение – ERR I8, ERR 20, ERR 32, ERR 4I, ERR 43, ERR 47, ERR 53, ERR 56, ERR 59;

4) ошибки при выполнении матричных операторов – ERR 49, ERR 50, ERR 5I, ERR 52;

5) ошибки при выполнении дисковых операторов – ERR 60, ERR 66, ERR 67, ERR 7I, ERR 72, ERR 73, ERR74, ERR 75, ERR 76, ERR 77 и ERR 78;

при возникновении ошибки счета индицируется строка, в которой была ошибка, затем на новой строке индицируется сообщение "ERR", за которым следует код ошибки.

К ошибкам оператора ASMB относятся ошибки, которые выявляются в процессе трансляции ассемблерной программы. При обнаружении ошибки индицируется оператор ASMB, сообщение "ERR I4" и две цифры, указывающие уточненное состояние при трансляции, выдаваемое ассемблером. При этом надо иметь ввиду, что ошибки трансляции ассемблерной программы с уточненным состоянием 0I, 02, 04, 05, 06, 08, 09 или I2 позволяют довести трансляцию ассемблерной программы до конца. При возникновении ошибки с уточненным состоянием 07, I0, II, I7 20 или 2I трансляция прекращается сразу.

Ошибки ввода-вывода выявляются при работе с внешними устройствами, когда устройства ввода-вывода работают со сбоями. При возникновении ошибки ввода-вывода индицируется строка, содержащая оператор ввода-вывода, сообщение "ERR 80" и две шестнадцатиричные цифры, указывающие уточненное состояние. Уточненное состояние указывает причину сбоя устройства.

2.3.2. Программообрабатываемые ошибки .

Большинство ошибок, выявляемых системой, приводят к прекращению выполнения программы (или строки непосредственного счета). Поэтому следует иметь средства управления такими ошибками без прерывания программы или нарушения программы выдачей сообщения об ошибке. Ошибки, возникающие во время счета по программе, которые позволяют продолжить программу при условии программной обработки и устранения ошибок, называются программообрабатываемыми ошибками.

К программообрабатываемым ошибкам относятся ошибки, возникающие при вычислении арифметических выражений, вычислении параметров операторов, при выполнении матричных операторов, выполнении операторов ввода-вывода.

К программообрабатываемым ошибкам не относятся системные ошибки, ошибки, связанные с переполнением памяти, ошибки, выявляемые при просмотре программы (ERR 22), ошибки, выявляемые при вводе данных с кла-

виатуры (ERR 29), ошибки, выявляемые при выполнении строки непосредственного счета.

2.3.3. Оператор обработки ошибок ON ERROR .

ON ERROR [<символьная переменная>,<символьная переменная>
<параметр ON ERROR> <номер строки>]

<Параметр ON ERROR> :: = GOTO / GOSUB/ THEN

Оператор ON ERROR предназначен для программной обработки ошибок, возникающих при счете по программе. Обычно система реагирует на ошибку остановом выполнения программы. Использование оператора ON ERROR позволяет продолжить выполнение программы без выдачи сообщения об ошибке. Номер полученной ошибки и номер строки, в которой произошла ошибка, соответственно заносятся в переменные, заданные в операторе, после чего производится переходит к строке, указанной в операторе. Длина обеих символьных переменных не должна быть меньше 4 байтов. Если длина объявлена меньше, то выдается сообщение об ошибке - ERR 59.

Если в качестве параметра заданы GOSUB или THEN при выполнении перехода к строке производятся следующие действия:

1) если в качестве параметра ON ERROR используется GOSUB, в магазине REN EXT запоминается адрес возврата к оператору, в котором обнаружена ошибка;

2) если в качестве параметра ON ERROR используется THEN , в магазине REN EXT запоминается адрес возврата к оператору, следующему за оператором, в котором обнаружена ошибка.

Использование параметров GOSUB и THEN позволяют пользователю оформлять обработку ошибок в виде подпрограмм. Кроме того, использование параметра GOSUB позволяет после обработки ошибки повторить тот оператор, в котором возникла ошибка. Использование параметра THEN позволяет после обработки ошибки продолжить программу с оператора, следующего за оператором, в котором возникла ошибка.

Оператор ON ERROR должен употребляться в программе ранее предполагаемого места появления ошибки. Каждый очередной оператор ON ERROR заменяет предыдущий, т.е. при обнаружении очередной ошибки будет осуществляться переход к строке с номером, указанным в последнем операторе ON ERROR.

При возникновении ошибок ввода-вывода в первые два байта первой символьной переменной заносится код ошибки 80, а в третий и четвертый байты - уточненное состояние. Во вторую символьную переменную заносится номер строки , где обнаружена ошибка (4 байта).

Оператор ON ERROR без параметров отменяет программную обработку ошибок. Примеры использования ON ERROR в программе:

1) в качестве параметра в операторе используется GOTO.

```
:10 ON ERROR  A%, B% GOTO100
:20 DATA 1, "ABC" : READ C% ,D
:40 ON ERROR
:50  GOTO 110
:60  END
:100 PRINT  A%, B%: GOTO40
```

При счете по данной программе возникает ошибка 43 (недопустимое присвоение), но при наличии оператора ON ERROR, счет по программе продолжается. Оператор ON ERROR в строке 40 отменяет дальнейшую программную обработку ошибок, поэтому следующая ошибка 11 (несуществующий номер строки) приводит систему к останову. В результате счета по данной программе имеем:

```
43    0020
50 GOTO 110
^ERR11
```

Если ошибка встречается в момент счета внутри цикла или подпрограммы, то после исполнения оператора ON ERROR возврат в цикл или подпрограмму возможен, если правильно подобран номер строки для перехода в операторе ON ERROR (внутри цикла или подпрограммы).

Например :

```
:10 ON ERROR  A%,  B% GOTO30
:15 FOR I=1 TO4: B =12/A:
:30 A=A+I
:35 PRINT  I,B
:40 NEXTI
```

При счете по данной программе имеем:

```
I     0
2     12
3     6
4     4
```

2) в качестве параметра используется GOSUB.

```
:10 DIM  Q%(16)
:20  ON ERROR  A%, B% GOSUB  50
:30 DATA LOAD BA  R(0) Q%( )
:40  HEXPRINT  Q%():END
:50 PRINT "НЕТ ДИСКА Р" :RETURN
```

При счете по данной программе, при отсутствии диска в кармане R, происходит зацикливание на строку 30.

```
Нет диска R
Нет диска R
.
.
.
Нет диска R
```

3) в качестве параметра используется THEN

```
:10 D = 24
:20  ON ERROR  A%, B%, THEN50
:30  C  = D/E
:40  PRINT  C,: GOTO60
:50  E=6:RETURN
:60 PRINT E
```

При счете по данной программе возникает ошибка ERR 03 (деление на нуль) и счет продолжается дальше. В результате имеем:

```
 0      6
```

2.4. Команды и операторы управления.

Команды языка БЭЙСИК обеспечивают пользователю средствами управления работой системы. Они не являются программируемыми и выполняются в режиме непосредственного счета.

2.4.1. CLEAR .

CLEAR [<параметр CLEAR>]

<Параметр CLEAR> :: = U /N/ P [<Диапазон строк>]

Оператор CLEAR предназначен для очистки памяти машины от ранее введенных программы и данных.

CLEAR без параметров стирает в памяти весь текст программы и значения всех переменных.

CLEAR U стирает все значения переменных (общих и необщих), но текст программы сохраняется в памяти.

CLEAR N стирает значения необщих переменных. Значения общих переменных и текст программы сохраняются.

CLEAR P без указания номеров строк стирает текст программы, описания и значения переменных сохраняются. Если номера строк заданы (например, CLEAR P10, 100), то стирается текст программы, начиная со строки с первым заданным номером до строки со вторым заданным номером включительно.

CLEAR P с первым указанным номером строки стирает часть прог-

раммы, начиная со строки, номер которой задан, и до конца программы. CLEAR P с указанным вторым номером, с предшествующей ему запятой стирает часть программы, начиная с начала и до указанной строки.

Примеры:

1) CLEAR
2) CLEAR V
3) CLEAR N
4) CLEAR P
5) CLEAP P10
6) CLEAP P10, 20
7) CLEAP P,100

2.4.2. CONTINUE .

Команда CONTINUE используется для того, чтобы продолжить выполнение программы после того как она была остановлена оператором STOP или командой HALT/STEP или в результате появления ошибки.

Выполнение программы продолжается с оператора, следующего за последним исполненным оператором или с оператора, на котором произошла ошибка в случае останова по ошибке.

Выполнение программы по команде CONTINUE не продолжается (выдается сообщение об ошибке), если:

1) произошло переполнение памяти;

2) была введена переменная, которая не определена в программе раньше;

3) были выполнены операторы CLEARV или CLEARN;

4) был изменен текст программы при выполнении операторов CLEAR, CLEAP P, RENUMBER или вводом новой строки программы;

5) выполнена команда RESET.

При останове программы по ошибке выполнение ее может быть продолжено только при условии устранения ошибки во время останова, если это возможно.

2.4.3. HALT/STEP .

Команда HALT/STEP используется для останова выполнения программы и перехода в режим непосредственного счета, чтобы затем можно было совершить пошаговое исполнение программы от одного оператора до другого и/или продолжить выполнение программы.

По команде HALT/STEP осуществляется следующее:

1) если идет исполнение программы, по команде HALT/STEP ее исполнение останавливается после того, как закончено исполнение

текущего оператора. Исполнение программы, начиная со следующего оператора может быть продолжено по команде CONTINUE ;

2) если выполнение программы было остановлено по оператору STOP или по команде HALT/STEP, повторная команда HALT/STEP вызывает индикацию следующего по программе оператора и его исполнение, после чего исполнение программы опять останавливается. Каждый раз по команде HALT/STEP индицируется следующий оператор и исполняется. Команда HALT/STEP также может быть использована для пошагового исполнения программы в режиме слежения, если, например, нужно проверить значения переменных (см. TRACE пункт 2.4.I2).

Выполнение программы в пошаговом режиме не продолжается, если:

I) произошло переполнение памяти;

2) введена переменная, которая не была определена раньше;

3) выполнены операторы CLEARV или CLEARN ;

4) текст программы был изменен по операторам CLEAR,CLEAR P, RENUMBER или вводом новой строки программы;

5) выполнена команда RESET.

Пример:

```
.
.
:90 GOSUB 200
:IOOPRINT "X,Y"
:IIO X=I.2: Y=5*Z+X: GOTO 30
```

Оператор TRACE вводится для того, чтобы пользователь мог видеть какие переменные принимают новые значения. Для того чтобы просмотреть выполнение программы по шагам с IOO строки необходимо:

I) установить режим слежения	: TRACE
2) перейти к строке IOO	:GOTO IOO
3) ввести команду HALT/STEP	IIO:X=I.2:Y=5*Z+X :GOTO 30
	X=I.2
4) ввести команду HALT/STEP	IIO: Y=5*Z+X :GOTO30
	Y=2I.6
5) ввести команду HALT/STEP	IIO: GOTO30
	TRANSFER TO 30

2.4.4. LIST .

LIST [<символ S>] [<устройство>]

LIST [<символ S>] [<устройство>,] [<диапазон строк>]

<диапазон строк> :: =[<номер строки>],<номер строки>/<номер строки>[,<номер строки>]

Оператор LIST предназначен для вывода (листинга) текста программы, находящейся в памяти машины в порядке возрастания строк на указанное устройство.

Если в операторе LIST не указан диапазон строк, то выдается листинг всей программы: если программа занимает более 24 строк, то после выдачи всего текста на экране остаются последние 23 строки программы.

Если в операторе LIST задан только один номер строки (например, LIST I0), то индицируется только одна строка.

Если в операторе LIST задан диапазон строк (например, LIST I0, 40), то индицируется часть программы в этом диапазоне строк, включая заданные строки.

Если в операторе LIST задан только второй номер строки (например, LIST ,20), то индицируется часть программы от начала до заданной строки.

Наличие параметра S в операторе дает возможность получить на дисплее поэкранный листинг программы по 23 строки. После выдачи на экран 23 строк производится останов и для просмотра следующих 23 строк надо нажать клавишу CR/LF.

Чтобы исключить нажатие клавиши CR/LF после просмотра одного экрана, используется оператор SELECT P (см.п. 2.2.3) для задания паузы при индикации программы на экране (от I/6 до I.5 с). Наличие паузы дает возможность просмотреть всю программу по мере того как она медленно, строка за строкой, проходит по экрану.

Все сказанное о параметре S относится и к другим формам оператора LIST.

2.4.5. LIST'.

LIST [<символ S>][<устройство>,]<символ'> [<номер DEFFN'>]

Оператор LIST' осуществляет вывод на заданное устройство всех номеров строк, содержащих операторы GOSUB' и DEFFN'.

Если в операторе указан параметр <номер DEFFN'>, то осуществляется вывод номеров строк, содержащих операторы GOSUB' и номер строки, содержащей DEFFN' с указанным номером подпрограммы. Если подпрограмма, к которой обращаются, не существует, система выдает четыре вопросительных знака (????) вместо номера строки.

Примеры:

1) LIST /05,'

0101 - DEFFN'2 индицируется номер строки, в которой упоминается оператор DEFFN', а ниже - номер
0145 строки, где есть обращение к этой подпрограмме

по оператору GOSUB'

2) LIST' I5

Если подпрограммы нет, а обращение к ней GOSUB' I5 есть, то на экране индицируется:

???? DEFFN' I5
0020

3) LIST' I0
0695-DEFFN'I0

Если подпрограмма DEFFN'I0 есть, а обращения к ней нет, на экране дисплея индицируется только номер строки, содержащей оператор DEFFN'I0.

4) :I0 DEFFN'I5 "ТЕКСТ"
:20 GOSUB' 0 : GOSUB' 0I
.
.
.
:200 DEFFN'00
:2I0 GOSUB' I
.
.
.
:900 DEFFN'I
:9I0 GOSUB' 2

При исполнении оператора LIST' получаем следующее:

: LIST'
0200 DEFFN'0
0020
0900 DEFFN'I
0020 02I0
???? DEFFN'2
09I0
00I0 DEFFN'I5

При исполнении оператора LIST' 0I получаем следующее:

: LIST' 0I
0900 DEFFN'I
0020 02I0

2.4.5. LIST #.

LIST [<символ S>][<устройство>,]#[<диапазон строк>]

<диапазон строк> :: = [<номер строки>],<номер строки> / <номер строки>[,<номер строки>]

Оператор LIST# осуществляет вывод на заданное устройство номеров строк, к которым есть переходы (по операторам IF THEN ON ERROR, ON GOTO, GOSUB, KEY(N и др.).

Поиск таких операторов производится по всему тексту программы.

Диапазон строк определяет номера строк, поиск переходов к которым производится по программе. Если в операторе задан диапазон строк, то осуществляется вывод номеров строк, содержащих переходы к строкам программы указанного диапазона.

Если в операторе задан один номер строки, то осуществляется вывод номеров строк, содержащих переходы к строкам программы, начиная с указанного номера до номера последней строки программы.

Если первый номер строки в диапазоне строк опущен, то осуществляется вывод номеров строк, содержащих переходы к строкам программы, начиная с номера первой строки программы и кончая указанным номером строки.

Номера строк, к которым происходит обращение, но которые не существуют, заключаются в круглые скобки при индикации.

Примеры:

1) LIST#205

   0205 - 0175 0200 0230 0255 на экране индицируются номера строк, в которых упоминается строка 205.

2)
```
:10 GOTO 100
:20 IF I=3 THEN 90: IF I=4 THEN100
:30  GOSUB 200
:40  KEYIN A#,50, 300: GOTO 40
:50 PRINT  A#: GOTO 40
:90 X=2.7
:100 Y =Z/X
:110 GOTO 500
.
.
.
:200 REM ПРОЦЕСС
.
.
.
:290 RETURN
:300 END
```

По оператору LIST имеем следующую информацию:

```
: LIST#
0040  - 0040 0050
0050  - 0040
0090  - 0020
0100  - 0010 0020
```

0200 - 0030
0300 - 0040
(0500) - 0110
По оператору LIST¤ , 40 имеем следующую информацию:
: LIST¤, 40
0040 - 0040 0050
По оператору LIST¤ 50 имеем следующую информацию:
: LIST¤ 50
0050 - 0040
0090 - 0020
0100 - 0010 0020
0200 - 0030
0300 - 0040
(0500) - 0110
По оператору LIST¤ 90, 250 имеем следующую информацию:
: LIST¤ 90,250
0090 - 0020
0100 - 0010 0020
0200 - 0030

2.4.7. LIST V .

LIST [<символ S>][<устройство>,]V[<диапазон переменных>]

<Диапазон переменных> :: =[<простая числовая переменная>,]
<простая числовая переменная> / [<простая символьная переменная>,]
<простая символьная переменная> /[<метка массива>,]<метка массива>

Оператор LIST V осуществляет вывод номеров строк, содержащих все переменные программы. Если в операторе LIST V указан диапазон переменных или одна переменная, то осуществляется вывод номеров строк, содержащих указанную переменную или указанный диапазон (т.е. всех переменных заданного типа, начиная с первой заданной и до второй).

Примеры:

1) LIST V на экране индицируются все номера строк, в которых упоминаются переменные, используемые в программе:

A% - 0500 0560
X% - 0545 0580 0960 1100
A - 0160
N - 0965
B¤ - 1145

2) :10 DIM A(3), B¤ 80, N(40)
:20 AI=5: X,XI,Y=0.3
:30 A(I)=AI*Y/2
:40 B¤=D¤(I)

Указание всех переменных для данной программы:

```
:LIST V
A( ) - 0010 - 0030
AI   - 0020 0030
B¤   - 0010 0040
N( )  - 0010
X     - 0020
XI    - 0020
Y     - 0020 0030
D¤()  - 0040
```

Указание единственной переменной:

```
:LIST V A1
0020 0030
```

Указание диапазона простых числовых переменных:

```
:LIST V AI, Y
AI - 0020 0030
X  - 0020
XI - 0020
Y  - 0020
```

Указание диапазона простых символьных переменных:

```
:LIST V B¤,D¤
B¤() - 0010 0040
D¤() - 0040
```

2.4.8. LIST *

LIST [<символ $>] [<устройство>,] * <символьный аргумент> [<диапазон строк>]

Оператор LIST* обеспечивает возможность вывода программных строк, содержащих контекст, указанный в символьном аргументе.

Если диапазон строк не указан в операторе, поиск контекста производится по всей программе; если диапазон строк указан – то в заданном диапазоне строк.

Если указана только одна строка (например, LIST* 20), то поиск осуществляется в диапазоне от строки с указанным номером до последней строки программы.

Если указан только второй номер строки в диапазоне (например, LIST*,20), то поиск осуществляется от начала программы до строки

с заданным номером.

Программные строки, содержащие контекст, выводятся на устройство, определяемое оператором SELECT LIST или на устройство, непосредственно заданное в операторе LIST. Концевые пробелы, содержащиеся в контексте, в поиске не участвуют.

Пример:

```
: I0  A$= "ABCDEF": PRINT A$
:20  B$= STR ( A$,3, 3,)
: LIST *= "PRINT"
: RUN
I0  A$ ="ABCDKP": PRINT  A$
```

2.4.9. RENUMBER .

RENUMBER [<номер строки>] [,<новый номер>]

<Новый номер> :: = <номер строки>, <шаг> /<номер строки>/,<шаг>

<шаг> :: =<цифра> / <цифра><цифра>

Оператор RENUMBER предназначен для перенумерации строк программы и ссылок на них. Все строки программы нумеруются заново, начиная со строки, номер которой задан первым параметром <номер строки>. При его отсутствии перенумеровывается вся программа.

Параметр <новый номер строки> присваивается строке, с которой начинается перенумерация. Если он не задан, то принимается равным третьему параметру.

Параметр <шаг> определяет шаг возрастания новой нумерации. Если его значение не задано, то он принимается равным I0.

Оператор RENUMBER без параметров перенумеровывает всю программу начиная с номера I0, через I0 номеров строк, т.е. I0,20,30 .

Все обращения к номерам строк в операторах, например, в GOTO, IF THEN и т.п. перенумеровываются в соответствии с новыми номерами.

Примеры:

1) RENUMBER

2) RENUMBER 30,I20,20 строка 30 получает номер I20 и осуществляется дальнейшая перенумерация строк программы с шагом 20.

3) RENUMBER , 5,5 первая строка программы получает номер 5 и осуществляется перенумерация строк программы с шагом 5.

4) RENUMBER 5 строка 5 получает номер I0 и осуществляется дальнейшая перенумерация строк программы с шагом I0.

5) RENUMBER ,,5 — то же, что и RENUMBER , 5,5

6) RENUMBER I2,,20 — то же, что и RENUMBER I2,20,20

7) RENUMBER I0,I5 — строка I0 получает номер I5 и осуществляется дальнейшая перенумерация строк программы с шагом I0.

2.4.I0. RESET .

Команда RESET используется для немедленного прекращения выполнения программы или строки непосредственного счета. При этом очищается экран дисплея, все устройства ввода-вывода устанавливаются в исходное положение и управление передается на клавиатуру. Текст программы в памяти не стирается и все переменные сохраняют свои текущие значения. Режим слежения и паузы по команде RESET отключаются.

После выполнения команды RESET программа не может быть продолжена, а должна быть запущена заново. Поэтому RESET должен рассматриваться как крайняя мера при необходимости остановить выполнение программы. Команду RESET не следует использовать для останова выполнения во время записи на диск, поскольку в этом случае нельзя гарантировать сохранность данных на диске.

Если машина производит неправильные действия и не реагирует на команду RESET, пользователь должен нажать кнопку сброса на лицевой стороне дисплея для выхода из БЭЙСИК-системы.

2.4.II. RUN .

RUN [<номер строки>] .

Оператор RUN предназначен для запуска программы на счет.

По оператору RUN без параметров осуществляется следующее:

I) проверка формальной правильности загруженной программы, т.е. проверка всех обращений к переменным. После просмотра программы отводится зона значений для всех переменных программы и устанавливаются начальные значения: числовые переменные принимают значение нуль, а символьные переменные – пробелов (общие переменные сохраняют свои значения).

2) указатель текущего значения данных оператора READ возвращается к первому значению (в первом операторе DATA программы).

Затем программа выполняется, начиная со строки с наименьшим номером.

По оператору RUN с параметром <номер строки> исполнение программы начинается с указанной строки программы. Значения переменных сохраняются (полученные при последнем выполнении программы).

С помощью оператора RUN <номер строки> можно продолжать счет по прерванной программе, но ее выполнение не должно начинаться внутри цикла или подпрограммы.

Примеры:

1) RUN

2) :10 RUN 30

2.4.12. TRACE.

TRACE [OFF] .

Оператор TRACE предназначен для пооператорной отладки программы.

По оператору TRACE пользователь имеет возможность проследить за ходом выполнения программы. Режим слежения устанавливается после выполнения оператора TRACE и заканчивается после выполнения оператора TRACE OFF.

В режиме слежения TRACE на консоль вывода (CO) выводится текст строки и результаты (имя переменной и ее значение), когда переменной в ходе выполнения программы присваивается новое значение (операторами LET, READ, FOR и т.д.).

Если встречается оператор ветвления программы (GOTO, GOSUB, IF) на консоль CO выводится информация:

TRANSFER TO <номер строки>

В режиме слежения можно использовать команду HALT/STEP для получения более полной картины при пошаговой реализации программы.

Если TRACE имеет вывод на дисплей, то этот процесс может быть замедлен до скорости, удобной для чтения, при помощи организации пауз оператором SELECT P (см.п. 2.2.3).

Если по оператору TRACE осуществляется на дисплей вывод информации, содержащей управляющие символы (например НЕХ(03)), то коды, не имеющие графического обозначения заменяются точками на экране дисплея.

Примеры:

1) :40 TRACE
   :80 GOTO 200
   :200 REM

На дисплее индицируется: TRANSFER TO 200

2) :10 TRACE
   :20 DIM Z(5)
   :30 A=2
   :40 B=2

```
    :50 C = 2
    :60 X,Y, Z(5)=A  + SIN(B)/
```

В результате выполнения програ... на дисплее индицируется:

```
A = 2
B = 2
C = 2
X = 2.4546487134I3
Y = 2.4546487134I3
Z()= 2.4546487134I3
```

3)
```
    :5 DIMX(3)
    :7 TRACE
    :I0 FOR I=1 TO 3
    :20 PRINT X(I)
    :30 NEXT I
```

В результате на дисплее индицируется:

```
I = 1
0
NEXT I=2
0
NEXT I=3
0
NEXT END I
```

4)
```
    :I0 DIM A(4)
    :20 A(I) =24.2: A(2)=25.36; A(3)=48.00I: A(4)=I4.759
    :30 FOR  I=1 TO 4
    :40 TRACE
    :50 X=X+A(I)
    :60 TRACE OFF
    :70 NEXT I
```

В результате выполнения программы на дисплее индицируется:

```
X = 24.2
X = 49.56
X = 97.56I
X = II2.32
```

2.4.13. Ключи специальных функций.

На клавиатуре имеются шестнадцать клавиш специальных функций. Вместе с клавишей SHIFT эти шестнадцать клавиш обеспечивают доступ с клавиатуры к 32 подпрограммам или к определениям ввода текста.

Когда нажимается клавиша специальных функций (СФ), в программе отыскивается соответствующий оператор DEFFN'. Если оператор DEFFN' указывает подпрограмму, то начинается выполнение подпрограммы. Если оператор DEFFN' используется для определения строки символов при вводе текста, то указанный текст вводится в буфер ввода. Таким образом, нажатие клавиши СФ2 вызывает исполнение подпрограммы DEFFN'2 или ввод строки текста, которая соответствует DEFFN'2.

По клавише СФ может быть запущена программа, но только после того, как она была выполнена по команде RUN.

Например, клавиша СФ2 обозначена в операторе:

```
:100 DEFFN'2 "HEX("
```

Нажатие клавиши СФ2 после того, как было введено с клавиатуры следующее:

```
:20 PRINT
```

дает результат:

```
:20 PRINT HEX(
```

Если клавиша СФ предназначена для обращения к помеченной подпрограмме, то подпрограмма может быть выполнена по нажатии соответствующей клавиши СФ. После выполнения оператора RETURN управление передается на клавиатуру.

Клавиши СФ могут быть использованы вместе с обозначенными подпрограммами (операторы DEFFN') для того, чтобы обеспечить число точек ввода, с которых начинается выполнение программы так же как и для того, чтобы осуществлять переход на обозначенные точки в пределах выполняющейся программы. Поскольку по клавише СФ в магазине циклов и подпрограмм записывается информация о возврате, неоднократное использование клавиши СФ возможно, если удовлетворяется одно из следующих условий:

1) для уничтожения информации о возвратах используется оператор RETURN CLEAR;

2) используется команда RESET;

3) выполнение подпрограммы заканчивается оператором RETURN, который передает управление на клавиатуру и стирает информацию из магазина.

Несоблюдение одной из этих предосторожностей приводит в конечном результате к переполнению памяти.

2.5. Основные операторы.

Оператор языка БЭЙСИК является программируемым элементом, который служит основой построения программы на языке БЭЙСИК. В этом пункте излагается назначение основных операторов языка БЭЙСИК 02 и синтаксис каждого из них.

2.5.1. COM.

COM <список объявлений>

<список объявлений> :: = <объявление> / <список объявлений>, <объявление>

<объявление> :: = <объявление простой переменной> / <объявление массива>

<объявление простой переменной> :: = <простая числовая переменная> / <простая символьная переменная> [<длина>]

<длина> :: = <цифра> / <цифра><цифра> / <цифра><цифра><цифра>

<объявление массива> :: = <объявление целого одномерного массива> / <объявление целого двумерного массива> / <объявление действительного одномерного массива> / <объявление действительного двумерного массива> / <объявление символьного одномерного массива> / <объявление символьного двумерного массива>

<объявление целого одномерного массива> :: = <идентификатор переменной> % (<размерность>)

<объявление целого двумерного массива> :: = <идентификатор переменной> % (<размерность>, <размерность>)

<объявление действительного одномерного массива> :: = <идентификатор переменной> (<размерность>)

<объявление действительного двумерного массива> :: = <идентификатор переменной> (<размерность>, <размерность>)

<объявление символьного одномерного массива> :: = <идентификатор переменной> $ (<размерность>)

<объявление символьного двумерного массива> :: = <идентификатор переменной> $ (<размерность>, <размерность>)

<размерность> :: = <цифра> / <цифра><цифра> / <цифра><цифра><цифра> / <цифра><цифра><цифра><цифра> / <цифра><цифра><цифра><цифра><цифра>

Оператор COM позволяет определить переменные любого типа как общие. Оператором COM переменные могут быть объявлены общими для нескольких программ или сегментов программы, может быть зарезервировано место в памяти системы для массивов и задана длина символьных переменных и/или элементов символьного массива. Оператор COM должен выполняться в программе до использования любых переменных и массивов.

Длина общих переменных и размерность общих массивов должны быть идентичны во всех программах или сегментах, имеющих общие области. При наличии общих переменных, которые используются в нескольких последовательно исполняющихся программах, операторы COM не обязательно должны присутствовать в любой из этих программ, кроме первой. Чтобы объявить новые переменные общими необходимо использовать оператор COM в очередном сегменте.

В результате, когда программы или необщие переменные стираются из памяти (при вводе новых сегментов или программ в машину), эти переменные остаются нетронутыми. Значения общих переменных стираются только операторами CLEAR, CLEARV.

Примеры:

I) COM A (I0) , B (3,3) C2

2) COM A, B*2,C(2,3), B*(4,5) 6

2.5.2. COM CLEAR .

COM CLEAR [<параметр COM CLEAR>]

<параметр COM CLEAR> :: =<числовая переменная> / <символьный элемент>

<числовая переменная> :: =<числовой элемент> /<метка числового массива>

<числовой элемент> :: =<простая числовая переменная> /<элемент числового массива>

<простая числовая переменная>::= <идентификатор переменной>[%]

<элемент числового массива> ::= <идентификатор переменной>[%] (<AB>) / <идентификатор переменной> [%] (<AB>, <AB>)

<символьный элемент> :: =<простая символьная переменная> / <элемент символьного массива> /<метка символьного массива>

<простая символьная переменная> :: = <идентификатор переменной> * <элемент символьного массива> :: = <идентификатор переменной> * (<AB>)/ <идентификатор переменной>*(<AB>, <AB>)

<метка символьного массива> :: = <идентификатор переменной>(*)

Оператор COM, CLEAR предназначен для переопределения всех общих переменных (массивов) или части этих переменных (массивов) в необщие переменные (массивы) и наоборот.

Если в операторе COM CLEAR задан параметр, то переменные (массивы), ранее определенные оператором COM как общие и стоящие в операторе COM до параметра, указанного в операторе COM CLEAR, остаются общими переменными (массивами), а переменные (массивы), стоящие в операторе COM после параметра, указанного

в операторе COM CLEAR становятся необщими.

Оператор COM CLEAR без параметров, переводит все общие переменные (массивы) в необщие. Если в операторе COM CLEAR задан параметр, который не был определен ранее оператором COM, как общий, то переменные (массивы), используемые в программе до указанной переменной (массива), становятся общими, а сама переменная(массив) и все переменные (массивы), определенные за ней остаются необщими.

Примеры:

1) COM CLEAR — все переменные становятся необщими

2):10 COM A,B,X,У(10) — A и B остаются общими,
:200 COM CLEAR X — X и У становятся необщими

3):10 DIM X (10,10)
:20 A =B+C — массив X и переменная A стали общими,
:30 COM CLEAR B — B и C остаются необщими

2.5.3. DATA .

DATA <список DATA>

<Список DATA> :: = <элемент DATA> /<список DATA>,<элемент DATA>

<элемент DATA> :: = <цифровая константа> /<цифровой элемент>/ <символьная константа> / <символьний операнд>

Оператор DATA предназначен для указания констант и переменных, значения которых в процессе выполнения программы присваиваются переменным оператора READ (см. п. 2.15,19).

Константи и переменные, указанные оператором DATA, должны быть упорядочены согласно последовательности их использования в операторе READ .

Числовые константы и числовые переменные в операторе DATA должны соответствовать числовым переменным в операторе READ, а символьные константы и символьные переменные – символьным переменным.

Если программа содержит несколько операторов DATA, то они образуют список в соответствии с номерами строк.

Оператор RESTORE (см.п. 2.5.22) обеспечивает возможность использования данных оператора DATA более одного раза в течение выполнения программы.

Оператор DATA не используется в режиме непосредственного счета.

Примеры:

1) : 10 DATA 3,4,5,6

2) : 10 DATA 17.3, -23.9, 1.3E-4

3) :20 DATA ,HEX (4I42), "ABC"

2.5.4. DEFFN .

DEFFN <имя FN>(<простая числовая переменная>)= <AB>
<имя FN>:: = <цифра>/<буква>

Оператор DEFFNпредназначен для определения функций пользователя FN в пределах программы.

Параметр <имя FN> используется для обозначения определения функции и может задаваться любой цифрой (0–9) или буквой (A – Z), всего 36 наименований.

Параметр <простая числовая переменная> представляет собой формальный аргумент, при обращении к функции FN формальный аргумент заменяют фактическим.

Параметр <AB> служит для определения новой функции и может содержать формальный аргумент.

Обращение к функции может быть совершено из любой точки программы с помощью функции FN .

Функция FN должна иметь два параметра: букву или число, которое представляет имя функции и аргумент, чье значение должно быть передано функции для вычисления. Аргумент, обозначенный в функции FN , может быть любым числовым выражением.

Значение аргумента, полученного в результате вычисления выражения, обозначенного в функции FN , автоматически передается оператору DEFFN (формальный аргумент) для вычисления и результат возвращается функции FN . Функция, определенная пользователем, может быть использована в любом месте программы, где числовые функции системы являются допустимыми.

Буква или цифра, обозначенная как имя функции используется только для того, чтобы обозначить определенную функцию, когда происходит обращение к ней с помощью функции FN . Переменная, обозначенная как формальный аргумент, используется только как держатель места, указывая, где в выражении оператора DEFFN используется значение аргумента функции FN.

Пример:

```
I) :30 C=3
   :40 DEFFN  A(X) =X^2-X
   .
   .
   .
   :80 PRINT  FN A( C*2)
   :90 END
```

```
:RUN
 30
```

При выполнении программы сначала вычисляется выражение C*2 (строка 80). Результат 6 является значением аргумента, которое передается функции, обозначенной "A", формальный аргумент X в функции A (строка 40) заменяется на значение аргумента ,6. DEFFN A(6)=6^2-6=30. Результат 30, затем возвращается к функции пользователя FN , присваивается переменной и печатается.

Функция, определяемая оператором DEFFN , не может быть выражена через саму себя, но для ее определения может быть использована функция, определяемая другим оператором DEFFN,например:

```
I) :10 DEFFN I(A)=A ^ 2-A
   :20 DEFFN 2(A)=A+FN I(A)
```

Две функции не могут соотноситься друг с другом (производя в действительности бесконечный цикл), например,следующая пара операторов является недопустимой:

```
I):10 DEFFN I(A)=FN 2(A)
  :20 DEFFN2 (A)=FN I(A)
```

Оператор DEFFNне исполняется, когда он встречается в программе. Его исполнение управляется исключительно обращением на функции FN . По этой причине оператор DEFFN может располагаться где угодно в программе.

Примеры:

```
1) :60 DEFFN A (Z) =Z^2-2
2) :50 DEFFN A ( C)= ( 3*A)-8/C+FN B(2-A)
3) :100 DEFFN B (D)= COS (0)+X/3
```

2.5.5. DEFFN' .

'DEFFN'<номер DEFFN'>[<список для присвоения>]/
DEFFN'<номер DEFFN'><список символьных констант>
<номер DEFFN'>::= <цифра>/<цифра> <цифра>/<цифра> <цифра> <цифра>
<список для присвоения>::=<числовой элемент>/<символьный операнд>/
<список для присвоения> , <числовой элемент> /
<список для присвоения> , <символьный операнд>
<список символьных констант> :: = <символьная константа>/
<список символьных констант> ,<символьная константа>
<символьная константа> :: ='<цепочка символов>'/
"<цепочка символов>"

HEX (<цепочка элементов HEX>)

<цепочка элементов HEX> :: = <элемент HEX>/

<цепочка элементов HEX>, <элемент HEX>

<элемент HEX>:: =<шестнадцатиричная цифра><шестнадцатиричная цифра>

<шестнадцатиричная цифра> ::= 0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F

Оператор DEFFN' предназначен для:

I) ввода часто встречающихся текстов посредством нажатия одной из клавиш специальных функций:

2) организации помеченных подпрограмм.

Вместо того, чтобы каждый раз набирать на клавиатуре часто встречающиеся в ходе программы слова и выражения, можно запоминать их один раз и с помощью оператора DEFFN'(первая форма) ввести в память машины, а затем вводить в программу простым нажатием клавиш СФ.

<номер DEFFN'> в этом случае должен быть от 0 до 31 и представлять одну из клавиш СФ. Например:

в программе часто встречается текст "HEX(" . Данный символ будет вводиться клавишей СФ-I2. Текст программы в этом случае имеет вид:

```
:I0 DEFFN'I2 "HEX("
```

теперь, чтобы получить строку:

```
:40 PRINT HEX(
```

нужно ввести следующее:

```
:40 PRINT и нажать I2  клавишу СФ
```

С помощью клавиши СФ удобно вводить часто используемые операторы (операторы преобразования данных, матричные операторы, операторы сортировки), для которых нет отдельных клавиш на клавиатуре.

В одном операторе DEFFN' может быть использовано несколько символьных констант, которые отделяются друг от друга запятой, например:

```
:I0 DEFFN'5"ПРОГРАММА"; HEX (85);"Конец"
.
.
.
:I00  PRINT 5
```

Код HEX(85) воспринимается системой как CR/LF и означает конец ввода . Символы, которые следуют за HEX(85) в операторе DEFFN' игнорируются.

Оператор DEFFN′ (вторая форма) указывает начало помеченной подпрограммы. Подпрограмма может быть вызвана в программе оператором GOSUB′(см.п. 2.5.IO) или с клавиатуры нажатием соответствующей клавиши СФ.

Если обращение к подпрограмме осуществляется оператором GOSUB′, то номер DEFFN′может быть любым числом от 0 до 255, если обращение к подпрограмме осуществлено с клавиатуры, то номер DEFFN′может быть от 0 до 3I.

После нажатия клавиши СФ, либо выполнения оператора GOSUB′ происходит переход к оператору DEFFN′ с номером, соответствующим номеру в операторе GOSUB′или номеру СФ, после чего подпрограмма выполняется, начиная с соответствующего оператора DEFFN′.Например если нажата клавиша 2 СФ, исполнение программы начинается с оператора DEFFN′2.

По оператору RETURNуправление передается оператору программы, следующему за оператором GOSUB′ или клавиатуре, если обращение к подпрограмме было осуществлено нажатием клавиши СФ.

Оператор DEFFN′ может иметь список переменных (формальных параметров). Переменные в списке переменных получают значения аргументов, переданных подпрограмме из оператора GOSUB′.

Если число аргументов, переданных подпрограмме, не равно числу переменных оператора DEFFN′ или, если значения и переменные не соответствуют по типу (числовые, символьные), то выдается сообщение об ошибке. По клавише СФ может быть осуществлено обращение только к подпрограмме, не имеющей параметров.

Примеры:

```
I):IO  GOSUB′ 7 (AI, Z ,K)
  :
  :40 DEFFN′7(X,A,B)
  :50 X=A+B
  :60 RETURN
```

2.5.6. DIM

DIM <список объявлений>

<список объявлений> :: = <объявление>/<список объявлений>, <объявление>

<объявление> :: = <объявление простой переменной>/<объявление массива>

<объявление простой переменной>:: =<простая числовая переменная>/<простая символьная переменная> [<длина>]

<длина> :: = <цифра> /<цифра><цифра> / <цифра><цифра><цифра>

<объявление массива> ::= <объявление целого одномерного массива>/ <объявление целого двумерного массива> / <объявление действительного одномерного массива > / <объявление действительного двумерного массива > /<объявление символьного одномерного массива > /<объявление символьного двумерного массива >

Оператор DIM предназначен для резервирования места в памяти машины для переменных и массивов, к которым должно быть обращение в программе, а также для задания длины символьных переменных и элементов символьных массивов. Одним оператором DIM можно зарезервировать в памяти место для нескольких массивов и символьных переменных. Оператор DIM должен находиться в программе перед первым обращением к переменным и массивам.

Размерности массивов и длин символьных переменных или элементов массива должны обозначаться целыми положительными числами (ноль не разрешается). Если длина символьной переменной не задается в операторе DIM, то она принимается равной 16 байтам. Задавать длину для числовых переменных не требуется.

Если переменная или массив, встречающиеся в программе, не определены оператором DIM (или COM), то система автоматически резервирует место в памяти (для символьных переменных 16 байтов, для массивов 10 * 10). Значения переменных и массивов , определенных оператором DIM, стираются из памяти после ввода новой программы (или сегмента) или когда текущая программа запускается снова.

Примеры:

1) DIM A(45) - резервирование места в памяти для одномерного массива из 45 чисел.

2) DIM B(8,10) - резервирование места в памяти для двумерного массива из 80 чисел

3) DIM K$*35 - задание длины символьной переменной в 35 символов

4) DIM A$*(4,4)10 - резервирование места в памяти для двумерного массива 4 * 4 с длиной каждого элемента 10 символов

5) DIM B(80), L(3),M(8,7) - резервирование места в памяти для двух одномерных и одного двумерного массива

2.5.7. END .

Оператор END обозначает конец программы.

После выполнения оператора END счет по программе прекращается и на дисплее высвечивается:

END PROGRAM

FREESPACE = ..., где ..... - количество байтов свободной

памяти.

Использование оператора END в конце текста программы не обязательно, так как выполнение программы автоматически заканчивается после выполнения всех ее строк. В программе может быть более, чем один оператор END.

Оператор END может быть использован в режиме непосредственного счета, что позволяет в любой момент времени индицировать объем свободной памяти. В этом случае учитывается длина текста программ и не учитывается размер зоны значений, объявленных операторами DIM и COM, поскольку отведение памяти для переменных и массивов происходит во время реализации программы. Свободная память после ввода программы до счета на ней будет больше, чем после счета.

Пример:

```
:10  X=24-2*4
:20 PRINT X
:30 END
```

2.5.8. FOR .

FOR <простая числовая переменная> = <АВ> TO <АВ>[STEP <АВ>] .

Оператор FOR совместно с оператором NEXT предназначен для организации цикла. Оператор FOR фиксирует начало цикла и определяет параметры цикла. NEXT фиксирует конец цикла (см.п. 2.5.17). Операторы программы, расположенные между FOR и NEXT исполняются повторно до тех пор, пока не достигнуто значение выхода из цикла.

Цикл состоит их трех логически связанных частей:заголовка цикла, тела цикла и конца цикла.

Заголовок цикла определяется оператором FOR. Первый параметр <АВ> используется для задания начального значения числовой переменной, второй параметр <АВ> определяет конечное значение заданной величины, параметр STEP задает шаг изменения заданной величины. Значение STEP может быть задано произвольно и оно определяет число, на которое заданная величина должна увеличиваться или уменьшаться по мере каждого выполнения цикла. Значение STEP может быть положительным или отрицательным. Если параметр STEP не указан, то он считается равным "+1". Когда начальное значение цифровой переменной достигает конечного значения числовой переменной или превышает его, цикл останавливается и начинается нормальная последовательность исполнения программы, начиная с оператора, который следует за NEXT. Если шаг определен так, что достигнуть конечного значения нельзя, цикл выполняется только один раз, например:

```
:50 FOR  I=1  TO5STEP  -I
```

Тело цикла представляет собой последовательность операторов, которая, в частности, может содержать также и операторы другого цикла, и которая выполняется для каждого очередного изменения на шаг заданной величины. Циклы могут вкладываться друг в друга, число вложений ограничивается свободной памятью машины.

Существуют несколько специальных условий, относящихся к применению циклов:

I) недопустимо входить в середину цикла из любого места программы (например, с помощью операторов GOTO, GOSUB, IF THEN). Поскольку параметры цикла определены оператором FOR, то исполнение оператора NEXT, для которого не был выполнен соответствующий оператор FOR, завершится ошибкой (ERR 25);

2) недопустимо выходить из цикла до того, как он закончен. Если это часто повторяется в программе, то в результате выдается ошибка (ERR 0I);

3) недопустимо выходить из цикла по оператору RETURN. Исполнение оператора RETURN автоматически стирает все параметры цикла из памяти. Но, если обращение к циклу производится внутри подпрограммы, выход из цикла возможен до его завершения по оператору RETURN. Выполнение цикла при этом производится от заголовка цикла до оператора RETURN один раз (см. пример 5);

4) цикл со значением STEP равным нулю исполняется только один раз;

5)при непосредственном счете весь цикл (от FOR до NEXT) должен помещаться в одной строке непосредственного счета (см.пример 2).

Примеры:

```
I) :40  FOR A=5 TO 50 STEP 5
   :50 PRINT  A, LOG (A)
   :60 NEXT A
2) FOR  X=0  TO I0: PRINT  TAB(X); "+" : NEXT  X
3) :I0 FOR A=I TO4
   :20 FOR B=I TO6
   :30 IF  D(A)<10 THEN 50
   :40 NEXT  B
   :50 NEXT  A
4) :I0 FOR A=I TO 50
   :20 FOR   C=-I TO I0
   :30 .....
   :40 NEXT C
   :70 FOR B=I0 TO I00 STEP 2
```

```
    .
    .
    :100 GOTO 190
    .
    .
    .
    :150 NEXT B
    .
    .
    .
    :190NEXT A
5):80 GOSUB 100
   :90 .....
   :100  FOR X=.1 TOZSTEP .05
   :140  IF A (У) < 3.25 THEN200
    .
    .
    .
   :190 NEXT X
   :200 RETURN
```

2.5.9. GOSUB

GOSUB <номер строки>

Оператор GOSUB используется для осуществления входа в подпрограмму. Вход в подпрограмму производится по первой строке подпрограммы, номер которой указан в операторе GOSUB. Эта строка может содержать любой оператор языка БЭЙСИК. Последним выполняемым оператором подпрограммы должен быть оператор RETURN (см. п. 2.5.23), который возвращает исполнение основной программы оператору, следующему сразу за оператором GOSUB. Возвращение назад к основной программе осуществляется сразу же после того, как встречается оператор RETURN. Операторы, следующие за RETURNна этой же строке, не принимаются во внимание.

Оператор GOSUB может находиться внутри одной подпрограммы, вызывая другую подпрограмму. После выполнения оператора RETURN второй подпрограммы, происходит возвращение к первой подпрограмме, к оператору, следующему за оператором GOSUB . Такие подпрограммы называются вложенными. Число вложений подпрограмм практически неограничено.

Оператор GOSUB нельзя применять для входа внутрь цикла. Если подпрограмма имеет в своем составе цикл, то он должен быть закончен в этой подпрограмме.

Если обращение к подпрограмме происходит внутри цикла, то выход

из подпрограммы возможен не только по оператору RETURN, но и по оператору NEXT, соответствующему заголовку цикла, из которого производилось обращение.

Примеры:

```
I) :10 GOSUB  30
   :20 PRINT  X :STOP
   :30 REM ПОДПРОГРАММА
   :40 .....
   :50 .....
   :60 .....
   :70 GOSUB  150
   :80 PRINT  Q
   :90 .....
   :100 RETURN : REM КОНЕЦ ПОДПРОГРАММЫ
    .
    .
    .
   :150 REM  ВЛОЖЕННАЯ ПОДПРОГРАММА
    .
    .
    .
   :200 RETU RN : REM КОНЕЦ ВЛОЖЕННОЙ ПОДПРОГРАММЫ
2) :500  GOSUB 750 НЕДОПУСТИМЫЙ ПЕРЕХОД В ЦИКЛ
    .
    .
    .
   :700 FOR I=20 TO 50
    .
    .
    .
   :750 LET A(I) = LOG (I2*A) -  Z(I)
   :760 NEXT I
   :770 RETU RN
```

2.5.10 GOSUB' .

GOSUB' <номер DEFFN'> <список параметров GOSUB'>

<список параметров GOSUB'> ::= <AB>/<символьное значение>/
<список параметров GOSUB'> , <AB>/
<список параметров GOSUB'> ,<символьное значение>

<номер DEFFN'> - в пределах 0 - 255.

Оператор GOSUB' используется для обращения к помеченной подпрограмме, определенной по оператору DEFFN' (см.п. 2.5.5).

По оператору GOSUB' осуществляется переход к оператору DEFFN' (например, GOSUB' 6 служит для перехода к оператору DEFFN' 6). По оператору GOSUB', в отличие от GOSUB, исключается вход в середину подпрограммы. Последним исполняемым оператором помеченной подпрограммы является оператор RETURN (см.п. 2.5.23), который служит для возврата к оператору, следующему за оператором GOSUB'.

Правила использования оператора GOSUB' относительно вложенных подпрограмм и циклов те же, что и для GOSUB (см. п. 2.5.9).

Оператор GOSUB' может содержать параметры, которые должны соответствовать параметрам в операторе DEFFN'. Значения параметров из оператора GOSUB' присваиваются параметрам в операторе DEFFN'. Число параметров и типы параметров в операторе GOSUB' должны соответствовать числу параметров и типам параметров в операторе DEFFN', в противном случае выдается сообщение об ошибке ERR 09.

Примеры:

```
1) :100 GOSUB' 7
   :150 END
   :200 DEFFN'7:SELECT  OC(80)
   :210  RETURN
2) :25 GOSUB' I2 ("ИСКРА",I2.4,3 * У+Х)
   :30 END
   :100 DEFFN' I2 ( A$, B,C(2))
   :110 PRINT  A$, B,C(2)
   :120 RETURN
```

2.5.II. GOTO .

GOTO <номер строки>

Оператор GOTO предназначен для изменения нормальной (в порядке возрастания номеров строк) очередности выполнения строк программы.

При выполнении оператора GOTO в программе осуществляется переход на продолжение счета по программе с первого оператора в строке, номер которой задан параметром <номер строки> .

Оператор GOTO не запоминает адрес оператора следующего за GOTO, поэтому невозможно возвратиться к первоначальной очередности выполнения строк программы.

При использовании оператора GOTO в режиме непосредственного счета, выполнение программы не начинается до тех пор пока не нажата клавиша HALT/STEP, CONTINUE или RUN .

Пример:

```
:I0 K=I5: GOTO 50
:30 Z=K+L
:40 PRINT  Z : END
:50 L=80: GOTO 30
:RUN
95
END PROG RAM
FREE SPACE =....
```

2.5.I2. IF THEN .

IF <условие> THEN <номер строки>

<условие> :: = <выражение> [<список логических элементов>]

<выражение> :: = <AB><операция сравнения> <AB> /<символьный аргумент><операция сравнения> <символьный аргумент>

<список логических элементов> :: = <логическая операция> / <список логических элементов> , <логическая операция>

<логическая операция> :: = AND<выражение> / OR <выражение>/ XOR <выражение>

Оператор IF THEN предназначен для организации условного перехода к указанной строке программы. Если логическое условие, реализуемое с помощью операций сравнения, удовлетворяется, то происходит переход к выполнению строки, номер которой указан в операторе IF THEN . Если условие не удовлетворяется, то выполняется следующий оператор.

При сравнении числовых значений с символьными выдается сообщение об ошибке.

Символьные переменные и константы сравниваются символ за символом, слева направо, с учетом конечных пробелов. Сравнение двух любых символов основано на относительных значениях их шестнадцатиричных кодов , например, шестнадцатиричный код A - 4I, а шестнадцатиричный код B - 42, таким образом A<B .

В операторе IF-THEN , кроме операций сравнения (< , <=, =,>=, >,<>), используются логические операции AND , OR, XOR, причем допускается проверка многочисленных отношений в одном операторе. Если AND определяет два отношения, то пара отношений является истиной только тогда, когда оба отношения выполняются. Если OR отделяет два отношения, то пара отношений является истиной, если хотя бы одно отношение выполняется. Если XOR отделяет два отношения, то два отношения являются истиной, если только одно из

двух отношений (но не оба) выполняется. Сравнение отношений выполняется слева направо без учета старшинства операций.

Примеры:

```
1):10 IF  A>B THEN 300
2):10 IF A$="ДА"  THEN100
3):10 IF B$=HEX(8032) THEN 200
4):10 IF X(I) <>  100 THEN350
5):10 IF  STR (B$ ,I,3)=A$(I)  THEN500
6):10 IF  A>B  AND B<C  THEN 100
7):10 IF  A$=  B$ORW$=C$ THEN220
8):10 IF  A<I XOR  B>0THEN 150
9):10 IF  A=5  OR  B>3  ANDC<>D  OR H=7 THEN 140
10):10 DIM  A$4,  B$5, C$5
   :20 A$="ABC"
   :30 B$=  HEX(4I424321)
   :40 C$= "ABC"
   :50 IF  A$= B$THEN100
   :60 IF  A$=  C$THEN200
   :100 PRINT "A$=  B$"; A$, B$
   :110  GOTO 300
   :200 PRINT"A$=C$"; A$,  C$
   :300 HEXPRINT A$, B$,  C$
```

В результате выполнения программы на дисплей выводится:

```
A$= C$ ABC        ABC
4I424320
4I42432I20
4I42432020
```

2.5.13. INIT .

INIT (<символьный аргумент>) <список символьных переменных>

<символьный аргумент> :: = <символьная переменная> / <символьная константа>

<список символьных переменных> ::= <символьная переменная> / <список символьных переменных> , <символьная переменная>

<символьная переменная> :: = <символьный операнд> / <метка символьного массива>

<символьный операнд> ::= <простая символьная переменная> / <элемент символьного массива> / STR (<символьный элемент> ,<AB>[,<AB>]).

Оператор INIT используется для присвоения символьным переменным и символьным массивам значения параметра, определенного в операторе

в круглых скобках. Если в круглых скобках стоит символьная переменная, то по оператору INIT используется ее первый символ.

Примеры:

1) :10 INIT (HEX00) A$, B$()
2) :40 INIT ("A") C$()
3) A$ = "1234": INIT(A$) B$

2.5.14. LET .

[LET]<присвоение>

<присвоение>:: =<список числовых элементов> =<AB> /<список символьных операнд> = <символьное значение>

<список числовых элементов> :: = <числовой элемент> / <список числовых элементов> ,<числовой элемент>

<список символьных операнд> :: = <символьный операнд> /<список символьных операнд> , <символьный операнд> .

Оператор LET предназначен для присвоения значения, находящегося справа от знака равенства всем переменным, указанным слева от знака равенства.

С помощью одного оператора LET можно присвоить одно и то же значение несколькими переменным, которые должны отделяться друг от друга запятыми (см. пример 1).

Имя оператора LET может отсутствовать (см примеры 3,4). При использовании оператора числовые значения должны присваиваться числовым переменным, а символьные значения - символьным переменным.

Примеры:

1) : 10 LET X(3),Z , Y=P+SIN (P-2.0)
2) LET K=3
3) :10 X=A*B - Z*Y
   :20 A$=B$
   :30 C$,D$ (2) ="ABCDE"
4) :10 C$= "ABCDE"
   :20 A$= "12345"
   :30 D$= STR(A$, 2)
   :40 B$= HEX(41)

2.5.15. MATCOPY .

MATCOPY[-] <символьная переменная> TO [-]<символьная переменная> .

Оператор MATCOPY предназначен для побайтной переписи данных из одной символьной переменной - источника (или части переменной - источника), указанной до параметра TO , в переменную - приемник

(или ее части), указанной после параметра TO.

Каждая переменная рассматривается как одна сплошная строка символов, без деления на элементы.

Выполнение оператора MATCOPY заканчивается, если исчерпана переменная - источник или переменная - приемник.

Если длина переменной - приемника меньше длины переменной -источника, то непомещавшиеся в переменную - приемник данные отбрасываются: если длина переменной - приемника больше , чем длина переменной - источника, то свободные байты в переменной - приемнике заполняются пробелами.

Направление заполнения переменной может быть разным, в зависимости от наличия знака (-);

I) если в операторе нет (-), то переменная - приемник заполняется от левого края и байты выбираются из переменной - источника слева направо (т.е. в том порядке, как они записаны);

2) если в операторе перед переменной - источника стоит (-),то байты из переменной - источника выбираются справа налево (т.е. в порядке обратном тому, как они записаны);

3) если в операторе перед переменной - приемника стоит (-), то переменная-приемник заполняется от правого края.

Переменные источника и приемника могут быть одной и той же переменной (см. пример I).

Примеры:

I) : I0 MATCOPY A¤() TO A¤()

2) :I0 MATCOPY B¤TO C¤()

3) :40 MATCOPY -STR (A¤., 5,I0) TO-STR(B¤, 20)

4) :I5 MATCOPY A¤() TO - B¤()

5) : I0 DIM A¤(5)I, B¤(I,7)I
:I5 A¤(I)="A": A¤(2)="B": A¤(3)="C":A¤(4)="D":A¤(5)="E"
:20 INPUT N
:30 ON N GOTO 40,50,60,70
:40 MATCOPYA¤() TO B¤()
:45 GOTO 80
:50 MATCOPY - A¤() TO B¤()
:55 GOTO 80
:60 MATCOPY A¤() TO -B¤()
:65 GOTO 80
:70 MATCOPY -A¤() TO - B¤()
:80 MATPRINT B¤;

после выполнения программы (в зависимости от значения N - I,2,

3,4 выполняются строки 40,50,60,70) печатается информация:

ABCDE
EDCBA
EDCBA
ABCDE

соответственно.

2.5.16. MATSEARCH.

MATSEARCH <символьная переменная>, <условие> <символьный аргумент>
TO<символьная переменная> [STEP <AB>]
<условие> :: = </ > /<>/ >= /<= / = / * / # <символьный аргумент> ;/ %

Оператор MATSEARCH предназначен для выделения из символьной переменной групп символов, удовлетворяющих условию относительно значения символьного аргумента, заданного в операторе. Условие # <символьный аргумент> обозначает диапазон, границами которого являются значения символьных аргументов. Условие % обозначает условие отождествления.

Просмотр символьной переменной осуществляется слева направо без разделения на элементы, номер первого байта группы символов, удовлетворяющих условию, заносятся как 2 байтовое двоичное значение в символьную переменную.- локатор, заданную в операторе после параметра TO. При указании условия отождествления, кроме номера первого байта заносится 2 байтовое двоичное значение длины группы символов, удовлетворяющих условию отождествления.

При выполнении оператора просматриваются группы символов переменной длиной, равной длине символьного аргумента без учета конечных пробелов (если символьный аргумент заполнен только одними пробелами, то он воспринимается как один пробел). Если пробелы должны быть учтены, нужно использовать функцию STR.

Просмотр групп символов осуществляется с шагом I, если не задан параметр STEP. Если параметр STEP задан, то просмотр осуществляется с шагом, равным целой части <AB>, указанного после STEP, при этом 0< / <AB> /<= 255. При отрицательном значении шага просмотр осуществляется справа налево.

При указании условия отождествления, поисковая переменная может содержать символ "Пусто" (#X00), который при сравнении считается совпадающим с группой символов любой длины, включая длину, равную нулю. Если поисковая переменная начинается с кода "Пусто",

то в длину найденной группы входит количество символов, предшествующих первому совпавшему символу, а номер байта равен номеру первого байта переменной, в которой производится поиск. Если поисковая переменная состоит из кодов "Пусто", то заносимый в локатор номер байта равен единице, а длина найденной группы равна всей длине переменной.

Выполнение оператора заканчивается, если:

1) заполнена символьная переменная - локатор;

2) длина группы символов, подлежащих сравнению (оставшихся для проверки) меньше длины символьного аргумента ;

3) вся символьная переменная просмотрена.

Если переменная - локатор не заполнена полностью после того как поиск завершен, в очередные два байта заносится код HEX(0000).

Символьная переменная - локатор, составленная оператором MATSEARCH, не может быть использована оператором MATMOVE, так как он задает местоположение байтов, а не элементов.

Примеры:

```
1) :10 DIM  A$(100), B$5, C$(10) 3,D $ (5)2
   :20 MATSEARCH C$(), ="XX" TO E%
   :30 MATSEARCH A$, =HEX(0102) TO E%
   :40 MATSEARCH STR(C$(),5,20), <B$  TO E%STEP5
   :50 MATSEARCH STR(C$(),5,20),>B$ TO E%
   :60 MATSEARCH STR(C$(),5,20), #  "XX",B$ TO  E%STEP2
2) :10 DIMA$(I)14:  INIT(FF)G$():K$="A"
   :30  A$(I)="BACDCBFAAFAFAR"
   :40 MATSEARCH A$(),=K$TO  G$()
```

в результате выполнения программы переменная - локатор G$() содержит следующие HEX-коды:

0002 0008 0009 000B 000D 0000 FFFF FFFF FFFF FFFF FFFF FFFF

2.5.17.NEXT

NEXT <простая числовая переменная>

Оператор NEXT определяет конец цикла, который был начат оператором FOR. Переменная, заданная в операторе NEXT, должна соответствовать переменной, определенной в операторе FOR (см.п. 2.5.8).

Оператор NEXT увеличивает значение переменной на величину шага (если шаг не задан, то - на единицу) и проверяет, не превосходит ли новое значение конечного значения арифметического выражения, стоящего за параметром TO в операторе FOR. Если этого не происходит, то осуществляется переход к оператору, следующему за оператором FOR,

в противном случае к оператору, следующему за оператором NEXT.

В режиме непосредственного счета оператор NEXT и соответствующий ему FOR должны быть на одной строке, в противном случае выдается сообщение об ошибке.

Примеры:

```
1) FOR X=I TO 5: PRINT X: NEXT X
2) :I0 FOR M=2 TO M-I STEP  30:   J(M)=I(M)^2: NEXT M
   :30 FOR X=8 TO I6 STEP 4
   :40 FOR A=2 TO 6 STEP 2
   :50 LET B(A,X)=B(X,A)
   :60 NEXT  A : NEXT  B
```

2.5.I8. ON.

ON <AB><параметр перехода> <список строк>

<параметр перехода> :: = GOTO / GOSUB

<список строк> :: = <номер строки> /<список строк> ,<номер строки>

Оператор ON ,предназначен для организации вычисляемого перехода в программе.

Использование оператора ON позволяет осуществить переход к одной из строк из заданного списка номеров строк, в зависимости от значения параметра <AB> , которое определяет позицию (порядковый номер) параметра <список строк> в списке. Значение параметра <AB>округляется до целого числа и по нему определяется, к какому номеру строки будет осуществлен переход. Если значение параметра <AB>меньше единицы или больше числа номеров строк, указанных в операторе в параметре <список строк> , то переход не происходит, а выполняется оператор, следующий за оператором ON .

Примеры:

```
1) :I0 A =2
   :20  ON A  GOTO  60,70,200   при исполнении строки 20 происходит переход к строке 70,т.к.
                                A=2
2) :20 A=5 : ON A GOTO I40,I50  перехода нет,т.к. величина A
                                превышает число номеров строк.
```

2.5.I9. READ.

READ <список для присвоения>

<список для присвоения> :: = <числовой элемент> /<символьный операнд> /<список для присвоения> ,<числовой элемент> /<список для присвоения> ,<символьный операнд>

Оператор READ предназначен для присвоения переменным из списка оператора READ значений констант и переменных, заданных в операторах DATA.

Числовые константы и переменные в операторе DATA должны соответствовать числовым переменным в операторе READ , а символьные константы и переменные - символьным переменным.

Каждый элемент из списка оператора DATA присваивается последовательно очередной переменной оператора READдо тех пор, пока всем переменным оператора READ не будут присвоены значения из оператора DATA, или до тех пор, пока все элементы оператора DATA не будут использованы.

Операторы READ и DATA используются вместе. Первый исполняемый оператор READ ищет первый оператор DATA и т.д.

Если оператор READ содержит больше переменных, чем имеется значений в операторе DATA, то производится переход к следующему оператору DATA, если его нет, выдается сообщение об ошибке.

Если оператор READ содержит меньше переменных, чем значений в операторе DATA , то следующий оператор READ начинает присвоение с первого неиспользованного значения в операторе DATA (при отсутствии оператора RESTORE см. п. 2.5.19).

Операторы DATA могут помещаться в любое место программы в том порядке, в каком их значения должны использоваться в программе.

Оператор READ может использоваться в режиме непосредственного счета только после запуска счета по программе по оператору RUN (см.пример I), т.к. только в этом случае система образует список данных.

Примеры:

```
I) :IO DATA  I,2,3
   :20  PRINT A
   .
   .
   :RUN
   :READ  A,B
2) :IO READ A,C%, STR (B%, 5,3)
3) :IO READ A,B,C
   :40 DATA  4,3I5,3.98
```

2.5.20 REM .

REM <последовательность символов>

Оператор REM предназначен для включения в текст программы различных пояснений или объяснений. Это облегчает чтение листинга

программы.

Оператор REM является невыполняемым оператором, он игнорируется системой, когда встречается во время выполнения программы. Поэтому операторы REM могут вставляться в любое место программы.

Однако пояснения занимают место в памяти и приводят к увеличению длины программы. Цепочку символов в операторе REM можно вводить без кавычек, но в ней нельзя использовать двоеточие, так как этот знак является разделением операторов в строке.

Примеры:

1) :10 REM ПРОГРАММА
2) : 20 REM "НАЧАЛО"

2.5.21. REPLACE .

REPLACE <числовой элемент> ,<символьная переменная> ,<символьный аргумент> [,<символьный аргумент>]

Оператор REPLACE обеспечивает контекстное изменение символьных переменных. Параметр <символьная переменная> представляет собой символьную переменную или символьный массив, который необходимо изменить. Первый параметр <символьный аргумент> содержит контекст , который необходимо заменить в символьной переменной. Второй параметр <символьный аргумент> представляет собой образ, на который необходимо изменить контекст, содержащийся в символьной переменной. В числовой переменной формируется число найденных контекстов. Если символьная переменная не содержит заданного контекста, то значение числовой переменной устанавливается равным нулю.

Пример:

```
:10 DIM  A$X10
:20  A$= "123123123I"
:30 REPLACE A,A$,"123","456"
:40 PRINT A$, "A =";A
: RUN
456456456I A=3
```

Если контекст превышает по длине символьную переменную, то замена не производится, выполнение оператора прекращается, ошибка не выдается. Если контекст найден, но образ не помещается в символьную переменную, ошибка не выдается и производится замена символов контекста на символы образа, помещающиеся в символьную переменную.

Если длина образа меньше длины контекста, то контекст заменяется на образ по всей длине символьной переменной, а оставшиеся символы заменяются пробелами.

Если символьные аргументы заданы функцией STR или символьной константой, операция поиска и замены производится по указанной длине, не исключая концевые пробелы. Во всех остальных случаях концевые пробелы при поиске и замене не участвуют.

При отсутствии второго параметра <символьный аргумент> производится исключение указанного контекста из символьной переменной.

Примеры:

```
1) :10 DIM  A$*10
   :20  A$ = "123123128I"
   :30 REPLACE A,A$"123", "4567"
   :40 PRINT  A$: PRINT "A="; A
   :RUN
   4567456712
   A=2
2) :10 DIM  A$*10
   :20  A$ ="ABCABCABCA": B$="ABC"
   :30 REPLACE A,A$,B$, "D"  :  PRINT A$
   :RUN
   DDDA
3) :10 A$ ="ABCDABCDAB":B$ = "CD "
   :20 REPLACE A,A$,  B$ :PRINT A$
   :RUN
   ABABAB
```

2.5.22. RESTORE

RESTORE [<AB>][,<номер строки>]

Оператор RESTORE предназначен для повторного использования значений оператора DATA операторами READ .

Значение <AB> является номером элемента в операторе DATA, с которого начнется считывание последующим оператором READ .

Если задан параметр <номер строки> , то указатель данных устанавливается на оператор DATA, содержащийся в этой строке. Если оператор DATA в указанной строке отсутствует, выдается сообщение об ошибке ERR I3.

Оператор RESTORE без параметров устанавливает указатель текущего элемента на первый элемент первого оператора DATA в программе.

Примеры:

```
1) :10 RESTORE
2) :10 RESTORE 3
```

```
3) :IO DATA  I,2,3,4,5
   :20 DATA  6,7,8,9,IO
   :30 RESTORE 3,20
   :40 READ  A,B,C :  PRINT  A,B,C
   :RUN
   8  9   IO
4) :IOO RESTORE (X-У)/2
```

2.5.23. RETURN .

Оператор RETURN предназначен для обозначения конца подпрограммы и передачи управления оператору, следующему за последним исполненным оператором GOSUB или GOSUB'.

Оператор RETURN автоматически стирает из магазина RENEXT адрес возврата из подпрограммы вместе с параметрами цикла, содержащегося в пределах подпрограммы.

Если подпрограмма была вызвана нажатием клавиши специальной функции, то оператор RETURN заканчивает работу подпрограммы и передает управление на клавиатуру.

Повторные входы в подпрограмму без исполнения оператора RETURN вызывают накопление адресов возвратов в памяти машины, которое может привести к переполнению. Эта информация может стираться из памяти без осуществления возврата к основной программе оператором RETURN CLEAR (см.п. 2.5.24).

Иногда оператор RETURN используется для того, чтобы завершить циклы, которые были начаты после того, как был совершен переход к подпрограмме (см.пример 3).

Если оператор RETURN исполняется до завершения цикла FOR, то информация цикла стирается из магазина вместе с параметрами подпрограммы, исполнение оператора NEXT данного цикла после оператора RETURN вызовет ошибку.

Примеры:

```
I):IO GOSUB  30
  :20 PRINT  X: STOP
  :30 REM ПОДПРОГРАММА
  :40 A=I: B=2
   .
   .
  :70 PRINT A,B
  :80 RETURN : REM  КОНЕЦ ПОДПРОГРАММЫ
2):IO GOSUB , 3(A,B¤)
```

```
:20 END
 .
 .
:100 DEFFN' 3(X,N$)
:110 PRINTUSING   70,X,N$
:120  ЦЕНА=$.$$   , КОД = $$$$
:130 RETURN
3) :10 GOSUB 100
 .
 .
 .
:100 REM ПОДПРОГРАММА
:110 FOR I=1 TO 10
 .
 .
 .
:190 RETURN
```

2.5.24 RETURN CLEAR .

RETURN CLEAR [ALL]

Оператор RETURN CLEAR предназначен для стирания адреса возврата из подпрограммы и информации цикла из памяти.

Исполнение оператора RETURN CLEAR стирает адрес возврата исполняемой подпрограммы, затем продолжается реализация программы с оператора, следующего за оператором RETURN CLEAR .

Параметр ALL обеспечивает сброс всего магазина возвратов из циклов и подпрограмм.

Примеры:

```
1) :100 DEFFN'15: RETURN CLEAR
2) :10 INPUT S
   :20  S1 =S1+S   +100000 : PRINT S1
   :40 STOP "НАЖМИТЕ КЛАВИШИ CØ1, CØ2, CØ3"
   :50 DEFFN' 1
   :60 S2=12000+.05*S1 :PRINT S2: RETURN
   :70 DEFFN'2
   :80 S3 =.05 * S1 :PRINTS3:GOTO      20
   :100 DEFFN' 3  :  A1= S1 * .01:PRINT A1: RETURNCLEAR
   :110 END
```

2.5.25. STOP.

STOP [<символьная константа>][#]

Оператор STOP предназначен для программируемого останова машины при счете по программе. Программа может содержать более чем один оператора STOP.

При выполнения оператора на дисплее индицируется слово STOP и значение символьной константы, если она указана в операторе, и/или номер программной строки, содержащей STOP, если в операторе используется символ # .

Для продолжения счета по программе используется команды CONTINUE или HALT/STEP. После ввода команды CONTINUE выполнение программы осуществляется с оператора, следующего за оператором STOP, по команде HALT/STEP осуществляется пошаговое исполнение программы (шаг - один оператор) .

Примеры:

1) :200STOP "СМЕНИТЬ КАССЕТУ"
2) :300 STOP "ТЕКСТ" #
   :RUN
   STOP ТЕКСТ 300

2.6. Операторы преобразования данных .

2.6.1. CONVERT .

CONVERT <преобразование>

<преобразование> :: = <символьная переменная> TO <числовая переменная> / <AB> TO <символьная переменная> ,(<формат>)

<формат> :: = <цепочка символов> / <символьный операнд>

Оператор CONVERT используется для преобразования символьной переменной в числовую и наоборот. Оператор CONVERT имеет две формы.

Оператор CONVERT первой формы преобразует значение символьной переменной в числовое значение и присваивает полученное значение числовой переменной. Значение символьной переменной должно представлять правильное по формату число (целое или действительное). Часть символьной переменной можно преобразовать в числовую, используя функцию STR.

Примеры:

1) :10 AX = "1234"
   :20 CONVERT AX TO X
   :30 PRINT "X="; X
   : RUN
   X=1234

2) :10 A$="ABC12.45DEF"
:20 CONVERT STR (A$,4,5) TO X
:30 PRINT "X="; X
:RUN
X=12.45

Оператор CONVERT второй формы преобразует числовое значение в соответствии с форматом, заданным в операторе и присваивает его символьной переменной.

Форматы переменных:

1) Формат 1 – число в естественной форме, например: +###.#

2) Формат 2 – число в экспоненциальной форме, например: -#.###^^^^

Числовые значения формируются согласно следующим правилам:

1) если формат начинается со знака (+), то в символьной переменной указывается знак (+ или –) соответственно знаку числовой переменной;

2) если формат начинается со знака (–), то знак (+) в символьной переменной заменяется пробелом , знак (–) указывается;

3) если в формате нет знака, то в символьной переменной знак числа также отсутствует;

4) если используется формат 1 , то числовое значение преобразуется в естественную форму, при этом в соответствии с форматом отбрасываются избыточные цифры или добавляются недостающие нули перед целой частью числа, десятичная точка устанавливается в определенное форматом место. Если число превышает отведенный ему формат, то выдается сообщение об ошибке;

5) если задан формат 2, то числовое значение преобразуется в экспоненциальную форму. Показатель ^^^^ заменяется на E+XX или E-XX, где XX – показатель степени, при основании 10; нули, стоящие перед целой частью, в формат не включаются.

Примеры:

1) :10 X=-15.23: A$="ABCDEF GHIJKLMN "
:20 CONVERT X^2 TO STR (A$, 3,6),(+###.#)

В результате выполнения программы символьной переменной A присвоено значение AB+231.9IJKLMN .

2):10 X=12
:20 CONVERT X TO A$, (-##.^^^^)

В результате выполнения программы символьной переменной присвоено значение 1.2E+01, пробел перед числом соответствует знаку плюс в числовой переменной.

2.6.2. PACK

PACK (<формат>)<символьная переменная> FROM <список PACK>

<список PACK> ::= <AB> / <числовая переменная> / <список PACK>, <AB> / <список PACK>,<числовая переменная>.

Оператор PACK предназначен для упаковки значений числовых переменных и массивов в символьную переменную по заданному формату.

Числовые переменные преобразуются в упакованную десятичную форму (две цифры - байт) согласно формату и последовательно присваиваются символьной переменной. Переменные массива последовательно упаковываются, начиная с первого элемента массива. Если символьная переменная недостаточна, чтобы записать все числовые величины, которые следует упаковать, то выдается сообщение об ошибке. В операторе PACK используются такие же форматы, что и в операторе CONVERT.

Числовая переменная упаковывается по следующим правилам:

1) знак ## формата соответствует одной цифре и требует 1/2 байта;

2) если в формате определен знак плюс или минус, то он занимает вместе со знаком показателя степени старшую тетраду первого байта и равен:

0 - если и число и показатель положительные;

1 - если число отрицательное, а показатель положительный;

8 - если число положительное, а показатель отрицательный;

9 - если и число и показатель отрицательные;

3) если формат не имеет знака, то формируется абсолютная величина числа, а знак показателя степени считают положительным;

4) положение десятичной точки не сохраняется в памяти. При распаковке данных (см. UNPACK) положение десятичной точки должно определяться форматом, который должен быть таким же, как формат упаковки;

5) в символьной переменной числовая переменная формируется как число и всегда занимает целое число байт;

6) если задан формат 1, то значение числовой переменной упаковывается с отбрасыванием цифр или добавлением нулей в дробной части и, если требуется, с добавлением нулей перед целой частью числа согласно заданному формату;

7) если задан формат 2, то значение числовой переменной упаковывается без добавления нулей перед целой частью. Показатель степени занимает один байт.

Примеры:

1) :10 PACK (##.##^^^^) A$ FROM X

2) :20 PACK(+#.##^^^^) A$ FROM N ( )

2.6.3. ROTATE

ROTATE [C] (<символьная переменная> <AB>)

где <AB> от I до 7.

Оператор ROTATE предназначен для циклического сдвига битов каждого символа (т.е. каждого байта) заданной символьной переменной.

Число позиций, на которое надо переместить каждый символ, определяется целой частью <AB>. Если эта величина положительная, биты каждого символа символьной переменной перемещаются влево, если отрицательная - то вправо. Сдвиг производится во всех символах символьной переменной, включая конечные пробелы.

Если в операторе ROTATE не задан параметр "C", то оператор перемещает биты каждого символа на определенное число бит, причем старшие биты каждого символа замещают по кругу младшие биты при сдвиге влево; при сдвиге вправо младшие биты замещают старшие биты каждого символа.

Если в операторе ROTATE задан параметр "C", то вся величина символьной переменной сдвигается на обозначенное число позиций (с переносом между байтами).

Примеры:

```
1) : 5 DIM  A$ 3
   : 10  A$ =HEX(0123FE)
   : 20 ROTATE  (A$ ,4)
```

При выполнении программы производится четырехкратный сдвиг битов символьной переменной A$:

```
0000  0001  (01)
0010  0011  (23)      исходное состояние
1111  1110  (FE)

0001  0000  (10)
0011  0010  (32)      конечное состояние
1110  1111  (EF)
```

получается результат: A$ = HEX (1032EF)

```
2)  : 10 DIM  A$ 2
    : 20   A$  = HEX (8142)
    : 30 ROTATE C (A$ ,1)
```

При выполнении программы производится сдвиг на одну позицию символь-

ной переменной A%:

1000	0001	0100	0010	исходное состояние
0000	0010	1000	0101	конечное состояние

Получается результат: A% =HEX(0285)

2.6.4. UNPACK

UNPACK (<формат>) <символьная переменная> TO <список UNPACK>
<список UNPACK> :: = <числовая переменная> / <список UNPACK>,
<числовая переменная>

Оператор UNPACK используется для распаковки числовых данных, первоначально упакованных с помощью оператора PACK.

Начиная с первого элемента, обозначенной символьной переменной или массива, упакованные данные распаковываются в соответствии с форматом, а затем заносятся в заданные числовые переменные или массивы. Формат упакованных данных указывается в операторе. В операторе UNPACK должен быть использован тот же формат, который был использован для упаковки данных (см. оператор PACK п.2.6.5.2). Если упакованных данных не хватает для распаковки, то выдается сообщение об ошибке. Все данные запоминаются последовательно в заданных числовых переменных и массивах. Массивы заполняются, начиная с первого элемента массива.

Примеры:

1) : 10 UNPACK(#^^^^) A% TO X,Y,Z
2) : 20 UNPACK(+#.##^^^^)A% TO B ( )
3) : 30 UNPACK (##.##) A% TO X,Y

2.6.5. %PACK.

%PACK [(<формат упаковки>)] <символьная переменная>[,<числовая переменная>] FROM <список переменных>
<формат упаковки> :: = F = <символьная переменная> / D = <символьная переменная>

Оператор %PACK предназначен для последовательной упаковки значений переменных из списка переменных в символьную переменную, являющуюся буфером для размещения данных, необходимых разделителей и другой служебной информации в зависимости от формата упаковки. Формат упаковки значений переменных задается параметрами F и D или их отсутствием.

При упаковке данных символьная переменная - буфер должна иметь длину, достаточную для размещения значений всех переменных, указанных в списке переменных, а также всех необходимых разделителей. В противном случае выдается сообщение об ошибке. Если длина буфера больше, чем необходимо для упаковки значений переменных из списка переменных и разделителей, то неиспользуемые байты сохраняют старые значения.

Окончание реализации оператора)(PACK происходит, когда исчерпан список переменных. По окончании выполнения оператора)(PACK в числовую переменную заносится число упакованных в буфер байтов.

При реализации оператора)(PACK массивы упаковываются поэлементно строка за строкой.

Примечание.

Если при заполнении символьной переменной - буфера возникла ошибка, то часть значений переменных из списка переменных, находящихся в списке до значения, упаковка которого привела к ошибочной ситуации, записана в символьную переменную - буфер.

Если в операторе задан параметр D , то значения переменных упаковываются в разделительный формат , т.е. в виде:

значение разделитель значение разделитель...значение разделитель

где "значение" - это значение переменной или элемента массива, если в списке переменных задана переменная или массив соответственно; „разделитель" - это HEX - код второго байта символьной переменной (символьного массива), указанного после D =. Длина этой переменной (массива) должна быть не менее двух байтов, при этом массив рассматривается без учета деления на элементы. Таким образом, разделитель занимает всегда один байт.

Для правильной работы обратного оператора)(UNPACK с параметром D код разделителя следует выбирать не совпадающим с кодами символов значений переменных.

Первый байт символьной переменной (символьного массива) должен содержать коды HEX (00), HEX(01), HEX(02) или HEX(03), в противном случае выдается сообщение об ошибке . Эта информация не используется оператором)(PACK, но необходима для оператора)(UNPACK с параметром D ,т.е. при распаковке данных,представленных в разделительном формате.

При упаковке в разделительный формат значений переменных представляются в следующем виде:

I) символьные переменные.

```
CCCCCCCCCC.....CCC
```

где: C - HEx-код символа.

Количество байтов, отведенное в буфере под значение символьной переменной (элемента символьного массива), определяется ее длиной.

2) числовые переменные.

Значения числовых переменных могут быть представлены при упаковке в двух формах: естественной и экспоненциальной, в зависимости от величины значения.

Если значение переменной $1E-12 < X < 1E+13$, то оно представляется в естественной форме и занимает от двух до 15 байтов, при этом выводятся только значащие цифры.

Значение числовой переменной в естественной форме имеет вид:

```
S DDDDDDD.DDDDDD
```

где: S - знак числа. Если число положительное, то записывается пробел (HEX(20)), если отрицательное, то минус (HEX(2D)) ;
D - цифры числа 0-9 (HEX(30) - HEX(39)); десятичная точка (HEX(2E)).

Если значение переменной $1E+13 <= X < 1E-13$, то оно представляется в экспоненциальной форме, занимает 15 байтов и имеет вид:

```
SD.DDDDDDDDEPDD
```

где: S, .., D - параметры, совпадающие с описанными выше;
P - знак порядка. Если порядок положительный, то записывается знак плюс (HEX(2B)), если отрицательный, то минус (HEX(2D));
E - признак порядка числа (HEX(45)).

Примеры правильного синтаксиса:

```
1) :10 X PACK  (D=D1X) RX FROM X,U,PX,AX( )
2) :10 X PACK  (D=DX ())RX( )  FROM X ( ), U, P(3)
3) :10 X PACK  (D=DX(6) AX( ) (10,100) FROM X, U , T ( )
4) :20 X PACK  (D=DX)WX ,A  FROM  X, U, RX,AX ( )
5) :30 X PACK  AX( ) (10,100), D FROM  X, Y, T ( )
```

Пример:

```
:10 DIM  B%(26), A%(5)
:20 A%  = "ABC": X=-12:  Y=4.56E-18
:30 D%  =HEX(002C)
:40 %PACK (D=D%) B% FROM  X, A%, Y
:50  PRINT  "B% =" ;  B%
```

В результате выполнения программы на экране дисплея будет выведено:
B% =-12,ABC , 4.56000000E-18,

В качестве разделителя используется запятая (HEX(2C)).
Если задан параметр P, то значения переменных упаковываются в полевой формат, т.е. каждому значению переменной или элемента массива из списка переменных отводится свое поле в символьной переменной - буфере. Данные в буфере представляются в виде:

поле значения 1	поле значения 2	.....	поле значения N

Описание типа поля значения и его длины для каждого значения из списка переменных задается двумя байтами символьной переменной (символьного массива), указанной после P =. При этом N переменной (метке массива) из списка переменных соответствуют (2N -1) и 2N байты символьной переменной (массива); символьный массив в этом случае рассматривается без учета деления на элементы.

Пара байтов, характеризующая поле значения для переменной из списка переменных, используется следующим образом:

1) первый байт задает тип поля значения (см.табл.2) (он может иметь лишь значения, перечисленные в таблице);

2)второй байт задает длину поля значения (количество байтов, отводимое под поле значения, определяется десятичным эквивалентом HEX-кода, содержащегося в этом байте).

Каждая переменная (метка массива) из списка переменных должна иметь свое описание (два байта) в символьной переменной (массиве), при этом описание для метки массива, а не для всего массива в целом.

Таблица 2

Значение байта - тип поля значения

HEX-код байта - тип поля значения	Соответствие формату
00	пропуск в символьной переменной количества байтов, указанного во втором байте - описания поля значения.
I0	числовой свободный формат.
2X	числовой десятичный формат.
5X	числовой упакованный формат.
A0	формат для символьных данных.

Примечания:

I) во всех форматах, кроме формата, задаваемого HEX (00), второй байт определяет количество байтов, отводимых в символьной переменной под упакованное значение;

2) X - шестнадцатиричная цифра, десятичный эквивалент которой определяет положение запятой в числе от правого края, а не от левого;

3) оператор ⌘РАСК с параметром F не должен начинаться с пропуска байтов в символьной переменной, т.е. первый байт начального поля значения не может равняться HEX (00). В противном случае выдается сообщение об ошибке.

Количество пар байтов, первый байт которых не есть HEX(00), должно быть не меньше количества переменных и числа элементов массивов в списке переменных. Если первый байт в паре есть HEX(00), то второй байт пары определяет количество пропускаемых байтов в символьной переменной до записи в него следующего упакованного значения из списка переменных. При этом пропускаемые байты сохраняют старые значения. При упаковке данных по оператору ⌘РАСК с параметром F тип переменных (меток массивов) и тип поля значения должны совпадать, т.е. символьные переменные (массивы) должны упаковываться в формат символьных данных, а числовые - в форматы числовых данных (см.табл.2). В противном случае выдается сообщение об ошибке.

Если поле значения больше, чем необходимо для размещения значения,

то свободные байты поля значения заполняются пробелами и располагаются после значения в случае форматов A0 и I0, а в случае форматов 2X, 5X заполняются нулями, которые могут располагаться как до, так и после значения, согласно указанию о местоположении запятой в числе.

Оператор ⌘ PACK с параметром F упаковывает данные в следующие форматы:

1) символьные (тип поля значения A0)

```
CCCCCCCCCC.....CCCCC
```

где C - байт символа;

2) числовые имеют несколько форматов представления в зависимости от типа поля значения (см.табл.2).

Числовой свободный формат (тип поля значения I0) совпадает с числовыми формами данных для разделительного формата оператора ⌘ PACK с параметром D .

Числовой десятичный формат (тип поля значения 2X):

```
SDDDDDDDD.DDDDD
```

где : S - знак числа. Если число положительное, то записывается знак плюс HEX (2B), если отрицательное - минус HEX(2 D);
D - цифры числа 0-9 (HEX(30)-HEX(39) в естественной форме,т.е. без экспоненты.

Числовой упакованный формат (тип поля значения 5X):

```
NNN...NNNNNNZ
```

Где: N - цифры числа 0-9; Z - знак числа. Если число положительное, то Z =C, если отрицательное, то Z = D.

Местоположение десятичной точки (число) от правого края поля в форматах 2X, 5X определяется второй тетрадой байта - тип поля значения.

Примеры правильного синтаксиса:

1) :I0 ⌘ PACK (F=F⌘) R⌘ FROM X,Y , A⌘ ( )
2) :I0⌘ PACK (F = F⌘ ())H⌘ () FROM X, Y, A,(1,2)

т.е. как и для разделительного формата оператора ℵ PACK.
Пример:

```
:10 DIM  Bℵ(50
:20 Aℵ ="ABC"; X=-12: Y =1.2345: Z=12.345:   INIT(FF)Bℵ
:30 Fℵ =HEX(A005200510052400A0002530552065506)
:40 ℵ PACK (F=Fℵ )Bℵ FROM Aℵ ,X, Y, Z, Z, Z, B
:50  HEXPRINT  Bℵ
```

В результате выполнения программы на экране дисплея будет выведен следующий результат:

4142432020203030313220312E32332B3030303132333 43530FFFF0000123450
0000000123400001234500CFFFFFFFFFFFF

Если в операторе ℵ PACK параметры D и F отсутствуют, то данные по оператору упаковываются в стандартный формат представления данных на внешних носителях (дисках) по операторам DATA $AVE DA, DATA $AVE, DATA $AVE DC, т.е. в виде:

а разделитель данные разделитель .... разделитель данные

где: А - контрольный байт. Эти байты не используются операторами ℵ PACK и ℵ UNPACK, а лишь дают системе информацию о начале и конце блока данных; разделитель данных содержит информацию о типе следующих за ним данных и их длине в байтах, при этом информация в разделителе формируется следующим образом:

тип значения длина значения

где: старшие два бита первого байта задают тип значения (00 - числовое значение, 01 - символьное значение). Остальные шесть бит этого байта и следующий за ним байт содержит длину значения в байтах.

При отведении места под символьную переменную необходимо учитывать место под контрольные байты и разделители данных.

Оператор ℵ PACK без параметров упаковывает данные в соответствии с форматами представления их на внешних носителях.

Примеры правильного синтаксиса:

1) :10 ℵ PACK Bℵ FROM X,B,Pℵ , Aℵ ()
2) :20 ℵ PACK Dℵ (10,100) FROM B1ℵ (A,6),A( )

2.6.6. ⅄TRAN.

⅄TRAN (<символьная переменная>, <символьная переменная>) [<элемент HEX>] [R]

R - параметр, задающий тип преобразования.

Оператор ⅄TRAN предназначен для преобразования кодов. Возможны два вида преобразования:

1) указан параметр R. В этом случае вторая символьная переменная используется как список преобразования. То есть каждый байт первой символьной переменной преобразуется по следующим правилам: во второй символьной переменной отыскивается ближайший с начала байт, равный рассматриваемому байту первой символьной переменной; байт, предшествующий найденному, второй символьной переменной заносится на место рассматриваемого байта в первой символьной переменной (см.пример 1);

2) не указан параметр R. В этом случае вторая символьная переменная используется как таблица преобразования, т.е. каждый байт первой символьной переменной заменяется байтом второй символьной переменной, номер которой на единицу больше значения байта первой символьной переменной (см.пример 2).

Байты первой символьной переменной просматриваются последовательно слева направо (массивы просматриваются строка за строкой).

Если в операторе ⅄ TRAN указан параметр <элемент HEX>, то коды байтов из первой символьной переменной маскируются, а затем осуществляется поиск во второй символьной переменной.

Маскирование - это логическое умножение байта и параметра <элемент HEX> (соответствует оператору AND).

Для реализации оператора ⅄TRAN с параметром R байты во второй символьной переменной используются парами (первый байт представляет собой код, на который надо заменять коды (маскированные коды), идентичные коду, содержащемуся во втором байте).

При появлении двух подряд идущих символов пробел (HEX(2020)) реализация оператора завершается.

Если во второй символьной переменной не найден соответствующий код байта (для операторе ⅄ TRAN с параметром R) или вторая символьная переменная содержит меньше байтов, чем номер, вычисленный по коду байта, выбранного из первой символьной переменной для оператора ⅄TRAN без параметра R, то код байта (маскированного байта) из первой символьной переменной возвращается на свое место.

Примеры правильного синтаксиса:

1) :10 ⅄ TRAN (A1⅄ (),B⅄)

2) :20 ⅄ TRAN(A⅄() , M⅄ (3))

Примеры:

```
1) :10 DIM A$ 5, B$ 14
   :20 A$ = "GOSIO"
   :30 B$ =HEX (4I2A4247434F2050442344512020)
   :40 $TRAN (A$, B$) R
   :50  PRINT  "A$ ="; A$
```

В результате выполнения программы распечатается: A$ =

```
2) :10 DIM A$ =4, R$
   :20A$= "GPI"
   :30  R$ ="0123456789ABCDEF"
   :40$TRAN (A$ ,R$) OF
   :50  PRINT "A$ =" ; A$
```

В результате выполнения программы распечатается: A$ = 7030

2.6.7 $UNPACK .

$UNPACK [(<формат упаковки>)]<символьная переменная> [, <числовая переменная>] TO <список переменных>

Оператор $UNPACK предназначен для распаковки данных из символьной переменной - буфера и последовательного присвоения распакованных значений переменным (массивам) из списка переменных.

Формат, в котором упакованы данные в символьной переменной, должен совпадать с форматом, тип которого задается параметрами F, D или их отсутствием в операторе $UNPACK . При несовпадении форматов может быть выдано сообщение об ошибке или данные после распаковки будут искажены.

Реализация оператора $UNPACK заканчивается, если исчерпан весь список переменных или, если исчерпан буфер (подробнее см. описание каждой из форм оператора $UNPACK).

При реализации оператора $UNPACK , если в списке переменных стоит метка массива, то данные, распаковываемые из символьной переменной, заносятся в массив поэлементно строка за строкой.

Если при распаковке символьной переменной возникает ошибка, то часть переменных (массивов, из списка переменных, находящихся за переменной при заполнении которой возникла ошибка, имеют новые (распакованные значения), а остальные сохранят старые значения.

В числовую переменную по окончании выполнения оператора $UNPACK заносится число использованных в буфере байтов.

Если в операторе задан параметр D , то при распаковке значения предполагается, что они упакованы в символьной переменной в двоичном формате, т.е. занесены в него по оператору $ PACK.

с параметром D или получены другим способом, но представлены в виде:

значение	разделитель	значение	разделитель	...	значение	разделитель

смысл понятий "значение" и "разделитель" совпадает с их смыслом в операторе ℋ PACK с параметром D .

Элементы, указанные в скобках, являются допустимыми при некоторых значениях первого байта символьной переменной (массива), указанной после D = (см. табл.3).

Описание использования символьной переменной (массива) совпадает с ее использованием в операторе ℋ PACK с параметром D , за исключением ее первого байта, использование которого в операторе ℋ UNPACK описано в таблице 3 и проиллюстрировано примером 2.

Таблица 3

Значение HEX-кода первого байта символьной переменной

Значение HEX-кода первого байта	Правила распаковки по оператору ℋ UNPACK (разделительный формат)
00	Ошибка, если данных в символьной переменной недостаточно для всего списка переменных. Дополнительный разделитель в символьной переменной означает пропуск элемента из списка переменных (дополнительный разделитель есть, если между разделителями нет данных).
01	Если данных в символьной переменной недостаточно для всего списка переменных, то оставшиеся элементы из списка переменных игнорируются. Дополнительный разделитель в символьной переменной означает пропуск элемента из списка переменных.
02	Ошибка, если данных в символьной переменной недостаточно для всего списка переменных. Дополнительные разделители в символьной переменной игнорируются.
03	Если данных в символьной переменной недостаточно для всего списка переменных, то оставшиеся элементы из списка переменных игнорируются. Дополнительные разделители в символьной переменной игнорируются.

Примечание.

Переменные, стоящие в списке переменных, но не использующиеся (пропускаемые), сохраняют старые значения.

Для того, чтобы данные, содержащиеся в символьной переменной, могли быть распакованы, они должны быть представлены в виде:

1) символьные значения (см.п. 2.6.5).

При распаковке по оператору %UNPACK длина распакованного значения равна меньшей из двух длин - длины переменной, в которую заносится распакованное значение, и длины упакованного значения.

При этом, если длина упакованного значения меньше длины переменной, которой будет присвоено это значение после распаковки, то распакованное значение дополняется пробелами. Если длина упакованного значения больше, то она устанавливается по длине переменной, которой присваивается.

2) числовые значения.

Могут быть представлены в символьной переменной в двух формах (см.п.2.6.5).

Числовые значения всегда могут быть распакованы в символьную переменную, а символьные значения в числовую лишь при соблюдении всех ограничений на формат числового значения (в противном случае выдается сообщение об ошибке).

Примеры правильного синтаксиса:

```
1) :IO%UNPACK (D=P%) R% TO X, Y, A% ( )
2) :IO%UNPACK   (D=P%( )  )R% TO  X ( )
3) :IO%UNPACK   (D=STR (P%,3,Y))W% ()(10,20)   TOX
```

Примеры:

```
1) :10 DIM  R%23
   :20  P%=HEX (002C)
   :30  R%  ="-12,ABC,4.560000000E-18"
   :40%UNPACK (D=P%) R%  TOX , A% ,Y
   :50  PRINT "X=";  X, "A% =" ; A% , "Y =";Y
```

В результате выполнения программы на дисплее будут получены результаты:

```
X=-12     A% =ABC              Y = 4.560000000E-18
```

```
2)  :10  DIM   R% 12
    : 20  P   = HEX (032C)
    : 30   R% ="ABC, DEF,  GHI"
    : 40%UNPACK  (D =P% ) R%  TOW% , X%, Y%, Z%
    : 50   PRINT  „W% =";  W%;„X% =";X%;„Y% =";  Y%;„Z% =";  Z%
```

В результате выполнения программы на дисплее получается следующий результат:

W$ = ABC X$ =DEF Y$ = GHI Z$ =

Если оператор 20 заменить на 20 P$ = HEX(002C) или 20 P$ = HEX (012C), то будут получены результаты:

W$ = ABC X$ =DEF Y$ = Z$ = GHI

Если оператор 20 заменить на 20 P$ =HEX(022C), то при выполнении программы будет получено сообщение об ошибке.

Если в операторе $UNPACK задан параметр F, то при распаковке значений предполагается, что они упакованы в символьной переменной в полевом формате; т.е. занесены в него по оператору $ PACK с параметром F или получены другим способом, но имеют формат представления, аналогичный полученному по оператору $ PACK с параметром F.

Распаковка значений из символьной переменной в список переменных происходит в соответствии с информацией, хранящейся в символьной переменной (массиве.), где (2N -1) байт задает тип поля упаковки, а - 2N байт задает длину, отведенную в символьной переменной под значение N переменной в списке переменных (подробнее см. оператор $ PACK полевая форма). При несоответствии форматов и длин значений, находящихся в символьной переменной, и форматов и длин значений, задаваемых соответствующими байтами в символьной переменной (массиве), может произойти искажение информации, заносимой в переменную из списка переменных, или может быть выдано сообщение об ошибке.

Для обеспечения возможности распаковки данных форматы представления данных должны соответствовать приведенным в операторе $ PACK, причем допустимы следующие отклонения:

1) во всех числовых форматах допустимы дополнительные пробелы, которые игнорируются при распаковке по оператору $UNPACK;

2) допустимо представление числовой информации в формате 10 с плавающей запятой, аналогично представлению в разделительном формате с плавающей запятой;

3) в формате 5X знак плюс может иметь в своем коде вместо кода "C" коды "A" и "E", а минус - вместо "D" код "B".

Примеры правильного синтаксиса:

1) :40 $UNPACK (F=P$) R$ TOU$, X$, Y$, Z$

2) :40 $UNPACK (F=A$ ()) R$ () (10,20) TOU, X, Y$, Z ()

3) :40 $UNPACK (F=P$) R$, K TOU$,X$, Z$

Примеры:

1) :10 DIM R$ 37, A$ 9, C$ 3

:20 R$ ="J. SMITH M5.80130 1234.56789"

```
:30 P$ =HEX(A00AA00I1007I003I010)
:40 %UNPACK (P=P$ )R$ TOA$,C$, X ,Y,Z
:50   PRINT  A$; C $; X; Y; Z
```

После выполнения программы на дисплее будут получены результаты:

```
J. SMITH  5.8013  I  234.56789
```

Видно в частности: %UNPACK допускает отклонения от форматов. Хотя упакованные числа без знака, распаковка все-таки произведена.

```
2) :I0 DIM  R$21
   :20  R$ = "+I234567"
   :30 F$ =HEX (240BA006A005)
   :40 STR (R$, 9,I0)=HEX(4I424344454647484950)
   :50%UNPACK (F=P$ ) R$ TOX, Y$, Z$
   :60       PRINT  X, Y$ , Z$
```

После выполнения программы на дисплее будут получены результаты:

```
   I23.4567     ABCDE          FGHIP
```

Если в операторе %UNPACK параметры F и D отсутствуют, то данные, находящиеся в символьной переменной, должны быть представлены в стандартном формате представления данных на внешних носителях (см. оператор % PACK без параметров), в противном случае выдается сообщение об ошибке.

При распаковке необходимо соблюдать соответствие между типом данных, упакованных в символьную переменную, и типом переменных, т.е. символьные данные должны заноситься в символьные переменные (массивы), а числовые – в числовые, в противном случае выдается сообщение об ошибке.

При распаковке символьного значения в символьную переменную количество символов, заносимых в символьную переменную, определяется наименьшей из длин.

Примеры правильного синтаксиса:

```
I) :I0%UNPACK W$ TO A,B$  ,C()
2) :I0%UNPACK L I$ (5,I00) TO B$ ()
```

2.7. Матричные операторы.

2.7.I. Общие сведения о матричных операторах.

Матричные операторы позволяют выполнять действия над целыми массивами (матрицами и векторами) согласно правилам линейной алгебры, а также обрабатывать (печатать, вводить значения и т.п.) символьные массивы.

При использовании матричных операторов следует выполнять следующие

правила;

1) каждый матричный оператор должен начинаться со слова MAT;

2) каждая переменная, которая используется в матричном операторе, должна быть массивом;

3) недопустимы многократные операции в одном матричном операторе, например, оператор MATA=B+C-D недопустим;

4) размерность массивов при выполнении матричных операторов может переопределяться явным или неявным образом, причем, массив с новыми размерами не может содержать элементов больше, чем их было в исходном определении;

5) все матричные операторы могут использоваться не только при счете по программе, но и в режиме непосредственного счета, за исключением оператора MATREAD.

2.7.2. Размерность массива .

С помощью матричных операторов можно обращаться как с числовыми, так и с символьными массивами. Правила языка BASIC требуют, чтобы были установлены размеры каждого массива . Обычно это делают в операторах DIM или COM до того, как его использовать в матричном операторе. Если размер не указан с помощью DIM или COM, то он автоматически получает размеры 10 * 10.

2.7.3. Изменение размерности массива.

Переопределение размерности массива матричными операторами бывает двух типов:

1) явное - в операторе задается новая текущая размерность, например: MATCON (5,5);

2) неявное - новая текущая размерность определяется в соответствии с текущей размерностью операндов и правилами действий с матричными операндами, например:

```
:10 DIM A(10,10),B (2,2),C(2,2)
.
.
: 40 MAT A=B+C
```

Текущая размерность матрицы (вектора), получаемой в результате выполнения матричных операторов, не может превосходить максимальную размерность этой матрицы (вектора).

2.7.4. Общие формы.

Матричные операторы обеспечивают 14 матричных операций. Общие формы матричных операторов представлены в таблице 4.

Таблица 4

Матричные операторы

Оператор	Назначение оператора	Пример	Примечание
MAT +	сложение матриц	MATA=B+C	***
MATCON	присвоение каждому элементу матрицы значения равного единице	MATA=CON	*
MAT=	присвоение элементам одной матрицы значений другой	MATA=B	***
MATIDN	приведение матрицы к единичной	MATA=IDN	*
MATINPUT	ввод элементов матрицы	MATINPUTA,B[X]	*,**
MATINV	вычисление матрицы, обратной данной, и определителя	MATA=INV (A),B	***
MAT *	умножение матриц	MATA=B * C	***
MATPRINT	печать матриц	MATPRINTA,B[X]	**
MATREAD	присвоение элементам матрицы значений из оператора DATA	MATREADA,B[X]	*,**
MATREDIM	переопределение размерности матрицы	MATREDIMA (X,Y)	*,**
MAT ()*	скалярное умножение матрицы на число	MATA=(K)*B	***
MAT -	вычитание матриц	MATA=B-C	***
MATTRN	транспонирование матриц	MATA=TRN (B)	***
MATZER	присвоение каждому элементу матрицы значения нуля	MATA=ZER	*

Примечания:

1) * - размерность переопределяется явным образом;

2) *** - размерность переопределяется неявным образом;

3) ** - применяется для числовых и символьных матриц (векторов); операторы, не отмеченные **, используются только для числовых матриц (векторов). В процессе выполнения матричных операторов могут возникать ошибочные ситуации при:

I) несоответствии размерностей операндов, в[illegible] операторе. В этом случае действия, заданные оператором, не в[illegible]ются, размерность и значения результирующего операнда не переопределяются;

2) вычислениях (например, переполнение или несоответствие типов матриц (векторов) и данных. В этих случаях размерность результирующей матрицы (вектора) не переопределяется, за исключением операторов MATREAD и MATINPUT, а ее значения переопределяются не полностью. При этом следует иметь в виду, что заполнение матриц идет построчно.

Примечание.

Ошибочные ситуации, возникшие при выполнении лишь отдельных операторов, описаны при рассмотрении этих операторов (например, сингулярная матрица в операторе MATINV).

2.7.5. Сложение матриц.

MAT <числовой массив> = <числовой массив> + <числовой массив>

<числовой массив> :: = <идентификатор переменной>[%]

оператор сложения матриц (MAT+) предназначен для сложения числовых матриц или векторов.

Оператор выполняет сложение матриц (векторов), указанных в правой части оператора, и присвоение значения суммы матрице (вектору), указанной в левой части. При этом размерность результирующей матрицы (вектора) изменяется в соответствии с размерностями матриц (векторов) слагаемых. Оба операнда, стоящие в правой части, должны иметь одинаковую размерность и один тип (целый или действительный). Один и тот же числовой массив можно указывать в обеих частях уравнения.

Примеры правильного синтаксиса:

I) :10 MATA = B+C

2) :10 MATA =A+B

3) :20 MATD %=B%+C%

Примеры:

```
I) :10 DIM A(2,3)
   :20 MATREADB (2,3),A
   :30 MATN =A+B
   :40DATA 1,2,3,4,5,6,7,8,9,8,7,6
```

В результате выполнения программы матрица N получает размерность (2*3).

2) :10 DIM A (2),B (3)
:20 MATREAD A,B (2)
:30 MATN =A+B
:40 DATA 3,4,-5,6

В результате выполнения программы матрица N получает размерность (2*I).

3) :10 DIM N(8),A%(2),B%(2,I)
:20 MATREAD A%,B%
:30 MATN =A%+B%
:40 DATA 3,4,-5,6

В результате выполнения программы вектор N получает размерность (2).

4) :10 DIM A(2),B (2)
:20 MATREAD A,B
:30 MATA=A+B
:40DATA 3,4,-5,6

2.7.6. MAT CON.

MAT <числовой массив> = CON [(<размерность> [, <размерность>])]

Оператор MATCON предназначен для присвоения всем элементам числового массива значений, равных единице. Кроме того, возможно явное переопределение размерности.

Если указан параметр, задающий размерность, то оператор переопределяет текущую размерность числового массива в соответствии со значениями, задающими размерность, и затем всем элементам массива присваивает единичное значение.

Примеры правильного синтаксиса:

1) :10 MATA=CON
2) :20 MATB%=CON (2,6)
3) :10 MATC=CON (I0)
4) :40 MATA=CON (5,7)

Примеры:

1) :10 DIM A (5)
:20 MATA =CON (3)
:30 MATB%= CON

В результате выполнения программы массив A получает новую текущую размерность (3), а массив B% - размерность (10* I0), и все элементы массивов A и B% с новой размерностью получают значения, равные единице.

2) :10 MATA=CON (2,2)
:20 MATPRINTA;

В результате выполнения программы на экран дисплея выводится:

I I

I I

2.7.7. Присвоение матриц .

MAT <числовой массив> = <числовой массив>

Оператор присвоения матриц (MAT=) предназначен для переписи значений из одного числового массива в другой. Оператор выполняет присвоение элементам числового массива, стоящего в левой части, значений элементов числового массива, стоящего в правой части.

При этом размерность массива, стоящего в левой части, приводится в соответствие с текущей размерностью массива в правой части.

Примеры правильного синтаксиса:

1) :10 MAT A=B

2) :20 MAT C=B%

Примеры:

1) :10 DIM A(3),B(10)
:20 MAT A = CON
:30 MAT N =A

В результате выполнения программы массив N приобретает размерность(3).

2) :10 DIM A (3)
:20 MAT A = CON
:30 MAT N =A

В результате выполнения программы массив N приобретает размерность (3*1).

3) :10 DIM N%(10)
:20 MAT A=CON (3,1)
:30 MAT N%=A

В результате выполнения программы массив N приобретает размерность(3).

2.7.8. MAT IDN

MAT <числовой массив> =IDN[(<размерность> [,<размерность>])]

Оператор MAT IDN предназначен для приведения числовой матрицы к единичной.

Если задан параметр <размерность>, то оператор переопределяет текущую размерность матрицы и затем приводит ее к единичной.

Новая текущая размерность должна задавать квадратную матрицу, в противном случае выдается сообщение об ошибке.

Если параметр <размерность> не задан, то матрица, стоящая в левой части (ее текущая размерность должна задавать квадратную матрицу), приводится к единичной, переопределения текущей размерности не происходит.

Примеры правильного синтаксиса:

1) :10 MATA = IDN

2) :10 MATE% = IDN (3,3)

Пример:

```
:10 DIM A(5,2)
:20  MATA=IDN (3,3)
:30 MATE%=IDN
```

В результате выполнения программы матрицы A и E% приводятся к единичным с текущей размерностью (3 * 3) и (10 * 10) соответственно.

2.7.9. MATINPUT

MATINPUT <список массивов>

<список массивов> ::= <идентификатор переменной>[%][(<AB> [, <AB>])]/ <идентификатор переменной>$[(<AB> [,<AB>])]/<список массивов>, <идентификатор переменной>[%][(<AB> [, <AB>])]/ <список массивов>, <идентификатор переменной>$(<AB> [, <AB>])].

Оператор MATINPUT предназначен для ввода значений в массивы с клавиатуры во время выполнения программы. Кроме этого, возможно явное переопределение текущей размерности массивов.

Когда встречается оператор MATINPUT, на экране дисплея появляется вопросительный знак (?) и система ждет ввода данных с клавиатуры для заполнения массивов, заданных в операторе. Заполнение массивов осуществляется построчно.

Размеры массивов определяются в программе (с помощью операторов COM, DIM или MAT), если не заданы новые размеры в операторе MATINPUT. При этом необходимо соответствие типов массивов и вводимых данных, т.е. числовым массивам должны соответствовать числовые данные, а символьным - символьные.

Данные, вводимые по оператору MATINPUT, должны разделяться запятой. Символьные данные могут вводиться как в кавычках, так и без них, за исключением случая, если символьная переменная содержит запятые или начальные пробелы - в этом случае использование кавычек обязательно. Данные, вводимые по оператору MATINPUT, могут быть числовыми или

символьными константами.

Данные по оператору MATINPUT могут быть введены в несколько приемов, т.е. после введения группы данных (через запятую), нажимается клавиша CR/LF; данные заносятся в массив, и если при этом массив не заполнен полностью, то на следующей строке индицируется знак ? и система ждет продолжения ввода данных.

Если при этом нажать клавишу CR/LF (без ввода данных), то остальные элементы массива не заполняются (сохраняют старые значения) и система переходит к следующему оператору программы. Если вводится избыточное количество данных, то лишние данные игнорируются.

Если в вводимых данных обнаружена ошибка (по нажатию клавиши CR/LF), то данные, начиная с ошибочного значения, следует ввести вновь. Данные предшествующие ошибке заносятся в массив.

Примеры:

1) :10 MATINPUT A,B%,C%(3,5), D (2)
2) :20 MATINPUT L% (Y ,X+P^3)Z +2

2.7.10. MAT INV .

MAT <числовой массив> =INV(<числовой массив>) [, <числовой элемент>] . Оператор предназначен для вычисления матрицы, обратной данной, и определителя исходной матрицы.

Оператор выполняет вычисление матрицы, обратной квадратной матрице, стоящей в правой части, и присваивает ее значению матрице, стоящей в левой части, при этом размерность матрицы в левой части становится равной размерности матрицы, стоящей в правой части.

Если задан параметр <числовой элемент> , то значение определителя исходной матрицы заносится в этот элемент.

В качестве параметра <числовая матрица> в правой части оператора должна стоять матрица с текущей размерностью, задающей квадратную матрицу. В левой части должна быть матрица действительного типа.

Матрица, стоящая в правой части, не должна быть сингулярной (или близкой к сингулярной), так как сингулярная (особенная) матрица не имеет обратной и ее определитель равен нулю. В случае, если в правой части стоит сингулярная матрица, то выдается сообщение об ошибке.

Кроме того:

1) могут возникнуть ошибки при вычислениях (в связи со сложностью вычислительной схемы), приводящие к переполнению или потере малых значений. В этих случаях следует проводить масштабирование исходной матрицы до выполнения инверсии ;

2) полученная обратная матрица может быть не точно обратной за счет ошибок округления при расчетах для вычисления обратной матрицы. Ошибку округления можно обнаружить с помощью вычисления остатков, которые определяются:
R=E-A*N , где E - единичная матрица, A - матрица, подвергаемая инверсии, N - обратная A матрица. При точной инверсии каждый элемент R будет равен нулю. Если N является хорошим приближением инверсии, то каждый элемент R будет небольшим.

Примеры правильного синтаксиса:

1) :10 MATA = INV (B), C
2) : 20 MATA = INV (L)
3) :20 MATA = INV (A), A (I,J)

Пример:

```
:10  MATREAD  A(4,4)
:20 MAT  N  =INV (A),  D%
:30 DATA  0,2,4,8,0,0,1,0     1,0,0,1,4, 8,16,32
```

В результате выполнения программы матрица N получает размерность (4 * 4).

2.7.11. Умножение матриц .

MAT <числовой массив> = <числовой массив> * <числовой массив>

Оператор MAT * предназначен для умножения числовых массивов. Оператор выполняет умножение массивов, указанных в правой части оператора, и присвоение значения произведения массиву, указанному в левой части оператора. При этом текущая размерность результирующего массива переопределяется в соответствии с текущей размерностью сомножителей.

Операнды, стоящие в правой части, должны иметь один и тот же тип.

Если оба операнда, стоящие в правой части - матрицы, то количество строк второй матрицы должно быть равно количеству столбцов первой. В противном случае выдается сообщение об ошибке.

Получаемые в результате умножения размеры результирующей матрицы определяются числом строк множимого и числом столбцов множителя.

Недопустимо употребление одного и того же массива в обеих частях уравнения.

Примеры правильного синтаксиса:

1) :100 MATA=B* C
2) :200 MAT% =B%*D%
3) :10 MAT% = L% * M%

Примеры:

```
1) :10 DIM  A (2,3), B (3,4)
   :20  MATREAD  A,B
   :30  MATN  = A* B
   :40  DATA 0,I,4,7,7,7,5.I,0,4,4,I,0,4,3,4,3,4
```

В результате выполнения программы массив N получает размерность (2 * 4).

```
2) :I0  DIM A (3,2), B (2)
   :20  MATREAD A,B
   :30  MATN  =A* B
   :40  DATA  0,I,4,7,3,2,I,I
```

В результате выполнения программы массив N получает размерность (3* I).

```
3) :I0 DIM  A (3), B(I,2)
   :20  MATREAD  A,B
   :30  MATN  =A* B
   :40  DATA I,2,3,4,5
```

В результате выполнения программы массив N получает размерность(3* 2).

```
4) :I0  DIM  A (3,2) B(2), C(I0)
   :20  MATREAD  A,B
   :3Q  MATC  =A*B
   :40  DATA  I,2,3,4,5,6,4,7
```

В результате выполнения программы вектор C получает размерность (3).

2.7.I2. MATPRINT .

MATPRINT [<устройство> ,] <список MATPRINT> [<разделитель>] <список MATPRINT> ::= <идентификатор переменной>[%]/<идентификатор переменной>[%] /<список массивов><разделитель> <идентификатор переменной>[%] / <список массивов><разделитель> <идентификатор переменной>[%]

Оператор MATPRINT предназначен для печати массивов (матриц и векторов), как числовых, так и символьных. Печать массивов осуществляется построчно, каждая строка печатается с новой печатной строки. Печать возможна в двух форматах:

I) зонном (основная форма);

2) уплотненном (если после идентификатора стоит;).

Если задан параметр <устройство>, то печать осуществляется на заданном устройстве; в противном случае - на устройстве, заданном оператором SELECT PRINT, если он был выполнен ранее, или на

консольном устройстве.

Печать осуществляется в порядке, заданном в операторе.

Примеры правильного синтаксиса:

1) :10 MATPRINT A,B%,C$;D

2) MATPRINT / 05,A$

Пример:

```
:10 DIM A$ (2,2), L %(3)
:20 MATL % = CON
:30 MATM = IDN (2,2)
:40 A$(1,1)="AB": A$(1,2)="CD": A$ (2,1)="EF": A$(2,2)="GH"
:50 MATPRINT A$ ,N,N; L %
```

В результате выполнения программы будет последовательная печать массивов: A$ в зонном формате, N в зонном формате, N в уплотненном формате и вектора L %:

```
AB          CD
EF          GH
I           0
0           I
I 0
0 I
I
I
I
```

2.7.13. MATREAD.

MATREAD <список массивов>

Оператор MATREAD предназначен для присвоения значений, содержащихся в операторах DATA, числовым и символьным массивам при счете по программе. Кроме этого возможно явное переопределение текущей размерности массивов.

По оператору MATREAD происходит заполнение массивов, заданных в операторе, данными из операторов DATA, при этом заполнение происходит построчно. Данные выбираются из оператора DATA в том же порядке, в каком они встречаются в данной строке программы. Если данных в операторе DATA не хватает, то берется следующий оператор DATA, если его нет, выдается ошибка, выполнение программы останавливается.

Необходимо соответствие типов массивов и данных. Если в программе используется несколько операторов MATREAD, то каждый из них начинает

выборку данных из операторов DATA для присвоения с того места, где закончил выборку предыдущий оператор MATREAD. Повторное использование значений из операторов DATA достигается при помощи оператора RESTORE, как и в операторе READ.

Примеры:

```
1) :10 DIM A (I), B (3,3)
   :20 MATREAD A,B (2,3)
   :100 DATA I,-2,3,4,5,0,0
```

В результате выполнения программы будут получены следующие массивы: A () =I и B(2,3)= -2 3 4
5 0 0

```
2) :10 DIM B(7),A$(2,2),AI%(3)2
   :20 MATREAD A$,B(3),AI% ,L% (I,I)I
   :30 DATA I,2,3,4,-I,-2,-3, "KL","BC", "AD"
   :40 DATA "P"
   :50 RESTORE 2
   :60 MATREAD M (3,2)
```

В результате выполнения программы будут получены следующие массивы:

```
A$=I 2 ; B()=-I; AI% =KL ; L% ()= (P) ; M() = 2 3
3 4      -2        BC                      4 -I
         -3        AD                     -2 -3
```

2.7.14. MATREDIM .

MATREDIM <список массивов>

Оператор MATREDIM предназначен для задания новой текущей размерности числовым и символьным массивам.

При выполнении оператора MATREDIM последовательно просматривается его список и перечисленные в нем массивы получают новую текущую размерность в соответствии с заданной. При этом новая текущая размерность не должна быть больше, чем максимальная (произведение всех параметров, ее определяющих).

Причем, матрица не может быть переопределена вектором, а вектор-матрицей.

Примеры правильного синтаксиса:

```
1) :10 MATREDIM A (4,5)
2) :20 MATREDIM B% (20)8,CI% (4,5), D$(5), L (X,Y),M% (A)X+Y
```

Примеры:

```
1) :10 DIM A(5),B% (10)2,C% (4,2)
   :20 MATREDIM A(4),B% (1),C% (1,20) I, D (2,3)
```

В результате выполнения программы массивы, определенные в операторе DIM или по умолчанию (10* 10) приобретают новую размерность.

2) :10 DIM A(5,5),B (5,5)
:20 MATINPUT A(2,2),B(2,2)
:30 MATREDIM A(5,2): MATPRINT A
:40 MATREDIM B(2,5): MATPRINT B

После ввода I,2,3,4,I,2,3,4 на экран дисплея будет выведено:

```
I  2
3  4
0  0
0  0
0  0
I  2  3  4  0
0  0  0  0  0
```

3) :10 DIM A%(2,4)2
:20 MATINPUT A%(2,4)2
:30 MATPRINT A
:40 MATREDIM A%(2,4)I
:50 MATPRINT A

После ввода: AB,CD,EF,I2,3X,YA, BS,$N первый оператор MATPRINT выведет на экран дисплея данные матрицы A:

AB CD EF I2
3X YA BS $N

Второй оператор MATPRINT выведет на экран дисплея данные в виде матрицы:

A B C D
E F I 2

При использовании оператора MATREDIM значения элементов матрицы сохраняются, возможно только перераспределение индексов элементов в соответствии с заданными в операторе параметрами размерности матрицы.

2.7.I5. Умножение матрицы на скаляр .

MAT <числовой массив> = (<AB>) * <числовой массив>

Оператор MAT ( )* предназначен для скалярного умножения числа, заданного арифметическим выражением, и матрицы (вектора).

Оператор выполняет присвоение элементам массива, стоящего в левой части, значений соответствующих элементов массива, стоящего в правой части, умноженных на значение арифметического выражения. Числовой массив, стоящий в левой части, получает размерность массива, стоящего в правой части. Один и тот же массив может участвовать в обеих частях уравнения.

Примеры правильного синтаксиса:

```
1) :10  MAT B% =(5^X+37)* B
2) :10  MAT L  =(3)* C%
3) :10  MAT L I =(-I)* L I
```

Примеры:

```
1) :10 DIM A(2,2)
   :20  MAT READ  A
   :30  MAT B%=(3)* A
   :40  DATA  1,2,3,4
```

В результате выполнения программы массив B% приобретает размерность (2 * 2).

```
2)  :10  DIM A%(3)
    :20   MAT A%=CON
    :30 MAT L =(-2) *  A%
```

В результате выполнения программы массив L приобретает размерность (3* I).

```
3) :10 DIM N (10)
   :20  MAT A=CON (4,I): N (2) =5:  A(2,I)=4
   :30  MAT N = (N(2))* A
```

В результате выполнения программы массив N приобретает размерность (4).

2.7.16. Вычитание массивов

MAT <числовой массив>= <числовой массив> - <числовой массив>

Оператор MAT - предназначен для вычисления разности числовых массивов. Оператор выполняет вычитание массивов,указанных в правой части оператора, и присвоение значения разности массиву, указанному в левой части.

Размерность массивов, указанных в правой части должна быть одинакова, в противном случае выдается сообщение об ошибке. Один и тот же массив может быть указан в обеих частях уравнения. Результирующий массив получает размеры, одинаковые с операндами.

Оператор MAT - отвечает тем же требованиям, что и MAT +, только элементы массивов не складываются, а вычитаются (см.п. 2.7.5).

Примеры правильного синтаксиса:

```
1) :10 MAT C=A-B
2) :20 MAT C=A-C
3) :30 MAT D = D-E
```

Пример:

```
:10 DIM D(3,3), E(3,3)
:20  MATINPUT  D
:30 MATINPUT  E
:40 MATF=D-E
:50  MATPRINT  F
```

Если допустить, что:

```
   I I I    3 3 3      -2 -2 -2
D()=I I I ,E()= 3 3 3, тогда F()= -2 -2 -2
   2 2 2       3 3 3              -I -I -I
```

2.7.17. MAT TRN

MAT <числовой массив> = TRN <числовой массив>

Оператор MATTRN предназначен для транспортирования числовых массивов. Оператор выполняет транспонирование числового массива (матрицы или вектора), указанного в правой части, и присваивает полученное значение числовому массиву, указанному в левой части оператора.

Не допускается использование одного и того же числового массива в правой и левой частях оператора.

Размерность результирующего числового массива переопределяется в соответствии с текущей размерностью транспонируемого массива.

Примеры правильного синтаксиса:

1) :10 MATA= TRN (B)
2) :20 MATL % = TRN (K)

Примеры:

1)
```
:10 DIM A (3,2)
:20  MATREAD  A
:30  MAT B%=TRN (A)
:40 DATA  1,1,2,2,3,3
```
В результате выполнения программы массив B% получает размерность (2*3).

2)
```
:10 DIM  A (1,2),B(10)
:20  A(1,1)=2: A(1,2)=3
:30  MAT B = TRN (A)
```
В результате выполнения программы массив B получает размерность (2).

3)
```
:10  DIM A(2)
:20 A(1)=2:  A(2)=3
:30  MAT B=TRN  (A)
```

В результате выполнения программы массив B получает размерность (1*2).

2.7.18. MAT ZER

MAT <числовой массив> = ZER [(<размерность> [,<размерность>])]

Оператор предназначен для присвоения всем элементам числового массива значений равных нулю. Кроме того, возможно явное перераспределение текущей размерности.

Оператор MATZER отвечает тем же требованиям, что и MATCON, только элементам присваиваются значения нуль, а не единица.

Примеры правильного синтаксиса:

1) :10 MATC = ZER (5,2)
2) :20 MATB = ZER
3) :10 MATA = ZER(F,T)
4) :10 MATD =ZER(20)

2.8. Операторы сортировки.

2.8.1. MATCONVERT.

MATCONVERT <метка числового массива> TO <метка символьного массива> , [(<AB> , <AB>)].

Оператор MATCONVERT предназначен для преобразования элементов числового массива в элементы <поля элементов> символьного массива в формате, удобном для использования операторов сортировки MATSORT и MATMERGE.

Каждый элемент числового массива занимает восемь байтов в элементе символьного массива, распределенных следующим образом:

знак	показатель степени	мантисса

Знак занимает пол-байта и содержит информацию о знаке показателя степени и мантиссы:

9- мантисса и показатель положительны;
8- мантисса положительна, показатель отрицателен;
1- мантисса и показатель отрицательны;
0- мантисса отрицательна, показатель положителен.

Показатель степени занимает два полубайта (восемь бит) и содержит значение показателя степени в прямом коде, если мантисса и показатель имеют один знак, и в дополнительном коде, если они имеют разные знаки.

Остальные шесть с половиной байтов занимают значение мантиссы в

прямом коде, если ее знак положителен, и в дополнительном, если знак отрицателен. Если длина элемента (поля элемента) символьного массива больше восьми байтов, то остальные (правые) байты каждого элемента (поля элемента) заполняются пробелами, т.е. HEX(20); если длина меньше восьми байтов, то младшие значащие цифры мантиссы отбрасываются.

Длина элемента (поля элемента) символьного массива должна быть не менее двух байтов, в противном случае происходит искажение информации.

Для реализации оператора необходимо, чтобы количество элементов в символьном массиве было не меньше, чем в числовом, в противном случае выдается сообщение об ошибке.

Реализация оператора заканчивается, если исчерпан числовой массив; при этом, если количество элементов символьного массива больше, чем числового, то элементы символьного массива, в которые не заносятся значения элементов числового массива, сохраняют старые значения.

Примеры правильного синтаксиса:

1) :10 MATCONVERT A () TO A$ ( )

2) :20 MAT CONVERT L() TO M$ () (3,5 * (X-Y))

Пример:

```
:10 DIM N(4),A$(2,2)3,B$(4)8
:20 N(1)=123: N(2)=-456: N(3)=.123: N(4)=0
:30 PRINT " числовая матрица,N "
:40 FOR I=1 TO 4: PRINT, N(I): NEXTI
:50 MATCONVERT N() TO A$()
:60 PRINT "символьная матрица" A$: "
:70 PRINT, : HEXPRINT A$(1,1): PRINT, :HEXPRINT A$(1,2)
:80 PRINT, : HEXPRINT A$(2,1): PRINT,: HEXPRINT A$(2,2)
:90 MAT CONVERT N() TO B$ () (2,3)
:100 PRINT "символьная матрица B$ ():"
:110 FOR I=1 TO 4: PRINT,: HEXPRINT B$(I): NEXT I
```

В результате выполнения программы на экран дисплея выводится информация:

```
числовая матрица N
                123
               -456
               .123
                  0
символьная матрица A$ :
```

902123
097543
898123
800000

символьная матрица, в X () :

20902123202020 20
2009754320202020
2089812320202020
2080000020202020

2.8.2. MAT$ORT

MAT$ORT <метка символьного массива> TO <метка символьного массива> , <метка символьного массива>

Оператор MAT$ORT предназначен для построения массива-локатора, используемого в операторе MATMOVE и содержащего координаты местоположения элементов сортируемого массива в порядке возрастания значений элементов.

Первый параметр <метка символьного массива> - сортируемый символьный массив, второй параметр <метка символьного массива> - рабочий символьный массив, третий параметр <метка символьного массива> - массив-локатор.

Массив-локатор одномерный или двумерный символьный массив должен иметь не меньшее количество элементов, чем сортируемый массив (в противном случае выдается сообщение об ошибке), длина каждого элемента этого массива должна быть два байта.

Рабочий массив - одномерный символьный массив с количеством элементов не менее, чем в сортируемом массиве, длина каждого элемента равна двум байтам, служит рабочей зоной оператора.

Сортируемый массив - любой символьный массив. Если в сортируемом массиве находятся одинаковые элементы, то порядок следования их координат в массиве-локаторе может не совпадать с расположением элементов в сортируемом массиве.

Реализация оператора заканчивается, когда просмотрен (отсортирован) весь исходный массив данных. Если количество элементов в массиве-локаторе больше, чем в сортируемом массиве, то в первый неиспользуемый элемент массива-локатора заносится признак его конца HEX (0000).

Пример правильного синтаксиса:

```
:10 MAT$ORT A$ () TO W2$(), L%()
```

Пример:

```
:10 DIM C$(3,4)2, W$(12)2, A$(1,1)12, B$(1,12)1
:20 A$(1,1)="CAPILJGBWDRE"
:30 MATREDIM A$(3,4)1
:40 MAT$ORT A$() TO W$(),C$()
:50 PRINT "сортировка окончена, получен массив-локатор":
:60 HEXPRINT C$()
:70 MATMOVE A$(), C$(1,1) TO B$(1,1)
:80 PRINT "отсортированный массив:"
:90 MATPRINT B$;
```

В результате выполнения программы на экран дисплея выводится информация:

```
Сортировка окончена, получен массив-локатор:
0102020401010302
0304010302030301
0104020203030201
отсортированный массив:
ABCDEFGHIJKL
```

2.8.3. MATMOVE

MATMOVE <метка числового массива> ,<элемент символьного массива> [<числовая переменная>] TO <элемент числового массива>
/ MATMOVE <метка символьного массива> [(<AB> , <AB>)] , <элемент символьного массива> [, <числовая переменная>] TO <элемент символьного массива> :

Оператор MATMOVE предназначен для поэлементной переписи информации из массива в массив (типы массивов должны совпадать), с упорядочиванием информации в соответствии с массивом-локатором; массив-локатор может быть получен при реализации операторов MAT$ORT или MATMERGE (массив-локатор, полученный при реализации оператора MAT$EARCH не может использоваться оператором MATMOVE).

Оператор осуществляет поэлементную перепись данных из массива источника, указанного до параметра TO, в массив приемника, указанного после параметра TO; массив приемника заполняется построчно,

начиная с элемента, указанного в операторе.

Порядок переписи данных задается массивом-локатором (элемент символьного массива), каждый элемент которого имеет длину два байта и определяет положение элемента в массиве источника (номер строки и номер столбца).

Если длина элемента (поля элемента), заданного двумя <AB>, символьного массива источника меньше длины элемента массива приемника, то элементы, записанные в массив приемника, имеют в конце пробелы; если длина элемента (поля элемента) символьного массива источника больше длины элемента массива приемника, то значение элемента усекается справа.

Для переписи некоторого количества элементов массива можно использовать параметр <числовая переменная>, который является счетчиком количества переписанных элементов (значение этой переменной должно быть присвоено пользователем до реализации оператора MATMOVE и находиться в диапазоне от 1 до 7999 включительно, дробная часть значения этой переменной при реализации оператора отбрасывается).

По окончании реализации оператора MATMOVE числовой переменной (если параметр указан в операторе) автоматически присваивается значение, равное количеству переписанных из массива в массив элементов.

Реализация оператора заканчивается, если:

1) исчерпан массив-локатор;
2) в массиве-локаторе обнаружен код HEX (0000);
3) заполнен массив приемника;
4) количество переписанных элементов равно значению параметра <числовая переменная>.

Возможно использование одного и того же массива в качестве массивов-локатора, источника и приемника, при этом результат может быть отличен от того, который был получен, если эти массивы имели бы разные идентификаторы (что обусловлено поэлементной пересылкой данных массива).

Просмотр массива-локатора начинается с элемента, указанного в операторе. Примеры правильного синтаксиса:

```
1) :10 MATMOVE A (),L(2) TO B (5)
2) :30 MATMOVE A%(), L%(I,3),M TO B%(I,2)
3) :100 MATMOVE P%() (3,2), L%(0) TO A2%(3)
```

Пример:

```
:10 DIM A$ (1,1)12,LM(1,1) 24,B$ (3,4)1
:20 A$ (1,1)="FAGINEJOCLBK"
:30 L$(1,1)=HEX(0102030303010204020201010103020101040203030402)
:40 MATREDIM  A$ (3,4)1, L$(3,4)2
:50 MATMOVE  A$(), L$(1,1)  TO B$ (1,1)
:60 MATPRINT  B$;
```

В результате выполнения программы в массиве B$ находится следующая информация:

```
ABCD
FPGH
IJKL
```

2.8.4. MATMERGE .

MATMERGE <метка символьного массива> TO <метка символьного массива> , <метка символьного массива> ,<метка символьного массива>

Оператор MATMERGE предназначен для создания массива-локатора, используемого в операторе MATMOVE для объединения нескольких упорядоченных в порядке возрастания файлов данных в один упорядоченный файл и содержащего координаты местоположения элементов сортируемого массива в порядке возрастания значений элементов.

Сортируемый массив должен быть символьным массивом с количеством строк и столбцов не более 254.

Одномерный массив воспринимается как один столбец (т.е. длина строки 1). Каждая строка сортируемого массива должна содержать упорядоченный в порядке возрастания файл данных.

Для реализации оператора необходимо, чтобы сортируемый массив имел более одной строки.

Первый параметр <метка символьного массива> - сортируемый символьный массив, второй параметр <метка символьного массива> - рабочий массив 1, третий параметр <метка символьного массива> - рабочий массив 2, четвертый параметр <метка символьного массива> массив-локатор.

Рабочий массив 1 - одномерный символьный массив; он должен иметь количество элементов на единицу больше, чем количество строк в сортируемом массиве, и длина каждого элемента должна быть один байт. Этот массив используется следующим образом: каждый байт, кроме последнего, содержит номер (в виде HEx-кода) элемента, подлежащего сравнению в соответствующей строке сортируемого массива. Последний элемент этого массива содержит код причины прекращения выполнения оператора: 00 - заполнен массив-локатор;

01-FE - номер строки, которая исчерпана. Когда строка исчерпана, то соответствующему ей элементу рабочего массива I присваивается код FF, показывающий, что элементы этой строки не участвуют в сравнениях по оператору MATMERGE.

Для реализации оператора MATMERGE рабочий массив I должен быть подготовлен пользователем - обычно INIT(01)WIX (), т.е. каждый элемент рабочего массива I должен содержать номер (в виде HEX-кода) элемента в соответствующей строке сортируемого массива, с которого начинается сортировка; значение последнего элемента рабочего массива I безразлично.

Рабочий массив 2 - является рабочей зоной для оператора и не используется пользователем.

Он должен быть одномерным символьным массивом с количеством элементов не менее, чем в сортируемом массиве и длиной каждого элемента, равной двум байтам.

Массив-локатор - символьный массив, с длиной элемента, равной двум байтам.

Чем больше в нем элементов, тем быстрее сортировка (меньшее количество операторов MATMERGE требуется для полной сортировки исходных файлов).

Реализация оператора заканчивается, если:

1) заполнен массив-локатор (признак - последний элемент рабочего массива I содержит код HEX (00);

2) исчерпана строка сортируемого массива (номер исчерпанной строки заносится в последний элемент рабочего массива I, а его элементу с номером, соответствующим номеру исчерпанной строки, присваивается код HEX(FF). Если массив-локатор заполнен не полностью, то в него заносится код HEX(0000).

По окончании реализации оператора MATMERGE необходимо использовать оператор MATMOVE для записи в выходной массив-буфер отсортированных данных в соответствии с массивом-локатором. Затем можно продолжить сортировку по оператору MATMERGE. В случае, если прекращение выполнения предыдущего оператора MATMERGE было вызвано тем, что исчерпана строка сортируемого массива, то перед началом следующей сортировки в эту строку можно ввести новый файл данных, подлежащих сортировке, и в соответствующий этой строке элемент рабочего массива I занести код HEX(01) (номер элемента, с которого начинается сортировка этой строки). При этом новый вводимый файл данных должен быть упорядочен не только внутри себя, но и относительно данных, находящихся в этой

строке ранее (например, части одного упорядоченного файла данных могут заноситься в освободившуюся строку сортируемого массива).

Пример правильного синтаксиса:

```
:10 MATMERGE A%() TO W1%(), W2%(), L%()
```

Пример:

```
:10 DIM A%(1,1)18,W1%(4)1,W2%(3)2,LL%(12)2,B%(20)1
:20 A%(1,1)="АБНЦЛРВСРЛЮДЕЖЮЯ"
:30 MATREDIM A%(3,6)1
:40 INIT (01) W1%()
:50 MATMERGE A%() TO W1%(),W2%(),L%()
:60 PRINT "окончен первый этап сортировки:"
:62 PRINT "содержимое рабочего массива 1:"
:70 HEXPRINT W1%()
:75 PRINT "содержимое массива-локатора:"
:78 HEXPRINT L%()
:80 MATMOVE A%(), L%(I), TO B%(I)
:85 PRINT "содержимое массива приемника:"
:90 MATPRINT B%
:110 MAT MERGE A%() TO W1%(),W2%(),L%()
:120 PRINT "окончен второй этап сортировки:"
:125 PRINT "содержимое рабочего массива 1:"
:130 HEXPRINT W1%()
:135 PRINT "содержимое массива-локатора:"
:140 HEXPRINT L%()
:150 MATMOVE A%(),L%(I), TO B%(13)
:160 PRINT "содержимое массива приемника:"
:170 MATPRINT B%
```

В результате выполнения программы на экран дисплея будет выведено:

```
окончен первый этап сортировки:
содержимое рабочего массива 1:
00
03
04
00
содержимое массива-локатора:
0101
0201
0202
0102

окончен второй этап сортировки:
содержание рабочего массива 1:
FF
06
06
01
содержимое массива-локатора:
0205
0304
0305
0106
```

```
0301                                    0000
0203                                    0203
0302                                    0302
0103                                    0103
0104                                    0104
0204                                    0204
0303                                    0303
0105                                    0105
содержимое массива приемника:          содержимое массива приемника:

A                                       A
B                                       B
C                                       C
D                                       D
E                                       E
F                                       F
G                                       G
H                                       H
I                                       I
J                                       J
K                                       K
L                                       L
                                        M
                                        N
                                        O
                                        P
```

2.9. Операторы ввода-вывода.

2.9.1. DATA LOAD BT

DATA LOAD BT [<устройство>,] <символьная переменная>

<устройство> ::= <логический номер> / <ФАУ>

<логический номер> ::= # <АВ>

<ФАУ> ::= <символ /> <элемент НГУ> / <символ /> <символьная переменная>. Оператор DATA LOAD BT предназначен для ввода информации с внешнего устройства в указанную символьную переменную.

Число вводимых байтов определяется длиной заданной символьной переменной, характер информации при этом не анализируется (т.е. данные это или служебная информация).

Примеры:

```
1) :10 DATA LOAD BT  A$
2) :10 DATA LOAD BT /08,B$()
3) :20 DATA LOAD BT #5,C$
```

2.9.2. DATA SAVE BT.

DATA SAVE BT [<устройство> ,] <символьная переменная>

Оператор DATA SAVE BT предназначен для вывода на внешнее устройство информации, сформированной в символьной переменной.

Вывод осуществляется без сопровождения какой-либо служебной информацией. Число выводимых байтов определяется длиной символьной переменной, заданной в операторе.

Примеры:

```
1) :100 DATA SAVE BT A$()
2) :150 DATA SAVE BT A1$()
3) :200 DATA SAVE BT#6,B$
4) :250 DATA SAVE BT/08, STR(A$, 3,10)
```

2.9.3. HEXPRINT

HEXPRINT [<устройство> ,] <список HEXPRINT>[<разделитель>]

<список HEXPRINT> :: = <элемент HEXPRINT> /<список HEXPRINT>

<разделитель> <элемент HEXPRINT>

<разделитель> :: = <символ,> / <символ ;>

<элемент HEXPRINT>::= <символьный аргумент> / TAB (<AB>)

Оператор HEXPRINT предназначен для индикации (печати) символьной информации в шестнадцатиричной форме. Пробелы, имеющиеся в символьном аргументе, индицируются (печатаются) как 20.

При использовании запятой в операторе HEXPRINT следующий за ней элемент индицируется (печатается) с начала другой строки.

Наличие точки с запятой между элементами HEXPRINT показывает, что информация выводится в уплотненном формате. Символы, не поместившиеся на этой строке, переносятся в начало следующей строки.

Использование функции TAB аналогично описанному в операторе PRINT (см.п.2.9.10).

Примеры правильного синтаксиса:

```
1) :10 HEXPRINT A$
2) :10 HEXPRINT A$,B$(1); $TR(C$,3,4)
```

3) :20 HEXPRINT A1K (); C% ,Z%()

Пример.

```
:10 A% ="ABC"
:20 HEXPRINT A% ;
:RUN
41424320202020202020202020202020
```

2.9.4. IF END THEN

IF END THEN <номер строки>

Оператор IF END THEN предназначен для определения состояния текущего файла данных в процессе считывания или обновления информации в нем. При считывании закрывающей записи файла система формирует специальный признак, который анализируется при выполнении оператора IF END THEN .

Оператором IF END THEN пользуются в случае, когда не известна длина файла данных; если закрывающая запись считана, управление передается строке с заданным номером, если нет -то оператору, следующему за IF END THEN .

Пример:

```
:10 DATA LOAD DC A,B,C%
:20 IF END THEN 40
:30 GOTO 10
:40 PRINT A,B,C%
```

2.9.5. % .

%<цепочка символов>

Оператор % используется вместе с оператором PRINTUSING для обеспечения ему формата для вывода (индикации, печати) информации. Оператор % содержит текст, предназначенный для печати и форматы, согласно которым выводятся элементы, заданные в операторе PRINTUSING.

В оператор % можно включать любые символы (кроме двоеточия), которые вставляются до и после описания формата. Описание формата должно содержать хотя бы один символ # или набор символов +, -, ., #, ^ , %% в определенной последовательности. Запятые и точка с запятой могут быть внесены в целую часть описания формата (но до точки десятичной дроби).

Примеры:

1) :10 % год=##,месяц=##,число=##

2) :20 %##,##ABC%,##,#,##ABC

3) :30 % +#.## ^^^^

4) :40 %##H.#H#H#^^^^.##

Оператор % используется для формирования ассемблерных программ. В этом случае цепочка символов должна представлять собой автокодные конструкции. В этом случае не допускается использование нескольких операторов % в одной строке.

2.9.6. INPUT .

INPUT [<символьная константа> ,] <список переменных> /

INPUT#<числовой элемент>

Оператор INPUT предназначен для ввода с клавиатуры числовых и символьных данных в процессе выполнения программы.

При выполнении оператора INPUT на экране дисплея появляется вопросительный знак (?). Пользователь вводит значения данных, соответствующие переменным в операторе INPUT и нажимает клавишу CR/LF. До тех пор пока не будут введены значения данных, на экране дисплея индицируется знак "?" .

Величины, введенные пользователем, последовательно присваиваются переменным в списке переменных оператора INPUT. Если более, чем одно значение введено на строку, то отдельные значения должны быть отделены запятыми. Данные можно вводить и в несколько строк.

Если количество значений, введенных по оператору INPUT меньше требуемого, т.е. клавиша CR/LF нажата до окончания ввода всего списка данных, то на следующей строке появится знак ?, означающий что программа ждет данных. Если они не будут заданы, то при очередном нажатии клавиши CR/LF осуществляется переход к выполнению оператора, следующего за оператором INPUT. Переменные в списке переменных оператора INPUT, которые не получили новые значения, остаются неизменяемыми.

Значение символьной переменной вводится:

1) в кавычках;

2) без кавычек;

3) в апострофах;

4) при помощи HEX-функции.

Если значение символьной переменной не заключено в кавычки при вводе, то запятые рассматриваются как окончание цепочки символов, а пробелы, стоящие перед значением, игнорируются. Если ведущие пробелы и запятые должны быть включены в цепочку символов, то цепочка символов должна быть заключена в кавычки.

Если при вводе данных на экране дисплея высветится сообщение об ошибке, то ввод данных должен быть повторен, начиная с ошибочного значения. Значения, которые предшествовали ошибке, воспринимаются.

В оператор INPUT можно включать различные сообщения, пояснения перед вводом данных с помощью параметра <символьная константа>.

Совместно с оператором INPUT могут быть использованы клавиши специальных функций.

Если клавиша CØ была определена для ввода текста (DEFFN') и система ожидает ввод (на дисплее высвечивается знак ?), то нажатие клавиши CØ вызывает ввод цепочки символов, связанной с этой клавишей.

Если клавиша CØ используется для вызова помеченной подпрограммы и система ожидает ввода, то нажатие клавиши CØ вызовет для выполнения подпрограмму. Когда в подпрограмме встречается RETURN, осуществляется возврат к началу оператора INPUT для его выполнения.

Если в операторе INPUT указан параметр %, то осуществляется ввод показания таймера, запущенного при инициации системы, в числовой элемент оператора INPUT.

Нажатие клавиши HALT/STEP прерывает выполнение программы после выполнения оператора INPUT.

Примеры:

```
1) :10 INPUT X,Y,Z%
2) :20 INPUT "данные" , A,B
3) :20 INPUT STR(A%, I,5)
4) :40 INPUT L%(3,4),B%(I,2)
5) :10 DEFFN' 01  "COLOR T.V "
   :20 INPUT A%
   :RUN
   ?    нажмите клавишу CØ  01
   ? COLOR  T.V.
6) :10 DIM A(30)
   :20  N=I
   :30  INPUT "количество", A (N)
   :40  N =N +I: GOTO30
   :50  DEFFN' 02
   :60   T=0
   :70  FOR B=I  TO N
   :80  T=T+A(B)
   :90  NEXT B
   :100 PRINT "итого="; T
```

```
:110 N =1
:120 RETURN
```

В процессе выполнения программы на экране дисплея индицируется:

```
количество
?8
количество
?10
количество
?15
количество
? (нажмите клавишу СП 02)
итого=33
количество
?
```

```
7) :10  INPUT #S1
   :20  READ A,B,C
   :30  READ Q
   :40  PRINT A*B/C+SQR(Q)
   :50  PRINT  Q*A+B
   :60  DATA  1,2,3,4,5
   :70  INPUT #S2
   :80   PRINT  "время счета=" ; (S1-S2)/2000; "С"
```

2.9.7. KEYIN.

KEYIN <символьная переменная> ,<номер строки> [,<номер строки>]

Оператор KEYIN предназначен для обработки информации, вводимой с клавиатуры в процессе счета по программе.

Если к моменту выполнения оператора была нажата клавиша, код нажатой клавиши (в соответствии с таблицей 5) записывается в первый байт символьной переменной, после чего происходит переход к строке, номер которой задан в операторе первым.

Таблица 5

наименование клавиши	Шестнадцатеричный код клавиши			
	РУС		ЛАТ	
	нижний регистр	верхний регистр	нижний регистр	верхний регистр
0	30	30	30	30
1	31	31	3F	31
2	32	32	32	32
3	33	33	33	33
4	34	34	34	34
5	35	35	35	35
6	36	36	36	36
7	37	37	37	37
8	38	38	38	38
9	39	39	39	39
.	2E	2E	2E	2E
EXP(-	2D	2D	2D	2D
SQR(+	2B	2B	2B	2B
LOG(/	2F	2F	2F	2F
ABS(*	2A	2A	2A	2A
пробел	20	20	20	20
; +	2B	38	2B	38
1 !	21	31	21	31
2 "	22	32	22	32
3 #	23	33	23	33
4 $	24	34	24	34
5 %	25	35	25	35
6 &	26	36	26	36
7 '	27	37	27	37
8 (	28	38	28	38
9)	29	39	29	39
0 :	3A	30	3A	30
- =	3D	2D	3D	2D
: *	2A	3A	2A	3A
. >	3E	2E	3E	2E
, <	3C	2C	3C	2C
/ ?	3F	2F	3F	2F

Продолжение табл. 5

Наименование клавиши	Шестнадцатеричный код клавиши			
	РУС		ЛАТ	
	нижний регистр	верхний регистр	нижний регистр	верхний регистр
ПИ -	DF	DF	2D	2D
STMT NUMBER	81	91	A1	B1
RUN	82	82	82	82
CLEAR	83	93	A3	B3
LIST	84	94	A4	B4
PRINT	86	96	A6	B6
LOAD	87	97	A7	B7
SAVE	88	98	A8	B8
ARC	2C	2C	2C	2C
TAN (;	3B	3B	3B	3B
SIN ((	28	28	28	28
COS ( )	29	29	29	29
Й J	CA	EA	6A	4A
Ц C	C3	E3	63	43
У U	D5	F5	75	55
К K	CB	EB	6B	4B
Е E	C5	E5	65	45
Н N	CE	EE	6E	4E
Г G	C7	E7	67	47
Ш [	DB	FB	7B	5B
Щ]	DD	FD	7D	5D
З Z	DA	FA	7A	5A
Х H	C8	E8	68	48
Ф F	C6	E6	66	46
Ы V	D9	F9	79	59
В W	D7	F7	77	57
А A	C1	E1	61	41
П P	D0	F0	70	50
Р R	D2	F2	72	52
О O	CF	EF	6F	4F
Л L	CC	EC	6C	4C
Д D	C4	E4	64	44

Продолжение табл.5

Наименование клавиши	Шестнадцатеричный код клавиши			
	РУС		ЛАТ	
	нижний регистр	верхний регистр	нижний регистр	верхний регистр
К V	D6	F6	76	56
З	DC	FC	7C	5C
Ч ^	DE	FE	5E	5E
С S	D3	F3	73	53
М M	CD	ED	6D	4D
И I	C9	E9	69	49
Т T	D4	F4	74	54
Ь X	D8	F8	78	58
Б B	C2	E2	62	42
Ю @	C0	E0	40	40
Я Q	D1	F1	71	51
LINE ERASE	9F	9F	9F	9F
CONTINUE	8A	8A	8A	8A
BACKSPASE	89	89	89	89
DELETE	9D	9D	9D	9D
RECALL	9B	9B	9B	9B
CR/LF	85	85	85	85
INSERT	9C	9C	9C	9C
ERASE	9E	9E	9E	9E
EDIT	9A	9A	9A	9A
0 вверх	AA	AA	AA	AA
1 <- - - -	AB	AB	AB	AB
2 <-	AC	AC	AC	AC
3 ->	AD	AD	AD	AD
- - - ->	AE	AE	AE	AE
вниз	AF	AF	AF	AF

Если была нажата одна из клавиш СИ, в первый байт символьной переменной помещается шестнадцатеричный код клавиши и происходит переход к строке, номер которой указан в операторе KEYIN вторым (см.пример 4).

Если клавиша не была нажата к моменту выполнения оператора, то выполнение программы продолжается с очередного оператора, следующего за оператором KEYIN .

Нажатие клавиши HALT/STEP и RESET прерывает выполнение программы.

Примеры:

```
1) :10  KEYIN  A$ ,100,200
2) :20 KEYIN  B$ (I), 100,100
3) :30 KEYIN  STR(A$ ,3,5),50,80
4) :10 DIM  A$I
   :20  KEYIN  A$ ,30,100:  GOTO  20
   :30  PRINT  A$ : GOTO  20
   :100 PRINT  "специальная функция";: HEXPRINT  A$ :GOTO 20
```

При выполнении строки 20, система ждет нажатия одной из клавиш. Соответствующий нажатой клавише символ индицируется на экране дисплея при выполнении строки 30 (при нажатии одной из клавиш CO осуществляется переход к строке 100).

2.9.8. LINPUT .

LINPUT [<символьная константа>,] [-] <символьная переменная>

Оператор LINPUT дает возможность пользователю вводить и корректировать символьные данные, включая пробелы, кавычки, запятые непосредственно в символьную переменную.

Длина символьной переменной не должна превышать 1000 символов.

Когда выполняется оператор LINPUT происходит следующее:

1) устанавливается режим редактирования;

2) индицируется значение символьной константы, если она задана в операторе;

3) появляется пробел, символ * и значение символьной переменной;

4) курсор устанавливается в положение первого символа, вызванного значения.

Пользователь может вводить новое значение или редактировать текущее значение символьной переменной.

Во время реализации оператора LINPUT движение курсора ограничивается длиной символьной переменной. Движение курсора за пределы отображения символьной переменной невозможно. Введенную строку символов заканчивают нажатием клавиши CR/LF.

Первоначальное значение символьной переменной можно возвратить до того, как была нажата клавиша CR/LF, нажатием клавиш LINE ERASE, EDIT и RECALL.

Знак " - " перед символьной переменной в операторе LINPUT является признаком отсутствия индикации символа " * " перед вызываемой на редактирование символьной переменной.

Примеры:

```
1) :10 DIM A$(30)
   :20 A$ ="программа"
   :30 PRINT "задача"
   .
   .
   :100 LINPUT "сообщение",A$
2) :10 LINPUT C$
3) :20 LINPUT "коррекция",B$()
```

2.9.9. PLOT.

PLOT [<устройство> ,] <список PLOT>

<список PLOT> ::= <элемент PLOT> / <список PLOT>,
<элемент PLOT> <элемент PLOT> ::= [<AB>]<символ < > [<AB>],
[<AB>], [<параметр PLOT>]<символ>>
<параметр PLOT> ::= <символьное значение> /U/D/ R/S/ C
используется целая часть арифметических выражений.

Второй параметр <AB> задает перемещение по оси X (в дальнейшем X), третий параметр <AB> задает перемещение по оси Y (в дальнейшем Y). Все целые части числовых параметров ограничены пределами 7999 < =//> =-7999.

Оператор PLOT предназначен для управления устройством графопостроителя и графического дисплея.

Перемещения пишущего узла графопостроителя по прямой от текущего положения задаются параметрами <AB>. Если один из двух параметров <AB>, задающих перемещение по осям X и Y (X и Y), или оба параметра не заданы, то по умолчанию значение параметра равно нулю.

X задает проекцию перемещения, как количество шагов по оси X; если x>0, то перемещение осуществляется вправо, если X<0, то влево. Y задает аналогично перемещение либо вверх, либо вниз. Оператор PLOT заданное перемещение по прямой аппроксимирует шагами и выводит на графопостроитель соответствующие управляющие НЕХ-коды.

Перемещение пишущего узла осуществляется в поднятом положении, если задан параметр U (вверх) или параметр вообще не задан, и

в опущенном, если задан параметр *D* (вниз) . Параметр *R* (восстановить) оператора PLOT вызывает перемещение пишущего узла в поднятом положении в начало координат. Параметры *R*, *U* и *D* действуют только в пределах одного элемента списка оператора PLOT.

Если в качестве параметра PLOT задано символьное значение, то перемещение, заданное одновременно X и Y, осуществляется в поднятом положении, а затем вычерчиваются символы, заданные в константе, или значение символьной переменной согласно таблице КОИ-8. Если задан параметр C, то X задает масштаб символов. При этом X должно быть положительным, а Y безразлично. Если задан параметр *S*, то X и Y задают промежуток между символами по горизонтали и вертикали соответственно. Параметры C и *S* действуют до их нового задания на один данный оператор PLOT. Если параметр C не задан, то масштаб символов равен единице. Если параметр *S* не задан, то промежуток между символами равен двум шагам по горизонтали.

Если в элементе PLOT задан первый параметр <AB> , то элемент списка исполняется повторно, а количество повторений задается целой частью <AB>, причем оно должно быть больше нуля, иначе выдается сообщение об ошибке. Если параметр <AB> не задан, то элемент списка PLOT исполняется один раз.

При программировании чертежа на графическом дисплее и графопостроителе следует учитывать размер рабочего поля. Выход пишущего узла за его пределы (переполнение координаты X или Y) влечет неправильное исполнение PLOT с параметром R.

Примеры:

1) PLOT 10<10,20,HEX(41)> , <,,R>
2) PLOT 5<20,20,HEX(41)> , <5,-50,U> ,5<10,10,HEX(43)> ,<,,R>
3) PLOT 10 <1,1,D>,<5,4 S>, <7,3,"ТЕКСТ"> ,<2,,C>,
   <7,3, "ТЕКСТ" > , <,,R>
4) PLOT <50,,> , <50,,D>, <-25,50, D> , <-25,-50,D>, <,,R>
5) PLOT <100,,D>, <,50,D>, <-100,,D>, <,-50,D>, <100,50,D>,
   <,-50,U> , <-100,50,D>,<,,R>

2.9.10. PRINT.

PRINT [<устройство>] / PRINT [<устройство> ,] [<список PRINT>][<список разделителей>]

<список PRINT> ::= <элемент PRINT> / <список PRINT>
<список разделителей> <элемент PRINT>

<список разделителей> :: = <разделитель> / <список разделителей> <разделитель>

<разделитель> ::= <символ,> / <символ ;>

<элемент PRINT> ::= <символьный аргумент> /<числовая переменная> / <AB> / TAB (<AB>)

Оператор PRINT предназначен для вывода (индикации,распечатки) числовой и символьной информации на заданное устройство.

Символьная константа выводится на экран в том виде, в котором она задана пользователем, но без кавычек и апострофов. Если символьная константа задана HEX-функцией (для обозначения специальных символов, которых нет на клавиатуре, а также кодов управления системой), то на экран выводятся изображения символов или осуществляется управления экраном (например HEX(03) очищает экран).

При обозначении символьной переменной в операторе PRINT, ее значение индицируется на экране (или печатается) с конечными пробелами.

По оператору PRINT выводится числовая информация, представленная числовой константой, числовой переменной или арифметическим выражением. Арифметическое выражение вычисляется в момент выполнения оператора PRINT, а затем выводится.

Числовые константы выводятся в двух формах: естественной и экспоненциальной. Вывод осуществляется в форме с естественной запятой, если значение находится в пределах от 1E-12 до 1E13 (нули впереди целого числа не индицируются, десятичная точка тоже), в противной случае - результат индицируется в экспоненциальной форме.

При выводе числового элемента по оператору PRINT индицируется пробел (подразумевается знак плюс) или знак минус, значение элемента в одной из двух форм и пробел после него.

Функция TAB в операторе PRINT используется для вывода информации, начиная с определенного знакоместа строки (отсчет ведется каждый раз с начала строки в пределах одной строки). Аргумент функции TAB может быть любым арифметическим выражением в пределах от 0 до 255. Дробная часть результата или его отрицательное значение во внимание не принимаются.

Если аргумент функции TAB больше заданной длины строки устройства, то информация выводится в начало следующей строки. Если аргумент функции TAB меньше номера знакоместа, на котором находится курсор в данный момент, то функция TAB не выполняется.

При указании в операторе PRINT массива, производится вывод значений элементов массива последовательно друг за другом. В случае,

когда в операторе PRINT массиву предшествуют переменные, тогда первая строка массива выводится последовательно за переменными в заданном формате; после вывода последнего элемента первой строки массива элементы следующей строки выводятся с новой строки, и каждая последующая строка массива выводится с новой строки. Если в операторе PRINT после массива заданы переменные, то они выводятся последовательно за элементами последней строки массива в заданном формате.

Элементы печати в операторе PRINT должны быть отделены один от другого разделителями : запятой или точкой с запятой.

Наличие запятой между элементами PRINT показывает, что информация выводится в зонном формате (длина зоны 16 символов). Элемент PRINT, перед которым имеется запятая, выводится в начало следующей зоны.

Если числовой элемент PRINT выходит за пределы последней зоны, то он полностью выводится в начало первой зоны следующей строки. Цепочка символов разделяется и частично переходит с одной строки на другую.

Наличие точки с запятой между элементами PRINT показывает, что информация выводится в уплотненном формате. Единственными пробелами между элементами печати в этом случае, являются те, которые входят в само значение. Элемент PRINT, не поместившийся в конце строки, полностью переносится в начало следующей строки. Исключением является вывод цепочки символов. Если точка с запятой следует за последним элементом печати в операторе PRINT, то возврат каретки не производится. Запятая и точка с запятой могут использоваться совместно, образуя смешанный формат.

В программе после ее запуска на счет (по оператору RUN) все операторы PRINT работают относительно некоторой позиции, которая и задает расположение информации на дисплее (печатающем устройстве).

Первый оператор PRINT всегда выводит информацию с начала строки. Другие операторы PRINT зависят от того, каким знаком (точкой или точкой с запятой) закончился предыдущий оператор PRINT и уже с учетом анализа выводят последующую информацию.

Если выполнение программы было остановлено по нажатию клавиш HALT/STEP или CONTINUE, то позиция обнуляется и при дальнейшем запуске программы расположение информации по следующим операторам PRINT будет уже другим.

Примеры:

1) PRINT /OC
2) PRINT /05, "ABC",A#,B
3) PRINT/05,,A,B; C
4) PRINT #4, "A+C=" ;D
5) PRINT ,A# ; B# ,D# ; : PRINT "P+R="; D,: PRINT 0.05
6) PRINT STP(A# , I,5);,,A# ; B# ; HEX((4I4I4I4I4I)
7) PRINT I5+8 +TAN (3);.0I54;-15,I0E+I2;:PRINTI5
8) PRINT TAB (5^2-3); "ответ="SQR (I7)

2.9.II. PRINT AT .

PRINT AT [<устройство> ,](<AB> , <AB>[, <AB>])

Оператор PRINT AT используется для установки курсора на экране дисплея в заданное положение и стирания экрана дисплея или его части, начиная с заданного положения и до конца.

Первый параметр <AB> указывает номер строки, второй параметр <AB> указывает номер знакоместа в строке, третий параметр <AB>указывает число стираемых символов на дисплее; если этот параметр отсутствует, то стирания символов не происходит.

Если значения параметров превышают размеры экрана дисплея, то производится установка курсора по максимально допустимым параметрам. После выполнения оператора PRINT AT информацию можно вводить в заданное положение операторами PRINT, PRINTUSING, INPUT и LINPUT.

Если количество символов, подлежащих стиранию больше, чем количество символов на текущей строке, то символы следующей строки также будут стираться. При использовании оператора PRINT AT в режиме непосредственного счета, и при этом,не вводя информацию на заданное знакоместо, следует учесть, что после исполнения оператора PRINT выполняется команда перевод строки /возврат каретки, двоеточие и курсор индицируются на следующей строке, а не на той, где ожидалось. Эти особенности оператора следует учесть пользователю при написании программ.

Примеры:

1) PRINT AT (3,5,I0) - в третьей строке, начиная с пятого символа стираются десять символов и курсор возвращается на пятое знакоместо.

2) PRINT AT (X,Y).

2.9.12. PRINTUSING.

PRINTUSING [<устройство> ,] <формат PU> [<список PRINT>] [<список разделителей>]

<формат PU>:: = <символьный аргумент> / <номер строки>

Оператор PRINTUSING используется для вывода числовой и символьной информации на устройство в заданном пользователем формате.

Формат задается следующим образом:

1) символьной переменной;

2) символьной константой;

3) оператором % , номер строки которого указан в операторе PRINTUSING.

Если формат задан в операторе % или переменной, то этим форматом могут пользоваться несколько операторов PRINTUSING.

Элементы PRINT из списка PRINT, указанные в операторе PRINTUSING, индицируются (печатаются) согласно их очередности.

Описание формата должно состоять хотя бы из одного символа или набора символов: + - . ,; # ¤ ^^^^ в определенной последовательности. Кроме этого, формат может содержать и текст.

Элементы печати в строке печати могут быть либо числовыми, либо символьными.

При выводе числовой информации каждое значение в операторе PRINTUSING преобразуется согласно заданному формату. Символы в формате заменяются на соответствующие цифры элемента печати в операторе PRINTUSING.

Для распечатки числовых значений используются три формата:

1) формат 1 - для целых чисел, например # ##;

2) формат 2 - для чисел в естественной форме, например ##,##;

3) формат 3 - для чисел в экспоненциальной форме, например #.**^^^^

При формировании формата чисел необходимо руководствоваться следующими правилами:

1) если в начале формата не стоит знак числа, т.е. первым символом является # или точка, а число является отрицательным, то индицируется знак минус (-), и число занимает на одно знакоместо больше, чем указано в формате;

2) если в формате обозначен знак плюс (+), то всегда индицируется знак числа - как положительного, так и отрицательного - непосредственно перед первой значащей цифрой;

3) если в формате обозначен знак минус (-), то знак индицируется только в случае отрицательного числа, в случае положительного числа вместо знака выводится пробел;

4) если задается формат I, то целая часть числа индицируется, а дробная часть (если она имеется) отбрасывается. Если число, которое должно индицироваться, короче формата(т.е. содержит больше символов # , чем цифр в значении), то оно заполняется впереди пробелами. Если число больше формата (т.е. цифр больше, чем символов #), то индицируется формат ;

5) если задается формат 2, то целая часть дополняется впереди пробелами, если она меньше формата, и индицируется формат на месте значения, если она больше формата; дробная часть отбрасывается, если она меньше формата и дополняется нулями, если она больше заданного формата;

6) если задается формат 3, то цифровое значение приводится к заданному виду и индицируется согласно формату, если показатель степени больше /99/, то на место значения индицируется формат;

7) если формат задан для нескольких чисел, а в операторе PRINTUSING значений задано больше, то формат используется до тех пор, пока все значения не будут выведены на дисплей;

8) если формат содержит знак $, то он индицируется перед первой значащей цифрой. Если число имеет знак плюс (+) или минус (-), то знак $ индицируется между знаком и первой значащей цифрой;

9) если формат содержит запятую (,) или десятичную точку (.), то эти символы выводятся на обозначенное знакоместо. Однако, если запятая предшествует первой значащей цифре, то она заменяется пробелом.

При индикации символьной информации каждый символ * в формате заменяется на указанный символ. Если значение символьной константы или переменной короче формата (т.е. символов ** в формате больше, чем символов в символьной константе или переменной), то оно дополняется в конце пробелами. Если значение больше формата, то индицируется его часть.

В одном операторе PRINTUSING можно задать несколько описаний форматов. Описание формата завершается пробелом, либо каким-нибудь другим символом, кроме символов, описывающих формат.

Если оператор PRINTUSING заканчивается запятой или точкой с запятой, то при выполнении операторе в программе или в режиме непосредственного счета эти знаки рассматриваются как точка с запятой, и последующая информация выводится на дисплей (печатающее устройство) непосредственно за указанным знаком, а не на следующей строке (см.

пример 8).

Примеры:

```
  1) :10 DIM A$ 20:X=2.3:   Y=27.123:  A$ ="ANGLE=-##.##
LENGTH=+##"
     :20 PRINTUSING  A$ , X,Y
     : RUN
     ANGLE=  2.30LENGTH= +27
  2) :10. A=1: B=2:PRINTUSING"*.#"      , A,B
     : RUN
     1.0
     2.0
  3) :10 A=5: PRINTUSING  30,A
     :30  % #
     :RUN
     5.
  4) :10 A=5.42: PRINTUSING  30,A
     : 30  % #.# ^^^^
     : RUN
     5.4E+00
  5) :10  X=1:  Y=2:   Z=3:  PRINTUSING  30,X;Y;Z
     :30  % #. #
     : RUN
     1.0  2.0  3.0
  6) :100 PRINTUSING  200
     :200 % год месяц число
     :RUN
     год месяц число
  7) :100 T=1946:  A$ ="год рождения":PRINTUSING  300,A$ ,T
     :300 % # # # # # # # # # # # # # # # #
     :RUN
      год рождения 1946
  8) :10 DIM  A$2
     .
     .
     :220 A$ =  "# #"
     :230  PRINTUSING  A$ ,N,  F,
     :240  PRINT D
```

После выполнения операторов строк 230 240 на экран дисплея выводится значения N, F и на этой же строке значение D сразу после F, а не в следующей зоне, как можно было бы предположить по аналогии с оператором PRINT .

2.IO. Дисковые операторы

2.IO.I. Основные сведения о хранении данных на дисках

2.IO.I.I. Форматизация диска

Перед первым использованием диска необходимо выполнить его форматизацию, т.е. записать на диск начальную структуру служебных данных. После форматизации повторять ее обычно не требуется. Форматизация диска стирает все ранее записанные на нем данные.

При форматизации формируются адреса секторов и данные для контроля, остальные байты секторов заполняются кодами HEX (00).

Для форматизации гибкого диска необходимо выполнять оператор:

:PRINT/I8,HEX(I0000I00000400); - для левого дисковода,

:PRINT/I8,HEX(I0000000000400); - для правого дисковода.

Для форматизации жесткого диска необходимо выполнять оператор:

:PRINT/IC,HEX(I0000000000400); - для фиксированного диска,

:PRINT/IC,HEX(I0001000000400); - для съемного диска.

2.IO.I.2. Размещение данных на дисках

Информация, выводимая на диск операторами DATA *SAVE* DC, DATA *SAVE* DA, представляет собой логическую запись, которая может состоять из одной или нескольких физических записей. Размер физической записи на диске определяется информативной емкостью одного сектора диска и равен 256 байтам. Эти 256 байтов при выполнении операторов DATA *SAVE* DC и DATA *SAVE* DA содержат управляющую информацию, автоматически формируемую машиной, и непосредственно данные.

Управляющие байты сектора содержат идентификатор физической записи, который занимает первый байт сектора, таким образом, для размещения данных и их управляющей информации отводится 255 байтов.

В этих 255 байтах последовательно друг за другом записываются значения со своими управляющими байтами. Каждому значению соответствуют два управляющих байта, расположенные непосредственно перед значением и содержащие информацию о типе (числовая или символьная) длине значения.

Значения (со своей управляющей информацией) должны располагаться в физической записи целиком, неполностью помещающееся значение полностью переносится в следующую физическую запись. Каждое значение переменной занимает на диске на два байта больше, чем необходимо

ему в соответствии с форматом его представления.
Пример:

```
:10 DIM A%(5)50,C%43,B%(3) 64
:20 DATA SAVE DA  F (100,6 )A% (),B%(),C%
:30 DATA SAVE DA  F ( 6, 6 )C%,A%(),B%()
```

при выполнении строк 20 и 30 одна и та же информация записывается на диск, но размещение по секторам различное (по строке 20 она занимает три сектора, а по строке 30 - два).

Расположение информации, выведенной по строке 20:

255 байтов

A%(I)	A%(2)	A%(3)	A%(4)	свободно	I00 сектор
52 байта	52 байта	52 байта	52 байта	47 байтов	

255 байтов

A%(5)	B%(I)	B%(2)	B%(3)	свободно	I0I сектор
52 байта	66 байтов	66 байтов	66 байтов	5 байтов	

255 байтов

C%	свободно	I02 сектор
45 байтов	2I0 байтов	

Расположение информации, выведенной по строке 30:

255 байтов

C%	A%(I)	A%(2)	A%(3)	A%(4)	свободно	I03 сектор
45 байтов	52 байта	52 байта	52 байта	52 байта	2 байта	

255 байтов

A%(5)	B%(I)	B%(2)	B%(3)	свободно	I04 сектор
52 байта	66 байтов	66 байтов	66 байтов	5 байтов	

Свободная область сектора заполняется кодами HEX (00).

2.10.1.3. Режимы работы с дисками

БЕЙСИК позволяет работать с дисковыми устройствами в двух режимах:

1) режиме дискового каталога файлов;
2) режиме прямой адресации секторов.

В режиме дискового каталога файлов система автоматически следит за размерами и размещением каждого каталогизированного файла программы или данных. Это упрощает программирование работы с дисками. Многие сложные операции обслуживания файлов, расположенных на дисках, берет на себя система.

Режим прямой адресации секторов позволяет работать с любыми секторами на диске путем задания адреса сектора, но пользователь должен сам следить за размерами и размещением информации на диске, что часто требует от него дополнительных программ.

2.10.2. Доступ к диску

2.10.2.1. Параметры диска

Перед выполнением операций записи и считывания данных системе необходимо указать диск, к которому требуется доступ. Для системы каждый диск является независимой логической единицей хранения и записи данных, причем секторы диска пронумерованы, начиная с нуля, независимо от нумерации секторов других дисков. Поэтому для определения местоположения любого заданного сектора необходимо указать системе диск, содержащий данный сектор. Для ссылки на каждый отдельный диск служат параметры идентификации диска. В системе используются три параметра - R, F и T, где для жесткого диска:

F - фиксированный;
R - съемный,

для гибкого диска:

F - правый (нижний);
R - левый (верхний).

Употребление параметра T означает текущий диск (R или F).

Доступ к определенному диску достигается путем включения соответствующего параметра диска в оператор доступа к диску, например:

:DATA SAVE DA F (100,4) AM

2.10.2.2. Адресация дискового устройства и носителей.

Кроме идентификации отдельных дисков, относящихся к какому-либо

дисковому устройству, в системе используется идентификация каждого дискового устройства в целом посредством задания определенного адреса устройства.

Дисковое устройство в целом задают параметром <устройство>, который в дисковых операторах необязателен. При его отсутствии используется консольное дисковое устройство, адрес которого выбирается из нулевой строки таблицы устройств.

Физический адреса (ФАУ=I8 в шестнадцатиричной системе) и тип диска Р устанавливаются автоматически при инициации системы или оператором CLEAR. ФАУ консоли и тип диска Р или R можно задать оператором SELECT DISK.

Дисковое устройство можно задавать также через таблицу устройств (см.п.2.I0.4), указывая логический номер (#) в параметре <устройство> . В этом случае ФАУ и тип диска, если он не задан в дисковом операторе, выбираются из соответствующей строки таблицы устройств по заданному номеру файла.

Тип диска можно задать в дисковом операторе либо в операторе SELECT #. Если в операторе SELECT # тип диска не задан, то подразумевается Р.

Если в дисковом операторе тип диска определяется параметром Т, то он выбирается из соответствующей строки таблицы устройств.

В операторах работы с открытыми файлами DATA SAVE DC, DATA LOAD DC, DBACKSPACE, DATASAVE DC CLOSE тип диска не задают так как он приписан заданному в этих операторах номеру файла и был уже задан в операторах открытия файлов: DATA SAVEDC OPEN и DATA LOADDCOPEN по правилу, описанному в данном подпункте.

2.I0.3. Операции в режиме дискового каталога

2. I0.3.I. Автоматическое составление каталога именованных файлов.

В режиме дискового каталога используется I6 операторов, осуществляющих управление файлами. Эти операторы дают возможность создавать файлы программ и данных на диске, а также обращаться к ним по имени файла (а не путем задания адресов секторов). Каждый вновь создаваемый файл автоматически помещается в свободную область диска; имя, адрес и другая необходимая информация о файле записываются в специальную зону диска – указатель каталога, и используются при дальнейшей работе с этим файлом. Кроме того, в системе предусмотрены дополнительные возможности работы с

файлами. Такие, как переходы от записи к записи в пределах файла, копирование файлов и выдача основных характеристик файла.

Структура хранения файлов на диске называется каталогом, который состоит из двух зон:

1) зоны хранения каталога, где хранятся все файлы;

2) зоны указателей каталога, где хранится вся справочная информация о каждом файле.

Запись файлов в зоне каталога осуществляется последовательно: сектор за сектором, а их имена и адреса заносятся в указатель каталога.

Обращение к файлу осуществляется следующим образом: в зоне указателей каталога выполняется поиск имени (идентификация имени производится по восьми байтам) и по нему выбирается начальный адрес этого файла в зоне каталога. Поэтому пользователю не надо заботиться о выяснении действительного адреса каталогизированного файла. Система обновляет указатель каталога при записи нового файла и проверяет данные в каталоге при каждом обращении к существующему файлу.

Признаком режима дискового каталога является буквы DC (DISK CATALOG),стоящие после имени оператора.

2.10.3.2. Возможности каталоговых операций

В режиме дискового каталога используются операторы, которые обеспечивают пользователю следующие возможности работы:

1) составление каталога, состоящего из указателей и зоны хранения на диске;

2) запись и хранение программ по их именам без ссылок на адреса секторов. Загрузка программ с дисков в память без ссылок на адреса секторов;

3) открывание и повторное открывание файлов данных на диске по имени файла без ссылки на адрес сектора;

4) запись данных в открытый файл без ссылки на адрес сектора;

5) автоматическое занесение многосекторных записей на диск в соответствии с запросом по списку аргументов;

6) переход (от записей к записям) в прямом и обратном направлениях в пределах файла;

7) полный листинг содержимого указателя каталога;

8) изъятие ненужных файлов с повторным использованием освобожденных секторов;

9) копирование полного содержимого каталога (указатель каталога и зона хранения каталога) с одного диска на другой.

2.10.3.3. Создание каталога

Перед записью информации на диск в режиме дискового каталога необходимо создать каталог, для этой цели используют оператор SCRATCH DISK, который устанавливает зоны указателя каталога и каталога (если диск был подготовлен ранее, то старая информация в указателе каталога стирается).

Зона указателя каталога всегда начинается с нулевого сектора. Для задания величины зоны указателя в операторе SCRATCH DISK используется параметр LS, если он не указан, то зона указателя каталога – 21 сектора.

Зона каталога начинается всегда с сектора, следующего за зоной указателя каталога (за последним сектором зоны указателя каталога). Конец зоны каталога – адрес последнего сектора зоны каталога, задается пользователем в операторе SCRATCH DISK параметром END. Этот адрес не должен превосходить адреса последнего сектора на диске, иначе выдается сообщение об ошибке (ERR 80 CE).

При задании размера зоны указателя каталога следует иметь в виду, что в нулевом секторе можно поместить информацию о 15 файлах, а в любом другом о 16 файлах (имя файла занимает 8 байтов). Зону указателя каталога невозможно изменить (уменьшить или увеличить) без реорганизации всего каталога, поэтому для нее надо отводить достаточно места с учетом возможности появления дополнительных файлов.

После создания каталога на диске его можно использовать для работы с операторами режима дискового каталога файлов для записи, считывания и управления файлами данных в программе.

2.10.3.4. Листинг указателя каталога. LIST DC

Имена всех каталогизированных файлов на диске записаны в указателе каталога вместе с адресом каждого файла и с дополнительной информацией о файле. Список входных записей в указателе каталога можно получить с помощью оператора LIST DC. В операторе LIST обязательно должен быть параметр DC, в случае его отсутствия система попытается выполнить оператор LIST, по которому расшифровывается программа, находящаяся в памяти, а не указатель каталога.

LIST DC <тип диска> <параметр LIST DC>

<параметр LIST DC>:: =<устройство> /<устройство>,Символьный аргумент>

Оператор LIST DC предназначен для индикации (распечатки) содержимого зоны указателя каталога на дисплее (или печатающем устройстве, задаваемом оператором SELECT LIST).

Параметры <устройство> и <тип диска> задают диск, данные из указателя каталога которого выводятся по оператору LIST DC.

При выполнении данного оператора на заданное устройство выводится следующая информация:

1) количество секторов, отведенных под зону указателя каталога (INDEX SECTORS);

2) адрес последнего сектора, отведенного под зону каталога END CAT.AREA);

3) адрес сектора, следующего за последним использованным сектором из зоны каталога (CURRENT END);

4) информация по каждому каталогизированному файлу;
   имя файла (NAME),
   состояние файла (S - ликвидируемый файл) (в графе TYPE),
   тип файла (P- программа, D - данные) (в графе TYPE),
   начальный адрес сектора файла (START),
   конечный адрес сектора файла (END),
   количество секторов, использованных в файле (USED).

Наличие в операторе символьного аргумента позволяет просматривать каталог с обозначенным условием. Например, если A$ = " M" то на дисплее (печатающем устройстве) будут индицироваться имена только тех файлов, у которых на третьем знакоместе стоит буква М. При сравнении имени файла с маской (значением символьного аргумента) пробел в маске означает любой символ в имени файла. Если длина символьного аргумента больше восьми байтов, то маской служат ее первые восемь байтов.

В случае, если значение символьного аргумента не определено, просматривается (листуется) весь каталог.

Примеры:

1) LIST DC F A$

2) :10 B$ ="ABC"
   :20 LIST DC R B$

в результате выполнения этих операторов на дисплее индицируются те имена файлов, в которых на первых трех знакоместах буквы ABC.

3) LIST DC P

4) LIST DC P/OC

2.10.3.5. Создание каталогизированного файла данных на диске.

DATA SAVE DC OPEN

Система не может знать, сколько записей будет занесено в файл, а также длину записей, поэтому пользователь должен сам определить количество секторов, необходимых для каждого файла. Системе должны быть выданы указания относительно резервирования достаточного для файла места на диске. Таким образом, запись файла на диск должна выполняться в два этапа:

1) файл данных нужно занести в каталог (или создать файл) по оператору DATA SAVE DC OPEN. В данном операторе указывается имя нового файла данных и количество секторов, которое нужно зарезервировать для данного файла. На данном этапе запись данных в файл отсутствует;

2) после открытия файла можно заносить записи данных в файл путем выполнения оператора DATA SAVE DC.

Как уже говорилось выше, создание файла данных на диске выполняется по оператору DATA SAVE DC OPEN.

DATA SAVE DC OPEN <тип диска>[X] [<устройство>,] <параметр SAVE OPEN>

<параметр SAVE OPEN>:: =([<резерв>]<символьный аргумент> / TEMP, <AB> ,<AB>

<резерв> ::= <AB>/ <символьный аргумент>

Параметр <AB> в параметре <резерв> задает количество секторов, резервируемых на диске под файл; параметр <символьный аргумент> в параметре <резерв> определяет имя файла, отмеченного как ликвидируемый (по оператору SCRATCH), на место которого будет помещен новый файл; параметр <символьный аргумент> в параметре <параметр SAVE OPEN> задает имя нового файла; X - параметр контрольного считывания данных после записи; TEMP - параметр организации временного рабочего файла.

Оператор DATA SAVE DC OPEN предназначен для резервирования секторов под вновь создаваемый файл данных и занесения информации об этом файле в зону указателя каталога. Этот же оператор применяют для создания временных рабочих файлов, располагаемых за зоной каталога (если задан параметр TEMP) или для повторного использования секторов файла, отмеченного как ликвидируемый.

Каждый файл открывают первоначально отдельно оператором DATA SAVE DC OPEN; для открытия нескольких файлов данных им необходимо присваивать разные номера файлов.

Параметр V означает выполнение контрольного считывания после записи на диск.

Имя вновь открываемого файла должно отличаться от имен всех файлов, размещенных в каталоге, но допустимо совпадение имен вновь открываемого файла и ликвидируемого (т.е. на месте старого файла можно записать новый с тем же именем).

Параметр TEMP означает создание временного рабочего файла. Временные файлы не заносятся в каталог и не могут быть размещены в зоне каталога на диске. В случае необходимости во временных файлах следует оставить для них достаточное количество секторов вне зоны каталога.

Параметры <AB>, заданные вместе с параметром TEMP задают начальный и конечный адреса секторов зоны, резервируемой для временного файла. Если значение начального адреса сектора меньше значения адреса последнего сектора зоны каталога, то выдается сообщение об ошибке (ERR 77).

Количество секторов, резервируемых под файл, должно быть хотя бы на единицу больше требуемого, так как последний сектор файла каталога отводится для служебной информации (для формирования концевой записи).

Примеры:

1) :10 DATA SAVE DC OPEN R(ID) "ДАННЫЕ 1"
2) :20 DATA SAVE DC OPEN P#1 ("ДАННЫЕ 1") "ДАННЫЕ 2"
3) :100 DATA SAVE DC OPEN P TEMP,1000,2000

2.10.4. Селектирование дисковых устройств и работа с открытыми файлами.

2.10.4.1. Таблица устройств

При выполнении любого дискового оператора системе сразу же требуются данные двух видов: тип диска, к которому требуется доступ, и дисковое устройство, в котором размещается данный диск. Данные первого вида представляются в виде параметров P или R в самом операторе. Поскольку в системе может быть несколько дисковых устройств, то для системы необходимо задать устройство, содержащее данный диск. Для идентификации устройств служит, как уже упоминалось ранее, приписываемый каждому устройству определенный адрес (ФАУ).

Возможны три способа задания адреса дискового устройства:

1) прямой способ - непосредственное включение ФАУ в оператор;

2) косвенный способ - включение в оператор ссылки на логический номер, связанный с соответствующим ФАУ;

3) неявный (по-умолчанию) к консольному устройству.

При неявном способе задания адреса дискового устройства неуказываемый адрес консольного устройства записывается системой в специально отводимой части памяти, называемой таблицей устройств. При выполнении любых операторов система обращается к таблице устройств для выборки адреса дискового устройства (если адрес не был задан в самом операторе).

Таблица устройств представлена в памяти системы в виде восьми строк (0 – 7), каждой из которых присвоен номер файла. Неуказываемый адрес консольного устройства записывается в нулевой строке таблицы.

Каждая строка таблицы устройств содержит следующую информацию:

1) номер файла (тождественно равен номеру строки и логическому номеру устройства);

2) физический адрес дискового устройства (ФАУ), на диске которого размещен файл с данным номером, и тип диска;

3) адрес начального сектора файла с данным номером;

4) адрес последнего сектора файла с данным номером.

При инициации системы автоматически заполняется нулевая строка таблицы устройств.

Возможно хранение адреса устройства в соответствующих строках таблицы и приписании любому логическому номеру файла. В этом случае ФАУ устройства должен быть записан в таблицу при выполнении оператора SELECT#, в том числе и SELECT#0.

Пример:

:50 SELECT#3 I8,#5, I8

Дисковому устройству с ФАУ I8 присваиваются логические номера 3 и 5 и отводятся третья и пятая строки таблицы устройств.

Теперь при выполнении оператора:

:60 LOADDOS#3 "PROG2"

будет загружаться программа с именем PROG2 с фиксированного диска дискового устройства, описанного в третьей строке таблицы устройств.

2.I0.4.2. Типы данных, необходимых для доступа к файлу

Для доступа к файлу системе необходимы следующие данные о файле:

I) тип диска и адрес дискового устройства, на котором расположен файл;

2) адреса начального, текущего и конечного секторов файла.

Эти данные переписываются из указателя каталога на диске в таблицу устройств каждый раз, когда файл данных открывается на диске впервые (оператор DATA SAVE DC OPEN) или повторно (оператор DATA LOAD DC OPEN). Адрес текущего сектора устанавливается равным адресу начального сектора файла. Файл данных открыт, если информация о нем занесена в таблицу устройств.

2.10.4.3. Работа с несколькими открытыми файлами.

Понятие открытого файла данных было введено ранее в п.п. 2.10.3.5, поэтому назначение таблицы устройств связано со спецификой работы системы с множеством открытых файлов данных на дисках.

Предположим, что необходимо открыть файл "ДАН" на диске типа F. В этом случае можно использовать оператор:

```
:200 DATA SAVE DC OPEN F(100), "ДАН"
```

После выполнения данного оператора открытым является файл с именем "ДАН", расположенный на фиксированном диске консольного дискового устройства. Информация о нем содержится в нулевой строке таблицы устройств.

Если затем выполнен оператор:

```
:200 DATA SAVE DC OPEN F(50) ,"ДАННЫЕ 2",
```

то автоматически закрывается файл с именем "ДАН" и открывается файл с именем "ДАННЫЕ 2", так как параметры файла "ДАННЫЕ 2" запишутся в нулевую строку таблицы устройств (вместо параметров файла "ДАН").

Таким образом, при записи каждого файла стирается информация о предыдущем файле.

Для открытия нескольких файлов данных одновременно необходимо использовать разные строки таблицы устройств. Перед тем, как использовать другие строки таблицы для открытия новых файлов, необходимо записать в эти строки физические адреса дисковых устройств по оператору SELECT # .

Например, если необходимо скопировать данные из файла "ДАННЫЕ I" в файл "ДАННЫЕ 2" на одном и том же фиксированном диске дискового устройства с ФАУ 18, можно выполнить следующую программу (в предположении, что в каждой логической записи находится 50 числовых величин).

```
:10 DIM A(50)
:20 SELECT #3 18, #5 18
:30 DATA SAVE DC OPEN F #3 (50)"ДАННЫЕ 2"
:40 DATA LOAD DC OPEN F #5 "ДАННЫЕ I"
```

```
:50 DATA LOAD DC #5 ,A()
:60 DATA SAVE DC #3,A()
:70 IF END THEN100
:80 GOTO 50
:100 DATA SAVE DC #3,END
```

Вся информация необходимая системе для работы с множеством открытых каталогизированных файлов размещается в таблице устройств. Эта информация автоматически переписывается из указателя каталога на диске в таблицу устройств каждый раз, когда файл данных открывается впервые или повторно при выполнении оператора DATA SAVE DC OPEN или DATA LOAD DCOPEN.

Один и тот же файл можно открывать многократно, используя каждый раз различные номера файлов; при этом в каждой строке таблицы устройств можно записывать параметры только одного файла.

Файл считается открытым, если его параметры записаны в строке таблицы устройств, а закрытым, если его параметры взяты из той таблицы.

Возможны следующие закрытия открытых файлов:

1) присвоение номера данного файла другому файлу. В этом случае параметры нового файла запишутся в таблицу устройств на место параметра закрываемого файла;

2) при выполнении оператора DATA SAVE DC CLOSE в выделенную строку (определяемую параметром <логический номер>)таблицы устройств заносятся нули вместо адресных параметров секторов файла, при этом адрес устройства сохраняется;

3) выполнение оператора CLEAR без параметров или инициализация системы.

2.10.5. Эффективное использование дисков

2.10.5.1. Организация временных рабочих файлов на диске

Временные рабочие файлы могут быть полезными при выполнении разнообразных операций обработки данных. Временный рабочий файл открывается с помощью оператора DATA SAVE DC OPEN, но в отличие от каталогизированного файла, данные о нем не заносятся в указатель каталога и не записываются в зоне каталога на диске. Но параметры временного файла записываются в таблице устройств в оперативной памяти.

Временные рабочие файлы можно использовать как зону, предназначенную для записи данных, которые не нужны в постоянном файле.

Конец зоны каталога задается оператором SCRATCH DIS K при создании каталога. При необходимости использования временных файлов нельзя отводить под каталог целый диск. Несколько секторов необходимо оставить вне зоны каталога для размещения временных рабочих файлов.

Для открытия временных файлов и для доступа к ним используются те же самые операторы, которые служат для работы с каталогизированными файлами. Так как временным файлам нельзя присваивать имена, то вместо имени файла используется специальный параметр TMP, который должен быть задан в операторе DATA SAVE DC OPEN при первичном открытии временного файла, а затем в операторе DATA LOAD DC OPEN при повторном открытии файла.

Временным файлам, как и каталогизированным, можно присваивать номера файлов (логические номера), обеспечивая возможность открытия нескольких временных файлов одновременно. Данные записываются во временный файл точно так же, как они записываются в каталогизированный файл; для временного файла необходимо резервировать, по крайней мере, на один сектор больше по сравнению с количеством секторов, действительно необходимых для записи данных. Временный файл закрывается точно так же, как и каталогизированный файл.

2.10.5.2. Изменение зоны каталога

Верхний предел зоны каталога на диске устанавливается первоначально параметром END оператора SCRATCH DIS K при создании каталога. Если возникает необходимость в увеличении места для размещения временных файлов или в увеличении или уменьшении количества секторов, отводимых для каталогизированных файлов, то можно изменить величину зоны каталога с помощью оператора MOVE END. Данный оператор изменяет только зону каталога, не изменяя размер указателя каталога. Зону каталога можно расширять и сжимать, но наибольший адрес сектора не должен превышать максимально возможный адрес сектора на диске.

2.10.5.3. Повторное использование ликвидируемых файлов

Один из способов максимального использования дисков для записи файлов состоит в повторном использовании секторов, занятых ликвидируемыми файлами. Единственный способ уничтожения ликвидируемых файлов заключается в выполнении оператора MOVE , который удаляет их при копировании каталога с диска на диск. Однако , в ряде случаев, проще и эффективнее записывать новые файлы программ и данных

непосредственно в секторах, занятых ликвидируемыми файлами, не корректируя каталог. Новые файлы записываются в секторах в этом случае с помощью операторов DATA $AVE.DC OPEN и $AVE.DC . Тип ликвидируемого файла (программа или данные) не имеет значения при открытии на его месте нового файла.

2.10.5.4. LIMITS

LIMITS <тип диска> <файл> <числовой элемент> , <числовой элемент> , <числовой элемент> [, <числовой элемент>]

<файл> :: = <ФАУ> ,<символьный аргумент>,/[<логический номер> ,] [<символьный аргумент> ,]

Оператор LIMITS предназначен для получения пользователем информации о местоположении файла и количестве секторов, занимаемом файлом на диске.

При выполнении каталоговых операций оператор LIMITS оказывается полезным для слежения за количеством свободных секторов файла во время данных.

Различают две формы оператора:

1) параметр <символьный аргумент> задан в операторе;

2) параметр <символьный аргумент> не задан в операторе.

Если в операторе задан параметр <символьный аргумент> , то данные о файле с заданным именем из указателя каталога переписываются в переменные, при этом значения параметров <числовой элемент> становятся равными соответственно начальному адресу сектора в файле, конечному адресу сектора в файле и количеству используемых секторов в файле. Имя файла задается параметром <символьный аргумент> . Если задан четвертый параметр <числовой элемент>, то в него заносится состояние файла:

1) 2 – файл данных;

2) 1 – файл программ;

3) 0 – файла нет (содержимое числовых элементов 1,2,3 устанавливается равным 0);

4) 3 – вычеркнутый файл программ;

5) 4 – вычеркнутый файл данных;

6) 5 – транслированный файл программ;

7) 6 – защищенный файл программ;

8) 7 – транслированный и защищенный файл программ.

Для реализации этой формы оператора LIMITS необходимо, чтобы файл имел запись окончания файла (сформированный при выполнении оператора DATA $AVE.DC END), так как только при ее формировании

определяется количество секторов в файле в указателе каталога.

При реализации оператора LIMITS с заданным именем файла перепись информации из указателя каталога в числовые элементы идет через таблицу устройств (номер строки таблицы устройств, через которую идет перепись информации, задается параметром <логический номер>, если он не задан, то через строку 0).

Таким образом, параметры файла в этой строке таблицы будут испорчены. Поэтому при выполнении этого оператора не рекомендуется осуществлять перепись через строку таблицы устройств, приписанную открытому файлу, так как иначе информация о нем будет утеряна и следующие обращения к нему (особенно по оператору DATA SAVE DC) могут привести к искажению информации в файле.

Если в операторе LIMITS параметр <символьный аргумент> не задан, то при реализации оператора цифровым элементам присваиваются значения начального, конечного и текущего адресов сектора открытого файла из таблицы устройств. Номер строки таблицы устройств соответствует значению параметра <логический номер>; по умолчанию рассматривается нулевая строка. При реализации этой формы оператора LIMITS обращения к диску нет.

Примеры:

1) :IO LIMITS F "ДАННЫЕ", A,B,C
2) :IO LIMITS T#I, "ЗАДАЧА",X,Y,Z(I),M
3) :20 LIMITS T #I, A,B,C

2.10.6. Операторы работы с каталогизированными файлами

2.10.6.I. DATA LOAD DC

DATA LOAD DC [<логический номер> ,] <список переменных>

Оператор DATA LOAD DC предназначен для считывания логических записей из файла каталога на диске с последующим присвоением считанных значений переменным из списка переменных.

Перед считыванием данных файл, занесенный ранее в каталог, должен быть открыт путем выполнения оператора DATA LOAD DCOPEN. В противном случае, если нет открытых файлов, издается сообщение об ошибке (ERR 60). Каждый оператор DATA LOAD DC обеспечивает считывание следующей логической записи из файла.

Если в одной логической записи данных недостаточно для заполнения списка переменных, то считывается следующая логическая запись.

Если в логической записи данных больше, чем требуется для заполнения списка переменных, то логическая запись считывается до конца, но лишние последние данные игнорируются.

При каждом выполнении оператора происходит изменение текущего адреса сектора, причем он становится равным адресу первого сектора очередной логической записи. При считывании записи окончания файла адрес сектора устанавливается равным адресу сектора, содержащего эту запись, а считывание прекращается. Условие окончания файла может быть проверено оператором IF *END* THEN.

Попытка считать информацию, записанную за пределами последнего сектора
файла, приводит к возникновению ошибочной ситуации (ERR 76).

Параметр <логический номер> задает строку таблицы устройств, в которой содержится информация о файле; по умолчанию номер строки равен нулю. Реализация оператора заканчивается, если исчерпан список переменных.

По оператору DATA LOAD DC может быть считана лишь информация, содержащая данные. Попытка считать по этому оператору программный файл приводит к возникновению ошибочной ситуации (ERR 67).

2.10.6.2 DATA LOAD DC OPEN

DATA LOAD DC OPEN <тип диска> [<устройство>] [<параметр LOAD OPEN>]

<параметр LOAD OPEN>:: = <символьный аргумент> / TEMP , <AB>, <AB>

Оператор DATA LOAD DC OPEN предназначен для открытия (на запись или чтение) файлов данных, записанных ранее в каталог файлов на диске.

При этом в таблицу устройств заносится номер файла, начальный, конечный и текущий адреса секторов этого файла (текущий адрес устанавливается равным начальному). После того как файл открыт, любое обращение по операторам режима дискового каталога (таких как DATA LOAD DC, DSKIP, DATA SAVE DC, DBACKSPACE) производится к этому файлу, если совпадает номер файла.

По умолчанию данные о файле заносятся в строку таблицы устройств с нулевым номером.

Если имя файла не найдено в указателе каталога или файл отмечен как ликвидируемый (см. оператора SCRATCH), то выдается сообщение об ошибке (ERR 73 или ERR 72 соответственно).

Параметр TEMP используется для открытия временного рабочего файла, после этого доступ к нему возможен с помощью операторов режима дискового каталога, хотя этот файл и находится за областью каталога.

Для повторного открытия файла всегда используется оператор DATA LOAD DC OPEN, даже если в этот файл надо записать данные, так как использование оператора DATA SAVE DC OPEN для повторного открытия файла приводит к ошибке (ERR 71).

Открывать файл необходимо каждый раз при переходе от работы с одним файлом к работе с другим.

Примеры:

1) :10 DATA LOAD DC OPEN F=#5, "ЗНАЧЕНИЯ"
2) :10 DATA LOAD DC OPEN F TEMP 8000,9000

2.10.6.3. DATA SAVE DC

DATA SAVE DC [#] [<логический номер> ,] <параметр SAVE END
<параметр SAVE END> = END / <список аргументов>

Оператор DATA SAVE DC предназначен для записи данных из списка аргументов на диск в виде логической записи.

Запись начинается с текущего адреса сектора файла с заданным номером.

Если номер файла (логический номер) не задан, то запись осуществляется в файл с номером нуль.

Чтобы осуществить запись в файл данных на диске, его необходимо открыть операторами : DATA SAVE DC OPEN или DATA LOAD DC OPEN (т.е., чтобы информация о нем была записана в таблицу устройств на текущий момент времени). При попытке записи в закрытый файл выдается сообщение об ошибке (ERR 60).

Одна логическая запись может занимать одна или несколько секторов. Кроме самих данных на диск записывается и служебная информация в соответствии с форматом представления данных по операторам DATA SAVE, DATA SAVE DC, DATA SAVE DA.

Параметр # задает необходимость считывания данных после записи для проверки правильности записи данных. Такая проверка повышает уверенность в записи, но увеличивает время на реализацию оператора.

Если в операторе задан параметр END, то в файл заносится специальная запись окончания файла о количестве секторов, занятых файлом.

После занесения записи окончания файла в этот файл можно записывать данные со стиранием старой записи окончания файла (для позиционирования начала записи на окончание файла используется оператор DSKIP END). После завершения каждой записи данных необходимо заносить новую запись окончания файла.

Примеры:

1) :100 DATA SAVE DC # # 3, L (), A (5,X), M#

2) :100 DATA SAVE DC #6,END

3) :200 DATA SAVE DC A(),A+B, "ПРИМЕР"

при выводе символьной переменной по этому оператору ее длина должна быть не более 253 байтов.

2.10.6.4. DATA SAVE DC CLOSE

DATA SAVE DC CLOSE [<параметр CLOSE>]

<параметр CLOSE> :: = <логический номер> / ALL

Оператор DATA SAVE DC CLOSE предназначен для закрытия всех открытых файлов, если задан параметр ALL, или одного, номер которого задан в операторе (по умолчанию закрывается файл с номером нуль).

При закрытии файла начальному, конечному и текущему адресам секторов файла в соответствующей строке таблицы устройств присваиваются нули. Последующее выполнение любого оператора со ссылкой на закрытый файл ведет к выдаче сообщения об ошибке (ERR 60).

Оператор DATA SAVE DC CLOSE рекомендуется использовать для закрытия файла после завершения связанных с ним операций, так как при этом исключается возможность ошибочного доступа и разрушения данных в этом файле.

Примеры:

1) :100 DATA SAVE DC CLOSE

2) :200 DATA SAVE DC CLOSE # 4

3) :200 DATA SAVE DC CLOSE ALL

2.10.6.5. DBACKSPACE

DBACKSPACE [<логический номер>][параметр BEG>

<параметр BEG> :: = BEG/ <AB>[S]

Оператор DBACKSPACE предназначен для возврата назад на определенное количество (заданное параметром <AB>) логических записей файла, если нет параметра S, или секторов, если задан параметр S.

При этом текущий адрес сектора файла в таблице устройств меняется (на соответствующее количество секторов). Параметр <логический номер> задает строку в таблице устройств, по умолчанию он равен нулю. Если в операторе задан параметр BEG, то осуществляется возврат на начало первой логической записи в файле. Текущий адрес сектора устанавливается равным начальному адресу сектора файла.

Если в операторе DBACKSPACE количество секторов возврата (при наличии параметра S) слишком велико, то текущий адрес сектора в таблице устройств устанавливается равным начальному адресу сектора в файле. То есть, при реализации оператора DBACKSPACE выход за пределы файла невозможен.

Использование параметра S целесообразно, когда известна длина логических записей (т.е. количество секторов ими занимаемых), так как в этом случае изменяются лишь параметры в таблице устройств и нет необходимости в доступе к диску. При возврате на логическую запись (параметр S отсутствует) необходим доступ к диску и просмотр записей на нем, но в этом случае пользователю не надо знать, сколько секторов занимают логические записи.

Примеры:

1) :I00 DBACKSPACE BEG
2) :I00 DBACKSPACE #2, 5S
3) :I00 DBACKSPACE I0

2.I0.6.6. DSKIP

DSKIP [<логический номер> ,] <параметр END>
<параметр END>::= END / <AB>[S]

Оператор DSKIP предназначен для пропуска определенного количества логических записей (заданного параметром <AB>), если нет параметра S , или секторов, если задан параметр S .

При этом в таблице устройств меняется (на соответствующее количество секторов) текущий адрес сектора. Параметр <логический номер> задает строку в таблице устройств, по умолчанию он равен нулю.

Если задан параметр END, то осуществляется пропуск всех логических записей файла до записи окончания файла.

Если задан параметр S , то пропускаются сектора. Если количество пропускаемых секторов велико (т.е. требует выхода за пределы файла), то текущий адрес сектора в таблице устройств устанавливается равным адресу конечного сектора файла.

Таким образом, при реализации оператора DSKIP выход за пределы файла невозможен.

Использование параметра S целесообразно, когда известно, сколько секторов занимают логические записи, так как в этом случае значительно сокращается время на реализацию оператора за счет того, что не нужен доступ к диску, а происходит лишь коррекция адреса в таблице устройств.

При пропуске же логических записей необходим доступ к диску и просмотр записей на нем, но в этом случае пользователю не надо знать, сколько секторов занимает логическая запись.

Примеры:

1) :IO DSKIP #5, END
2) :20 DSKIP I5
3) :IO DSKIP XNYS

2.IO.6.7. LOAD DC

LOAD DC <тип диска>[<устройство> ,] <символьный аргумент>[<диапазон LOAD>]

<диапазон LOAD> :: =[<номер строки>] , <номер строки>[, <номер строки>] / <номер строки>[, <номер строки>][, <номер строки>] / [<номер строки>],[номер строки>], <номер строки>.

Оператор LOAD DC предназначен для загрузки программы (или ее сегмента) с заданным именем, записанной на диске, в память системы.

Если оператор LOAD DC используется при счете по программе, то при его выполнении происходит следующее:

1) из памяти системы стираются все строки текущей программы или ее сегмента, ограниченного значениями параметров <номер строки> ;

2) стираются значения всех переменных, кроме общих (объявленных в операторе COM) и магазин возвратов;

3) программа с данным именем загружается в память системы с выбранного диска. Если программа с данным именем отсутствует, выдается сообщение об ошибке (ERR 73);

4) программа запускается на счет с номера строки, указанного третьим параметром <номер строки> в диапазоне LOAD.

В режиме непосредственного счета запуск на счет осуществляется только при наличии третьего параметра <номер строки>.

Если номера строки, указанного в качестве третьего параметра, не существует в загруженной программе, то счет начинается с ближайшего большего номера строки программы.

Если в операторе LOAD DC третий параметр <номер строки> отсутствует, то запуск на счет осуществляется с первого параметра <номер строки> или ближайшего занятого.

Если отсутствуют первый и третий параметр, то счет начинается с первой введенной строки.

В зависимости от наличия первых двух параметров <номер строки> стирание строк программы, находящейся в памяти, осуществляется

по следующим правилам:

1) если заданы оба параметра <номер строки> , то стирается часть программы, ограниченная этими номерами строк;

2) если задан только первый параметр <номер строка> , то стирается программа, начиная с заданного номера строки и до конца;

3) если задан второй параметр <номер строки>, то стирается часть программы от ее начала до заданного номера строки;

4) если параметры <номер строки> не заданы, то стирается программа целиком.

В режиме непосредственного счета при выполнении оператора LOAD память не очищается автоматически, что дает возможность соединять несколько программных сегментов для изготовления новых программ. При вводе текста программы, строки с совпадающими номерами заместятся на строки вводимой программы, а оставшиеся строки остаются без изменения. Поэтому при вводе новой программы необходимо предварительно очистить память машины, используя оператор CLEAR.

Оператор LOAD DC при использовании его при счете по программе позволяет организовать пользователю автоматическое прохождение сегментированных программ. При этом, для передачи значений переменных из одного сегмента программы в другой, эти переменные должны быть объявлены как общие.

Адрес диска, с которого загружается программа, задается параметрами <тип диска>[<устройство>,].

Примеры:

1) :IO LOAD DC "ПРОГР1"

2) :20 LOAD DC "ФАЙЛ" 250,300,2

3) :IO LOAD DC PAM(),20,IO

2.IO.6.8. MOVE TO

Оператор MOVE TO имеет две формы.

Первая форма оператора MOVE TO:

MOVE <тип диска>[<устройство>,] TO <тип диска>[,устройство>]

Оператор предназначен для копирования каталога с диска на диск с удалением всех ликвидируемых файлов из зоны каталога и их имен из зоны указателя каталога, при этом все оставшиеся файлы сдвигаются и информация о них в зоне указателя каталога корректируется. Временные файлы не копируются.

Если первый параметр <тип диска> - P, а второй R, то каталог с диска типа P переписывается на диск типа R. Если же первый параметр <тип диска> - R, а второй - P, то процесс копирования идет от диска R к диску P.

Для проверки правильности копирования пользователь вслед за оператором MOVE TO может применить оператор VERIFY.

Перед выполнением оператора MOVE TO необходимо разместить диск, на который осуществляется перепись, по оператору SCRATCH DISK с любыми возможными параметрами (т.е. значения параметров LS и END могут отличаться от тех же параметров для диска, с которого осуществляется перепись).

Если при разметке нового диска (перед копированием) по оператору SCRATCH DISK зона указателя каталога увеличивается, то при переписи может возникнуть ситуация, при которой адрес сектора, следующего за последним использованным сектором в зоне каталога, окажется больше значения параметра END.

В этом можно убедиться выполнив оператор LIST DC. Параметр END в этом случае необходимо увеличить оператором MOVE END.

Если в операторе не заданы параметры <устройство> , то ОАУ дисков задается оператором SELECT DISK.

Примеры:

1) :10 MOVE P #2 TO R #1

2) :20 MOVE R TO P

вторая форма оператора MOVE TO:

MOVE <тип диска>[(<устройство> ,] <символьный аргумент> TO <тип диска>[<направление MOVE>]

<направление MOVE> :: = <устройство> / <устройство>, (<резерв>)

<резерв> :: = <АВ> / <символьный аргумент>

Оператор MOVE TO предназначен для переписи файлов с одного диска на другой.

Возможность переписывания заданного файла на другой диск и получения каталогизированного файла на "ПРИЕМНОМ" диске реализуется путем задания имени файла в операторе MOVE . Параметр <АВ> указывает количество дополнительных секторов, резервируемых для записи нового файла. Если в операторе указано имя файла, на "ПРИЕМНОМ" диске, то оно относится к ликвидированному файлу, место которого занимается переписываемым файлом.

2.10.6.9. MOVE END

MOVE END <тип диска> [<устройство> ,] =<АВ>

Оператор MOVE END используют для изменения размера зоны каталога на диске. Адрес последнего сектора зоны каталога определяется первоначально параметром END в операторе SCRATCH DISK и в дальнейшем может быть изменен (без разрушения зоны каталога) лишь оператором MOVE END.

Значение параметра <АВ> определяет адрес нового конечного сектора зоны каталога. Если по данному адресу размещается какой-либо ранее записанный каталогизированный файл или значение данного адреса превышает максимально допустимое для диска значение, то выдается сообщение об ошибке. (ERR 75 или ERR 80 ОБ) соответственно.

Параметры <тип диска> и <устройство> определяют адрес диска, размеры каталога на котором должны быть изменены оператором MOVE END.

Примеры:

1) : 10 MOVE END# = 2100
2) :20 MOVE END #5,=I+Y
3) :20 MOVE END #1, =4000

2.10.6.10. SAVE DC

SAVE DC <тип диска>[/V] [<устройство> ,][<признак>][(<резерв>)] <символьный аргумент>[(<диапазон строк>)]

<признак> :: = P/T/G

Оператор SAVE DC предназначен для записи программ (сегментов программ) на заданный диск. В указателе каталога записываются имя файла, тип файла (программа), начальный адрес и конечный адрес секторов файла.

Программа автоматически записывается в секторах, отведенных системой под файл. Их адреса хранятся в зоне указателя каталога.

Параметр /V задает считывание после записи для контроля правильности записи программы на диск. Эта проверка увеличивает время на реализацию оператора SAVE DC.

Параметр <символьный аргумент> в параметре <резерв> задает имя старого (ликвидируемого) файла.

Параметр <АВ> указывает системе на необходимость резервирования определенного количества секторов (равного значению арифметического выражения) сверх необходимого количества секторов, занимаемых программой. Дополнительные сектора можно использовать

для последующего расширения программы. Для них отводится место на диске после секторов, занятых программой в свободной области зоны каталога.

Второй параметр <символьный аргумент> определяет имя записываемой программы.

Новая программа может быть записана на место ликвидируемого файла программы или данных, отмеченного оператором SCRATCH, если задан параметр <символьный аргумент> в параметре <резерв>. Если количество секторов, занятых ликвидируемым файлом, не достаточно для записи нового программного файла, то выдается сообщение об ошибке (ERR 26).

Параметры <символьный аргумент>, задающие имя ликвидируемого файла и имя записываемой программы, могут совпадать, но имя записываемой программы не должно совпадать с именами других файлов, занесенных в каталог.

Если в операторе не заданы ни имя ликвидируемого файла, ни параметр <AB>, то новая программа записывается в свободной области каталога и ей отводится столько секторов, сколько для нее требуется. Параметры P,T,G обозначают признак защиты программы при записи на диск. Параметры P и G указывают на то, то программа защищена от распечатки и повторной записи, т.е. она может быть лишь загружена и выполнена. Перед записью или распечаткой любой программы после загрузки защищенной программы необходимо полностью очистить память по оператору CLEAR.

Параметр P означает запись нетранслированной программы с защитой, параметр T означает запись транслированной программы без защиты, параметр G означает запись транслированной программы с защитой.

При записи транслированной программы на диск, ее текст записывается в преобразованном оттранслированном виде, поэтому на диске транслированная программа занимает меньший объем по сравнению с неоттранслированной программой. Кроме этого загрузка программы (или программного сегмента) в транслированном виде по оператору LOAD происходит быстрее, так как исключается процесс трансляции во время загрузки. Поэтому отлаженные программы рекомендуется хранить в транслированном виде, сохраняя для надежности копию программы в исходном тексте, так как восстановить нетранслированную программу значительно проще при потере какой-либо части программы, например, вследствие сбоя сектора с записью программы на диске. Так как при записи на диск программы в транслированном

виде записывается таблица переменных, то перед записью программы следует выполнить оператор CLEAR,V, стирающий из таблицы переменные, отсутствующие в тексте записываемой программы.

Параметр <диапазон строк> определяет первую и последнюю строки сегмента записываемой программы:

1) если задан только первый параметр <номер строки> в параметре <диапазон строк>, то записывается сегмент программы, начиная со строки с данным номером и до конца программы;

2) если задан только второй параметр <номер строки> в параметре <диапазон строк>, то записывается сегмент программы, начиная с первой строки программы и до строки программы с заданным номером;

3) если параметр <диапазон строк> отсутствует, то записывается вся программа.

В случае необходимости корректировки файла программы следует выполнить следующие действия:

1) загрузить программу с диска в память и откорректировать ее;

2) отметить этот программный файл на диске как ликвидируемый (оператор SCRATCH);

3) записать откорректированную программу на старое место на диске (оператор SAVE DC).

Примеры:

1) :IO SAVE DC F T (50) "КООРД"
2) :IO SAVE DC R "ЗАДАЧА"
3) :IO SAVE DC P ("ЗАДАЧА1") "ЗАДАЧА2" 1000
4) : SAVE DC R# (10) "ПРОГР1" 380,1000
5) :IO SAVE DC F,K#2,P"ПРОГ1"

2.10.6.11. SCRATCH

SCRATCH[<тип диска>[устройство],][<список имен>

<список имен> ::= <символьный аргумент> / <список имен>, <символьный аргумент>

Оператор SCRATCH предназначен для присвоения каталогизированным файлам, перечисленным в списке имен, состояния "ликвидируемый". При распечатке указателя каталога эти файлы помечаются символом "S". К ликвидируемым файлам нет доступа по операторам дискового каталога. Секторы, занятые ликвидируемыми файлами, могут быть снова использованы для записи новых файлов программ и данных

по операторам DATA SAVE DC OPEN (файла данных) и SAVE DC (программного файла). В этом случае возможны два варианта:

1) имена нового файла и ликвидируемого совпадают;

2) имена нового и ликвидируемого файлов различные.

Если имена файлов совпадают, то в указателе каталога признак "ликвидируемый" стирается, а сформированная там информация о старом файле будет относиться к новому файлу.

Если имена различны, то в указателе каталога формируется информация о файле с новым именем, а файл со старым именем становится "заменяемым".

Параметры <тип диска> и <устройство> задают адрес диска.

Оператор SCRATCH обычно используют до выполнения копирования с диска на диск по оператору MOVE с целью удаления файлов, ненужных в копии.

Если файл был отмечен как ликвидируемый, а затем на его место записан новый файл с другим именем, то при просмотре указателя каталога (по оператору LIST DC) будет получена информация о новом файле, хотя в зоне указателя каталога сохранилась и информация о старом, теперь уже замененном файле. Так как имена ликвидируемых и замененных файлов исчезают из указателя каталога только при перепаковке диска по оператору MOVE , то задавая новые имена для файлов, следует учитывать размер указателя каталога. При переполнении указателя каталога выдается сообщение об ошибке (ERR 78).

Примеры:

```
1) :10 SCRATCH F "ПРОГ1"
   :20 SAVE  F ("ПРОГ1)" "ПРОГ2"
2) :10 SCRATCH F"ПРОГ"
   :20  DATA SAVE  DC  OPEN F ("ПРОГ") "ДАННЫЕ"
   :30  DATA SAVE  DC   M (),  L ()
   :40 DATA  SAVE  DC AM ()
   :50 DATA  SAVE  DC  END
3) :10 SCRATCH  F "ЗАДАЧА": DATA SAVE DC  F    OPEN "ДАННЫЕ"
   :20 SKIP  10000
   :30 DATA  SAVE DC  END
```

В примере 1) на место "ПРОГ1" (данные или программа) записывается программный файл "ПРОГ2". В примере 2) на место файла "ПРОГ" записываются файл данных с именем "ДАННЫЕ", состоящий из двух логических записей. В примере 3) имя файла данных "ЗАДАЧА" заменяется на имя "ДАННЫЕ".

2.10.6.12. SCRATCH DISK

SCRATCH DISK <тип диска> [<устройство> ,] [LS=<AB>,]
END = <AB>

Оператор SCRATCH DISK предназначен для резервирования секторов диска для зоны указателя каталога и зоны каталога перед записью программ и файлов данных на диск. Выполнение этого оператора должно предшествовать выполнению операторов режима дискового каталога.

При выполнении оператора SCRATCH DISK на диске отводится место под зону указателя каталога, начиная с нулевого сектора. Параметр LS задает размер указателя каталога (это значение, задаваемое арифметическим выражением, должно быть от 1 до 255, в противном случае выдается сообщение об ошибке ERR 18). По умолчанию под зону указателя каталога отводится 24 сектора (с 0 по 23).

Запись каждого каталогизированного файла в указателе каталога состоит из имени файла и связанных с ним параметров. В каждом секторе указателя каталога можно разместить до 16 записей о файлах, исключение составляет нулевой сектор, в котором размещается 15 записей о файлах, поскольку начальная часть сектора используется для записи и хранения информации обслуживания каталога. Структура байтема зоны указателя каталога (УК) имеет следующий вид:

16 байтов

Тип файла	Начальный адрес сектора области каталога (ОК)	Конечный адрес сектора (ОК)	Коды HEX(00)	Имя файла
2 байта	2 байта	2 байта	2 байта	8 байтов

При подготовке каталога по оператору SCRATCH DISK зоны указателя каталога заполняется кодами HEX(00).

Параметр END задает границу каталога на диске (его значение, задаваемое арифметическим выражением, не должно превышать адрес последнего сектора на диске). Целесообразно оставлять часть секторов за зоной каталога для организации временных рабочих файлов.

Зону каталога на диске можно расширить и увеличивать, смещая конец зоны каталога путем повторения оператора MOVE END. Но размер указателя каталога нельзя изменить без полной реорганизации каталога.

Примеры:

1) :IO SCRATCH DISK R END =7920

Пользователь отводит на диске консольного дискового устройства 24 сектора (с 0 по 23) под зону указателя каталога и сектора с 24 по 7920 под зону каталога.

2) :IOO SCRATCH DISK F#5, LS=N, END=S

Пользователь на диске дискового устройства с логическим номером 5 отводит N секторов под зону указателя каталога (с 0 по $N-1$) и сектора с N по S под зону каталога, при этом $N<S$ (в противном случае выдается сообщение об ошибке ERR 55).

2.IO.6.I3. VERIFY

VERIFY <тип диска> [<устройство>,][(<AB>, <AB>)]

Оператор VERIFY предназначен для считывания содержимого всех секторов, расположенных в зоне заданной параметрами <AB>, указанного диска с целью контроля правильности записи информации на диске.

Значение первого параметра <AB> задает адрес первого сектора считываемой зоны, а второго - адрес последнего сектора считываемой зоны.

Значение первого параметра <AB> должно быть больше значения второго параметра <AB>. В противном случае выдается сообщение об ошибке (ERR 55, то же и для оператора COPY).

Если в операторе VERIFY параметры <AB> не заданы, то осуществляется проверка всех секторов в диапазоне 0 - I000.

При обнаружении ошибок на консольное устройство выводится информация о номере ошибочного сектора, например:

ERROR IN SEKTOR 960

Примеры:

1) :IO VERIFY F#2 (I0,2000)

проверяются сектора с I0 по 2000 на диске устройства с логическим номером 2.

2) :IO VERIFY R

проверяются сектора с 0 по I000 на диске консольного дискового устройства.

2.IO.7. Прямая адресация секторов

2.IO.7.I. Возможности режима прямой адресации секторов

В режиме прямой адресации секторов используются семь операторов, которые дают возможность считывать информацию, записывать

копировать ее на другой диск, проверять качество записи.

В этом режиме система не обеспечивает автоматического создания и ведения указателя каталога, нельзя также присваивать имена файлам программ и данных. Идентификация файлов осуществляется только путем ссылки на их начальные адреса секторов. Вся адресная информация должна быть подготовлена пользователем, поскольку система не готовит и не сохраняет такую информацию автоматически.

2.10.7.2. Типы операторов

В режиме прямой адресации секторов используются два типа операторов:

операторы DA;

операторы BA.

Операторы обоих типов обеспечивают прямой доступ к заданным секторам диска.

Операторы типа DA используют для записи и считывания программ и данных, начиная с сектора, номер которого задают в операторе. Если программа или логическая запись данных занимает один или более секторов, то информация записывается в необходимое количество секторов автоматически точно так же, как и при выполнении аналогичного оператора режима дискового каталога. Форматы предоставления данных для этих режимов также совпадают – содержат одинаковую управляющую информацию. Поэтому информацию, записанную в режиме дискового каталога, можно считать и в режиме прямой адресации операторами типа DA.

Операторы типа BA специально предназначены для записи и считывания информации, содержащейся в одном секторе – 256 байтов. Считать с диска по оператору BA можно информацию, записанную любым дисковым оператором.

2.10.7.3. Использование таблицы устройств

В режиме прямой адресации секторов нет каталога и указателя каталога, но таблица устройств существует и используется для хранения адресных параметров.

Таблица устройств в этом режиме хранит следующую информацию:

1) начальный номер сектора, заданный в операторе параметром <адрес сектора>;

2) максимально возможный номер сектора (определяемый типом диска);

3) номер сектора, следующего за последним использованным сектором по данному оператору.

После реализации оператора это значение присваивается параметру <переменная>, заданному в операторе.

Выбор строки таблицы устройств аналогичен ее выбору в случае режима дискового каталога; если в оператор режима адресации секторов включен логический номер, то используется строка таблицы устройств с этим номером, если же устройство задано через свой физический адрес или не задано вообще, то используется нулевая строка таблицы устройств.

При чередующемся использовании операторов обоих режимов необходимо быть особенно внимательным. Например, параметры открытого файла будут уничтожены, если в операторе режима адресации будет указан номер этого файла.

Прямая адресация дает также возможность копирования и проверки хранимых на диске данных операторами COPY и VERIFY.

2.10.8. Операторы прямой адресации секторов

2.10.8.1. COPY TO

COPY <тип диска> [<устройство>,][<AB> , <AB>]TO <тип диска> [<направление>]

<направление> :: = <устройство> / <устройство>, (<AB>)

Оператор COPY предназначен для физического копирования информации с диска на диск.

Значения параметров <AB> задают адреса первого и последнего секторов копируемой области. Если они не заданы, то осуществляется копирование всех секторов в диапазоне 0 - 1000. Параметр <AB> в параметре <направление> определяет номер начального сектора диска, на который производится запись. Если он не задан, информация на приемный диск записывается в тех же секторах, что и в копируемой области.

Первый параметр <устройство> задает диск, с которого осуществляется копирование, второй параметр <устройство> задает диск, на который осуществляется копирование.

В отличие от оператора MOVE оператор COPY при копировании каталога не удаляет файлы, отмеченные как ликвидируемые. Для проверки информации после копирования можно использовать оператор VERIFY.

Примеры:

1) :10 COPY F#2,(10,500) TO F#5,(20)

2) :20 COPY R TO F

2.10.8.2. DATA LOAD BA

DATA LOAD BA <тип диска> [<устройство>,](<адрес сектора>) <символьная переменная>

<адрес сектора> : = <AB> [,<переменная>] /<символьный аргумент> [,<переменная>]

Оператор DATA LOAD BA предназначен для считывания информации из одного сектора на диске в символьную переменную.

При выполнении этого оператора считывается все 256 байтов информации, содержащейся в секторе.

Информация считывается из сектора, номер которого задан параметром <адрес сектора> , номер следующего за ним сектора заносится в параметр <переменная> . Если в качестве этого параметра используется числовая переменная, то в нее заносится десятичный номер сектора, если - символьная , то номер заносится в ее первые два байта в виде двоичного числа.

Если длина символьной переменной больше 256 байтов, то его остальные байты сохраняют старые значения.

Если длина символьной переменной не достаточна для размещения 256 байтов, то последние считанные байты игнорируются.

Примеры:

1) :10 DATA LOAD BA P (50) AX
2) :10 DATA LOAD BA R #2,(L1,LX) AR

2.10.8.3. DATA LOAD DA

DATA LOAD DA <тип диска> [<устройство> ,](<адрес сектора>) <список переменных>

оператор DATA LOAD DA предназначен для считывания логических записей с диска, начиная с сектора, номер которого задан параметром <адрес сектора> , с последующим присвоением значений переменным из списка переменных.

Для того, чтобы считывание информации было возможно, она должна быть представлена в соответствующем формате - записана по операторам DATA SAVE DA и DATA SAVE DC.

Если в одной логической записи недостаточно данных для заполнения списка переменных, то считывается следующая логическая запись.

Если в логической записи данных больше, чем требуется для заполнения списка переменных, то логическая запись считывается до конца, но лишние, последние данные игнорируются.

После выполнения оператора DATA LOAD DA в параметр <переменная>, если он задан в параметре <адрес сектора>, заносится номер сектора, с которого начинается следующая логическая запись. Если в качестве этого параметра использована числовая переменная, то заносится десятичный номер, если символьная – номер заносится в ее первые два байта в виде двоичного числа.

Если при считывании логической записи встретилась запись окончания файла, то считывание данных прекращается, а значение <переменная> устанавливается равным номеру сектора, в котором содержится эта запись, а не номеру сектора следующего по порядку.

Примеры:

1) :10 DATA LOAD DA R (A%,L%)X, Y(),Z%
2) :30 DATA LOAD DA F #2, (B+5, L%)Z%(), Y(5,I)

2.10.8.4. DATA SAVE BA

DATA SAVE BA <тип диска>[#][<устройство>,] (<адрес сектора>) <символьная переменная>

Оператор DATA SAVE BA предназначен для записи символьной информации на диск, помещающейся в один сектор (256 байтов).

Если символьная переменная содержит более 256 байтов, то на диск выводятся лишь первые 256 байтов.

Если символьная переменная содержит менее 256 байтов, то остаток сектора заполняется кодами HEX(00).

Данные записываются в сектор, номер которого задан параметром <адрес сектора>, а номер следующего за ним сектора заносится в параметр <переменная>, если он задан в параметре <адрес сектора>. Если в качестве этого параметра задана числовая переменная, то в нее заносится десятичный номер, если символьная – то в ее первые два байта заносится номер в виде двоичного числа.

Наличие в операторе параметра % задает контрольное чтение данных после записи, тем самым увеличивая вдвое время выполнения оператора.

Примеры:

1) :10 DATA SAVE BA R # 5,(8+5,M%)X%
2) :20 DATA SAVE BA F/1C, (A%,B%)L%()
3) :20 DATA SAVE BA R%#5,(L,L)A% ()

2.10.8.5. DATA SAVE DA

DATA SAVE DA <тип диска>[#][<устройство>,] (<адрес сектора>) <параметр SAVE END>.

Оператор DATA SAVE DA предназначен для записи данных из списка аргументов, начиная с сектора, номер которого задан значением параметра <адрес сектора> . Данные записываются в том же формате, что и в режиме дискового каталога по операторам DATA SAVE DC. Каждый оператор DATA SAVE DC записывает на диск одну логическую запись, занимающую один или несколько секторов диска.

После выполнения оператора DATA SAVE DA в параметр <переменная>, если он задан, заносится номер сектора, следующего за последним сектором, занятым этой логической записью. Если в качестве этого параметра задана числовая переменная, то заносится десятичный номер, если символьная - в ее первые два байта заносится номер в виде двоичного кода.

Наличие в операторе параметра # задает контрольное чтение данных после записи.

Если в операторе задан параметр END, то на диск записывается логическая запись конца файла, которую можно использовать для поиска конца файла оператором IF END THEN .

Оператор DATA SAVE DA, как правило, не используется при работе с каталогизированными файлами, поскольку существует риск разрушения информации каталога при выполнении оператора режима прямой адресации секторов. Основные причины этого в том, что:

1) не обеспечена проверка границ файла;

2) в указателе каталога не обновляется информация о количестве использованных секторов по оператору DATA SAVE DA с параметром END .

Примеры:

```
1) :10 DATA SAVE DA F # 2, (C#,C# ) X,Y#
2) :10 DATA SAVE DA   R (20,B)X,A
3) :10 DATA SAVE DA   R/10,(A,B) END
```

2.10.8.6. LOAD DA

LOAD DA (тип диска)[<устройство> ,] (<адрес сектора>)
[<диапазон LOAD>]

Оператор LOADDA предназначен для загрузки с диска всей или части программы в память системы аналогично оператору LOAD DC.

Параметр <адрес сектора> задает номер первого сектора на диске, который содержит имя программы.

Если оператор LOAD DA используется при счете по программе, то выполняется следующее:

1) из памяти системы стираются все строки текущей программы или ее часть, ограниченная значениями первых двух параметров <номер строки> в параметре <диапазон LOAD>;

2) стираются значения всех необщих переменных;

3) программа с диска загружается в память системы;

4) программа запускается на счет с номера строки, указанного третьим параметром <номер строки> в параметре <диапазон LOAD>.

Если номера строки, указанного в качестве третьего параметра, не существует в загруженной программе, то счет начинается с ближайшего большего номера строки программы.

Если в операторе LOAD DA третий параметр <номер строки> отсутствует, то запуск на счет осуществляется с первого параметра <номер строки> или ближайшего большего.

Если отсутствуют первый и третий параметры, то счет начинается с первой введенной строки.

В режиме непосредственного счета запуск на счет осуществляется только при наличии третьего параметра <номер строки> в параметре <диапазон LOAD>.

В зависимости от наличия первых двух параметров <номер строки>, стирание строк программы, находящейся в памяти, осуществляется по следующим правилам:

1) если заданы оба параметра <номер строки>, то стирается часть программы, ограниченная этими номерами строк;

2) если задан только первый параметр <номер строки>, то стирается программа, начиная с заданного номера строки и до конца;

3) если задан только второй параметр <номер строки>, то стирается часть программы от ее начала до заданного номера строки,

4) если параметры <номер строки> не заданы, то стирается программа целиком.

В переменную, задаваемую в параметре <адрес сектора>, если она задана, после выполнения оператора LOAD DA заносится номер следующего за последним занятым текстом программы сектора. Если в качестве этого параметра используется числовая переменная, то заносится десятичный номер, если - символьная, то номер заносится в ее первые два байта в виде двоичного числа.

Оператор LOAD DA в счете по программе позволяет пользователю организовать автоматическое прохождение сегментированных программ. Для передачи значений переменных из одного сегмента программы в другой, переменные должны быть объявлены как общие.

Примеры:

1) :IO LOAD DA F(40,L)
2) :IO LOAD DA R#2, (N%,B) 3I0,450
3) :IO LOAD DA F(A,A),400
4) :IO LOAD DA R(2*,X,A)120,200,I0
5) :IO LOAD DA R(R%,D),,400

2.10.8.7. SAVE DA.

SAVE DA <тип диска>D<I>[<устройство>][<признак>] (<адрес сектора>) [<диапазон строк>].

Оператор SAVE DA предназначен для записи программы (сегмента программы) в заданном месте диска.

Программа записывается на диске, начиная с сектора, номер которого задан параметром <адрес сектора>. Так как оператор SAVE DA не позволяет присваивать программе имя, то записанная с его помощью программа, может быть считана с диска лишь по оператору LOAD DA.

После выполнения оператора SAVE DA в переменную, если она задана, в параметр <адрес сектора> заносится первый свободный номер сектора. Если этот параметр задан числовой переменной, то заносится десятичный номер, если - символьной, то номер заносится в ее первые два байта в виде двоичного кода.

Параметр K задает считывание после записи для контроля записи. Это увеличивает время выполнения оператора SAVE DA.

Параметры P, T и C обозначают признак защиты программы при записи на диск. Если защищенная программа была загружена в память системы, то любая другая программа может быть записана или распечатана, только, если перед ее загрузкой память была очищена по оператору CLEAR или по команде ReSET.

Параметры <номер строки> определяют первую и последнюю строки сегмента записываемой программы:

1) если задан только первый параметр <номер строки>, то записывается сегмент программы, начиная со строки с данным номером и до конца;

2) если задан только второй параметр <номер строки>, то записывается сегмент программы, начиная с ее первой строки и до строки програм-

мы с данным номером;

3) если параметры <номер строки> не заданы, то записывается вся программа целиком.

Использование оператора SAVE DA для записи программы в рамках дискового каталога может привести к разрушению каталога.

Примеры:

1) :IO SAVE DA F#2, P(A,A2)
2) :IO SAVE DA R # (A#,L)300
3) :IO SAVE DA F (M+P,L)10,30
4) :IO SAVE DA R/IC,(80,L)

2.II. Операторы специального назначения.

2.II.I. ASMB

ASMB <параметр ASMB>

<параметр ASMB> ::= <символьная переменная>,<символьная переменная>,<символьная переменная>[<режим>] / <символьная переменная> * / , <адрес запуска> [, <список аргументов>]

<режим> ::= * /#[<диапазон строк>]

<адрес запуска> ::= <цифра> / <цифра><цифра> / <цифра><цифра><цифра> / <цифра><цифра><цифра><цифра> / <цифра><цифра><цифра><цифра><цифра>

Оператор ASMB предназначен для создания программ пользователя на уровне инструкций ПЭВМ "Искра 226" с помощью ассемблера и организации счета по созданной программе.

Оператор ASMB может выполнять следующие функции:

1) трансляцию ассемблерной программы в последовательность кодов инструкций;

2) трансляцию ассемблерной программы в последовательность кодов инструкций с последующей выдачей листинга;

3) трансляцию ассемблерной программы в последовательность кодов инструкций с последующей записью объектных кодов программы в управляющую память;

4) запись объектных кодов программы в управляющую память;

5) запуск на счет программы на уровне инструкций по заданному адресу управляющей памяти.

Ассемблерная программа в рамках БейСИК-системы представляет собой последовательность идущих подряд строк БЕЙСИК-программы, содержащих оператор % (не допускается несколько операторов % в одной строке).

В теле оператора % располагаются конструкции ассемблера.

Текст ассемблерной программы, которую необходимо оттранслировать и выполнить, должен быть размещен в символьной переменной. Формирование символьной переменной, содержащей текст ассемблерной программы, должно производиться средствами БЭЙСИК-системы.

При формировании символьной переменной следует учитывать объем, необходимый для записи ассемблерной программы. Допускается использование символьной переменной, превышающей длину, необходимую для записи ассемблерной программы.

Трансляция ассемблерной программы в последовательность кодов инструкций обеспечивается реализацией следующей конструкции оператора ASMB:

ASMB <символьная переменная> <символьная переменная> <символьная переменная>

Первая символьная переменная предназначена для хранения ассемблерной программы пользователя.

Во второй символьной переменной формируется объектная программа, оттранслированная из текста в первой символьной переменной.

Третья символьная переменная предназначена для использования в качестве рабочего массива (объем массива не менее 556 байтов, в противном случае выдается сообщение об ошибке - ERR 59).

Перед трансляцией ассемблерной программы осуществляется проверка на наличие ассемблера в постоянной памяти загрузчика системы. Если ассемблера нет, выдается сообщение об ошибке - ERR 16.

Если в процессе трансляции ассемблерной программы выявляется ошибка, выполнение оператора ASMB прекращается и выдается сообщение об ошибке. Трансляция ассемблерной программы учитывает адрес расположения объектной программы в управляющей памяти. Начальный адрес расположения объектной программы в управляющей памяти определяется следующим образом:

1) формируется с помощью конструкции ассемблера в самом начале ассемблерной программы;

2) устанавливается с учетом фиксированного адреса управляющей памяти, равного 7e000 (восьмер.).

Трансляция ассемблерной программы в последовательность кодов инструкций с последующей выдачей листинга обеспечивается с помощью следующей конструкции оператора ASMB:

ASMB <символьная переменная><символьная переменная> <символьная переменная>,1[<диапазон строк>] .

Назначение символьных переменных аналогично описанному выше.

Символ "%" - признак выдачи листинга ассемблерной программы. Выдача листинга производится на устройство, заданное оператором SELECT LIST.

Конструкция <диапазон строк> обеспечивает выдачу листинга всей ассемблерной программы или ее части.

Трансляция ассемблерной программы в последовательность кодов инструкций с последующей записью объектных кодов программы в управляющую память обеспечивается реализацией следующей конструкции оператора ASMB:

ASMB <символьная переменная>, <символьная переменная>, <символьная переменная>*

Назначение символьных переменных аналогично описанному выше. Символ "*" - признак записи в управляющую память по адресу, указанному в ассемблерной программе или по адресу 76000 (восьмер.). При отсутствии ассемблерной конструкции "адрес".

Запись объектных кодов программы в управляющую память обеспечивается реализацией следующей конструкции оператора ASMB:

ASMB <символьная переменная>*

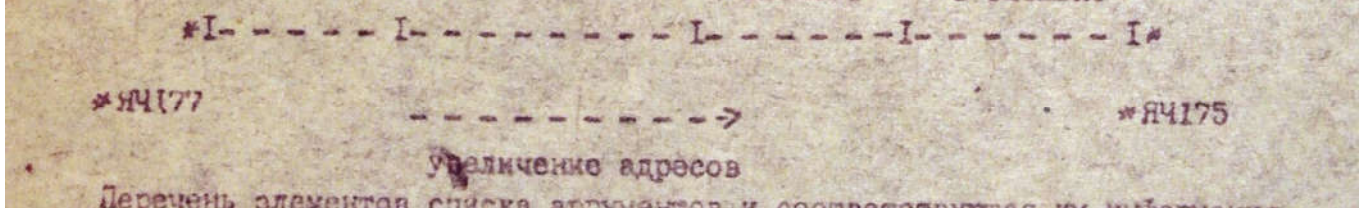
В символьной переменной располагается объектная программа.

Запуск программы на уровне инструкций по заданному адресу управляющей памяти обеспечивается реализацией следующей конструкции оператора ASMB:

ASMB, <адрес запуска>[, <список аргументов>]

<адрес запуска> - представляет собой восьмеричное число в диапазоне 0-76777.

Список аргументов, задаваемый в данной конструкции оператора ASMB, формируется в области ОЗУ, границы которого сформированы в ячейках 177 и 175.



Перечень элементов списка аргументов и соответствующая им информация записанная в ОЗУ, представлены в таблице 6.

Возврат в БЕЙСИК-программу из ассемблерной программы осуществляется при условии, что она оформлена как подпрограмма.

Таблица 6

Элемент списка	5 слово	4 слово	3 слово	2 слово	1 слово
Целая переменная (простая, элемент целого массива)				HEX(00) *	адрес значения
Значение арифметического выражения (целое), целая константа				HEX(01)	значение
Действительная переменная (простая, элемент действительного массива)				HEX(02) *	адрес значения
Значение арифметического выражения действительное, действительная константа	HEX(03)	значение (4 слова)			
Символьная переменная, символьная константа, символьный массив			HEX(04) *	адрес значения	длина
Массив целый			HEX(05) *	адрес значения	длина
Массив действительный			HEX(06) *	адрес значения	длина

* - в старшем байте указывается номер переменной.

2.11.2. ⁄OAD .

⁄OAD <символьная переменная> [<диапазон ⁄OAD>]

<диапазон ⁄OAD>::= [<номер строки>] ,<номер строки> [,<номер строки>] / <номер строки> [,<номер строки>][, <номер строки>] /[<номер строки>] ,[<номер строки>] ,<номер строки>

Оператор ⁄OAD предназначен для загрузки программ из символьной переменной, в память системы.

Если оператор ⁄OAD используется при счете по программе, то производится следующее:

1) сброс строк текущей программы или ее части, в соответствии с указанными номерами строк;

2) программа из символьной переменной загружается в память системы;

3) если при вводе программы возникают противоречия переменных, то производится сброс значений необщих переменных;

4) программа запускается на счет с номера строки, указанного третьим параметром <номер строки> в диапазоне LOAD.

Если номера строки, указанной в качестве третьего параметра, не существует в загруженной программе, то счет начинается с ближайшего большего номера строки программы.

Если в операторе LOAD третий параметр <номер строки> отсутствует, то запуск на счет осуществляется со строки, номер которой соответствует первому параметру или ближайшего большего.

Если отсутствуют первый и третий параметры, то счет начинается с первой введенной строки.

В режиме непосредственного счета запуск на счет осуществляется только при наличии третьего параметра <номер строки>.

В зависимости от наличия первых двух параметров <номер строки>, стирание строк программы, находящейся в памяти, осуществляется по следующим правилам:

1) если заданы оба параметра <номер строки>, то стирается часть программы, ограниченная этими номерами строк;

2) если задан только первый параметр <номер строки>, то стирается программа, начиная с заданного номера строки и до конца;

3) если задан второй параметр <номер строки>, то стирается часть программы от ее начала до заданного номера строки;

4) если параметры <номер строки> не заданы, то стирается программа целиком.

Примеры:

```
1) :10 LOAD A%()  100,300,2
2) :20 LOAD B%()  10,30
3) : LOAD C%(),20,10
```

2.11.3. SAVE

SAVE <символьная переменная> [<диапазон строк>]

Оператор SAVE предназначен для записи программы из памяти системы в символьную переменную.

Программа, находящаяся в памяти, записывается в символьную переменную полностью или частично в зависимости от значений первых

двух параметров <номер строки>.

Длина программы, которая может быть записана в символьную переменную, ограничивается длиной символьной переменной. Если программа или ее часть не помещается в заданную символьную переменную то выдается сообщение об ошибке (ERR 32).

2.II.4. ¤GIO .

¤GIO [<устройство>,] (<символьный аргумент> , <символьная переменная>) [<символьная переменная >]

Оператор ¤GIO предназначен для программирования обмена с заданным устройством в кодах канального процессора.

Первый символьный параметр <символьный аргумент> должен содержать программу обмена с заданным устройством. Размеры символьного аргумента должны быть достаточны для хранения всех байтов программы обмена. Иначе при переходе за границу этого параметра выдается сообщение об ошибке.

Второй символьный параметр <символьная переменная > определяет место в ОЗУ и представляет собой буфер обмена, необходимый при работе с указанным устройством.

Третий символьный параметр <символьная переменная>используется для хранения данных из сверхоперативной памяти канального процессора после выполнения оператора ¤GIO.

Примеры:

```
1) :10  ¤GIO /05,(M¤ ,L¤ )S¤
2) :20  ¤GIO /05,(A¤ ,B¤)
```

2.II.5. ¤GIO'

¤GIO' <символьный аргумент> [, <переменная >]

Оператор ¤GIO' предназначен для написания фрагментов программ в кодах центрального процессора "Искра 226".

Первый параметр <символьный аргумент> содержит программу пользователя в кодах центрального процессора (автокодную программу).

Второй параметр <переменная>резервирует место в оперативной памяти и представляет собой буфер для хранения информации пользователя, необходимой при работе с автокодной программой.

Начальный адрес буфера подготовлен в РОН БАЗА 7 , длина буфера сформирована в РОН РЕГ9.

Выход за пределы буфера должен контролироваться самим пользователем чтобы не испортить информацию в оперативной памяти.

Автокодная программа размещается в управляющей памяти, начиная с

восьмеричного адреса 76000; длина автокодной программы не должна превышать 512 слов.

После выполнения сформированной последовательности автокодной программы управление передается оператору, следующему за оператором ¤GIO' , если в выполняемой программе нет инструкции передачи управления в другие области управляющей памяти.

Примеры:

1) :IO ¤GIO' A¤

2) ¤GIO' A¤, B¤

2.12. Операторы графического взаимодействия.

2.12.1. Функциональные возможности.

Операторы графического взаимодействия обеспечивают:

1) поэлементное построение и стирание плоских изображений - графических объектов, их вывод на экран дисплея;

2) хранение построенных графических объектов в памяти, их запись и считывание с носителей;

3) линейные преобразования построенных графических объектов;

4) компоновку новых графических объектов из уже построенных графических объектов и (или) их частей;

5) построение и анализ экранного изображения с помощью светового пера;

6) представление визуальных характеристик построенных графических объектов в обычной числовой форме, т.е. преобразование графических данных в числовые.

2.12.2. Основные сведения о графических объектах и операторах.

Для хранения графических объектов в памяти используются одномерные или двумерные символьные массивы. Содержимое (значения) этих массивов являются значениями построенных графических объектов.

Каждый графический объект, созданный из символьного массива, рассматривается как непрерывная последовательность байтов без учета деления массива на строки и столбцы.

Объем ОЗУ, отводимый под графический объект, хранимый в символьном массиве, задается оператором DIM или COM, а по умолчанию под графический объект отводятся 1600 байтов ОЗУ. Минимальный размер ОЗУ, необходимый для графического объекта, должен быть не менее

64 баятов.

Для того, чтобы символьный массив рассматривался операторами в качестве графического объекта, надо использовать оператор XUPaN.

Каждый графический объект состоит из последовательности графических элементов трех типов:

1) точка;
2) вектор;
3) надпись.

Построение всех графических объектов ведется в прямоугольной системе координат, т.е. положение каждого графического элемента на плоскости определяется парой чисел X и У, являющихся горизонтальной и вертикальной координатами элемента.

Значения любых координат, приведенные к экранным дискретам, должны быть в пределах:

0 <= X < 32767 0 <= У < 32767

Таким образом, описание каждого графического элемента должно состоять из типа элемента (точка, вектор или надпись), его координат X и У, занимающих два байта ОЗУ каждая, и кроме того, для надписи- из последовательности символов, входящих в надпись.

Среди всей совокупности графических объектов выделен один специальный графический объект, называемый экранным графическим объектом. Любой графический оператор, относящийся к экранному графическому объекту, помимо изменения описания графического объекта в ОЗУ, обеспечивает отображение этого изменения в описании на экране дисплея.

Например, строится точка в экранном графическом объекте. Ее координаты и тип элемента (точка) заносится в ОЗУ, в дополнение к уже имеющемуся описанию данного графического объекта. Кроме того, так как это экранный графический объект, то построенная в ОЗУ точка высвечивается на экране, если только ее координаты входят в экранную область.

Аналогично происходит стирание элемента изображения - графического элемента - из описания графического объекта в ОЗУ, а также если графический объект - экранный, то и с экрана.

Поскольку размеры области возможных построений графических объектов значительно превышают размеры области экрана дисплея, то используют оператор WINDOW , задающий область значений координат графических объектов, совпадающую с областью экрана дисплея.

Для удобства построений графических объектов предусмотрены операторы, задающие и изменяющие систему единиц измерения координат; эти операторы в дальнейшем будем называть операторами метрики графических объектов.

За исходную единицу измерения координат принята одна экранная дискрета (0.4мм). Если при построении графического объекта операторами метрики графических объектов не вводятся другие единицы измерения, то все координаты исчисляются в экранных дискретах.

Введение новых единиц измерения координат не изменяет значение уже имеющихся координат в построенном графическом объекте, а определяет значения координат графического объекта при последующих построениях.

Для компоновки изображений из уже имеющихся графических объектов и (или) их частей используется оператор объединения графических объектов и (или) их графических элементов - A(LET.

Оператор A(LET создает новый (по значению, а не по наименованию) графический объект, состоящий из значений построенных ранее графических объектов, а также служит для присвоения значений экранному графическому объекту.

Каждый построенный графический объект может подвергаться линейным преобразованиям, включающим в себя:

1) сдвиг значений графических объектов (координат элементов графического объекта) по осям X и Y;

2) растяжение или сжатие графического объекта;

3) поворот графического объекта на заданный угол.

Линейное преобразование графического объекта состоит в последовательном (поэлементном) перестроении значений координат каждого графического элемента, входящего в преобразуемый графический объект, т.е. при линейном преобразовании формируется новый графический объект на том же месте в ОЗУ, под тем же наименованием.

Если преобразуемый графический объект - экранный графический объект, то по окончании преобразования новый графический объект выводится на экран дисплея.

Возможность создания, стирания и распознавания изображения на экране с помощью светового пера, обеспечивает непосредственную работу пользователя с экраном дисплея.

Оператор работы со световым пером предусматривает создание элементов изображения (точек и векторов) на экране путем ведения специального графического курсора, их стирание, а также определение характеристик уже построенных на экране графических элементов, их типов и координат.

Световое перо обеспечивает синтез и анализ экранного изображения наиболее простым и доступным способом.

Для управления работой светового пера выделен ряд функциональных клавиш специальных функций на клавиатуре машины "Искра 226", а также кнопка на самом световом пере.

Построенный графический объект может быть записан на диск (и в дальнейшем введен с диска) средствами записи (и считывания) символьных массивов в БЕЙСИК-системе.

Для вывода построенного графического объекта на графопостроитель предусмотрен оператор %COPY , параметры которого должны учитывать размеры графического поля и величину дискреты графопостроителя.

Для представления координат графических объектов, построенных аналитически или с помощью светового пера, в виде обычных чисел, используется оператор преобразования графических данных в числовые- TRANSFORM.

Все операторы графического взаимодействия, кроме оператора %PEN и операторов стирания, могут быть использованы для работы с графопостроителем, но предварительно должен быть выполнен оператор SELECT PLOT 14.

2.12.3. Основные конструкции и описание операторов.

2.12.3.1. %OPEN .

%OPEN[,] <список %OPEN>

<список %OPEN> ::= <метка символьного массива> / <список %OPEN>, <метка символьного массива>.

Оператор служит для начального открытия одного или нескольких графических объектов.

Если после имени оператора стоит запятая, то наименование и значение экранного графического объекта (заданного в одном из предыдущих операторов) сохраняются, если же запятой после имени оператора нет, то первый идентификатор становится идентификатором заданного графического объекта, при этом происходит очистка экрана дисплея (его графического поля)и установка курсора в левый нижний угол экрана.

По оператору %OPEN в ОЗУ заносятся следующие данные о каждом указанном графическом объекте:

1) длина графического объекта;
2) значения последних координат графического объекта;
3) значения начала координат графического объекта;
4) значения цены деления по осям X и Y.

По выполнении оператора этим данным присваиваются начальные значения. Начальными значениями для длины, последних координат и начала координат графического объекта являются нулевые значения, для цены деления - единичные.

Последними координатами графического объекта считаются координаты, получившиеся в результате рисования предыдущего элемента графического объекта и являющиеся началом строящегося вектора или надписи.

По оператору *ХОРЕМ* в ОЗУ создается служебная зона графического объекта со следующими данными:

1) первые два байта содержат (в двоичном коде) текущую длину графического объекта (первый байт - ее младшие разряды);

2) следующие четыре байта содержат значение последних текущих координат графического объекта (два байта двоичного кода для *Х* и два для *У*);

3) 16 байтов содержат значение начала координат по осям *Х* и *У* (по 8 байтов - представление десятичного числа в машине);

4) 16 байтов содержат цены делений по осям *Х* и *У* (по 8 байтов для *Х* и *У*);

5) следующий байт содержит тип графического элемента и режим (старший, 8 бит равен 1; 3 бит равен 1, если элемент высвечен на экране, и равен 0, если он не светится и может быть стерт; два младших бита кодируют тип графического элемента; 00- точка, 01 - вектор, 10 - надпись).

Далее как указано ниже.

Каждый графический элемент в ОЗУ задается:

1) типом (1 байт);

2) координатами *Х* и *У* для точки, или координатами *Х* и *У* конца для вектора (по 2 байта двоичного кода, байт младших разрядов, затем байт старших), или служебной зоной для надписи;

3) для надписи далее идут байты, содержащие коды символов;

служебная зона надписи состоит из:

1) длины надписи в символах (1 байт двоичного кода);

2) длины надписи по оси *Х* (два байта двоичного кода, первый байт - его младшие разряды);

3) длины надписи по оси *У*;

4) масштаба надписи по отношению к размеру символа 5 * 7 (1 байт двоичного кода);

5) расстояния в надписи между символами по осям *Х* и *У* (по 1 байту двоичного кода).

Объем ОЗУ для графического элемента, таким образом, равен 5 байтам для точки или вектора, и N+9 для надписи, где N - количество символов в надписи. Все графические объекты, выводимые на экран, измеряются в экранных дискретах. Экранная дискрета - это минимальное расстояние между двумя точками на экране, которое определено разрешающей способностью устройства (для дисплея она равна приблизительно 0.4 мм). Допустимая область построения графического объекта, приведенная к экранным дискретам, по осям X и Y есть:

0 <= X <= 32767 0 <= Y <= 32767

допустимая область построения графического объекта значительно превышает размеры экрана (т.е. 0 <= X <= 511, 0 <= Y <=255). Поэтому предусмотрена возможность вывода на экран любой части изображения, либо всего изображения, но в уменьшенном масштабе. Для этого используются операторы метрики и операторы линейного преобразования. Примеры:

```
:10  %OPEN A%(), B%()
:20  %OPEN , C%(),D%()
```

В операторе 10 массив A%() станет экранным графическим объектом, формируется неэкранный графический объект из массива B%().

В операторе 20 экранным графическим объектом попрежнему остается A%(), формируются новые неэкранные графические объекты из символьных массивов C%() и D%().

Примечание. Объявленный по оператору %OPEN экранным графический объект не может быть повторно объявлен неэкранным без предварительного объявления другого графического объекта экранным, в противном случае выдается ошибка (ERR16).

Примеры неправильного открытия графического объекта:

```
1) :10 %OPEN A%(), A%()
2) :10 %OPEN A%()
   :20 %OPEN , B%(),  A%()
```

В первом примере графический объект A%() одновременно объявлен экранным и неэкранным объектом, что является недопустимым.

Во втором примере в операторе 10 объявляется экранный графический объект A%(); в операторе 20 этот же массив, с одной стороны (согласно синтаксису оператора 20) остается экранным, с другой стороны он не может быть экранным, т.к. не является первым идентификатором, что недопустимо.

2.12.3.2. WINDOW .

WINDOW <АВ> , <АВ> , <АВ>, <АВ>

Оператор задает прямоугольную область значений координат экранного графического объекта. Первые два параметра <АВ> задают горизонтальные координаты, а последние два параметра - вертикальные.

При попадании в эту область графические элементы экранного графического объекта выводится на экран дисплея, т.е. оператор задает абсолютные значения экранной области в экранных дискретах.

Значения всех арифметических выражений, входящих в оператор WINDOW, должны быть неотрицательными и не превышать числа 32767, являющегося наибольшим числом в области построений графических объектов; кроме того, должны выполняться условия: первый параметр в операторе WINDOW меньше второго, а третий - меньше четвертого.

При начальной генерации системы значения параметров в операторе WINDOW устанавливаются следующими:

1 параметр - X (левый) равен 0;
2 параметр - X (правый) равен 511;
3 параметр - У (нижний) равен 0;
4 параметр - У (верхний) равен 255.

Абсолютные значения координат графического курсора после выполнения оператора WINDOW становятся равными: X - 1 параметру, У - 2 параметру.

Если к моменту исполнения оператора WINDOW существует экранный графический объект, то курсор устанавливается в левый нижний угол экрана; очистка экрана не производится.

Пример:

Пусть открывается некоторый графический объект (экранный), и внутри в этом графическом объекте формируется вектор, начинающийся в левом нижнем углу экрана (координаты равны 0), а концом имеющий координаты X=511, У=511. Пусть к этому моменту значения арифметических выражений оператора WINDOW соответствуют значениям, сформированным при начальной генерации системы.

В этом случае на экране высветится начальная часть вектора, а в ОЗУ сформируется описание всего вектора с указанными координатами.

2.12.3.3. FRAME .

FRAME <метка символьного массива> , <АВ> , <АВ> , <АВ> , <АВ>

оператор устанавливает для указанного графического объекта минималь-

ное и максимальное значения горизонтальных (первый и второй параметры) и вертикальных (третий и четвертый параметры) координат, входящих в область экрана. Должны быть выполнены условия: первый параметр в операторе FRAME меньше второго; третий - меньше четвертого.

Оператор FRAME задает метрику указанного графического объекта, т.е. положение начала координат и цену делений по осям X и У.

Началом координат будем считать положение точки с координатами (0,0) во введенной оператором FRAME системе единиц пользователя.

Ценой деления по осям X и У считается число экранных дискрет (абсолютных единиц), входящих в одну единицу измерения пользователя по осям X и У.

Пример:

```
:40  WINDOW  1000, 1511, 500, 755
:50  FRAME  A H (), 5, 15, -10, 30
```

После выполнения оператора 50 цена деления в экранных дискретах по осям X и У будет равна:

X=(1511-1000)/(15-5)=51.1 У=(755-500)/(30-(-10))=6.375

Начало координат:

В экранных дискретах X0=-5*51.1=-255.5 У0=-(-10)*6.375=63,75

В единицах пользователя X=1 У=1; X0=-5 У0=-(-10)=10

Обозначим через W1, W2, W3, W4 параметры оператора WINDOW, через F1, F2, F3, F4 параметры оператора FRAME, через X0,Y0 - координаты точки в экранных дискретах, через X,У - координаты точки в системе координат пользователя. Тогда начало координат пользователя в экранных дискретах и связь этих двух систем координат выражается так:

X0=-F1*(W2-W1)/(F2-F1) X0=(X-F1)*(W2-W1)/(F2-F1)

Y0=-F3*(W4-W3)/(F4-F3). Y0=(Y-F3)*(W4-W3)/(F4-F3)

Если требуется оператором FRAME установить минимальное и максимальное значения горизонтальных (V1,V2) и вертикальных (V3,V4) координат пользователя, входящих в область экрана, то параметрами оператора FRAME следует задать такие величины:

F1=(W2*V1-W1*V2)/(W2-W1) F2=(W2*V2+W1*V1-2*W1*V2)/(W2-W1)

F3=(W4*V3-W3*V4)/(W4-W3) F4=(W4*V4+W3*V3-2*W3*V4)/(W4-W3)

Например, чтобы при заданных W1=1000; W2=1511; W3=500; W4=755

установить V1=5; V2=15; V3=-10; V4=30

следует задать F1=-7445/511; F2=-2325/511;

F3=-22550/255; F4=-12350/255

тогда получается: X0=744.5; Y0=563.75

2.12.3.4. ORIGIN .

ORIGIN <метка символьного массива>, <AB> , <AB>

Оператор устанавливает новое значение начала координат для указанного графического объекта.

Параметры <AB> задают значения нулевой координаты по осям X и У соответственно во введенной системе измерения координат для указанного графического объекта.

Оператор ORIGIN не работает с операторами, использующими относительные координаты (DDOT, DDRAW и т.д.).

Пример:

Пусть для графического объекта BЖ() оператором FRAME была установлена цена деления по осям X и У: X=4, У=0.5 экранных дискрет.

:10 ORIGIN BЖ(), -20,40

После выполнения оператора 10 положение нулевой точки (начала координат) будет:

X0=4 * (-20)=-80 У0=0.5 * 40=20 экранных дискрет.

Положение нулевой точки может находиться и вне экранной области, как это видно из данного примера (X0 <0).

2.12.3.5. SCALE .

SCALE <метка символьного массива> , <AB> , <AB>

Оператор устанавливает значения новой цены деления (единицы измерения) по осям X и У.

Значения параметров <AB> должны быть положительны .

Первый параметр <AB> указывает, сколько предыдущих единиц измерения координат по оси X содержится в одной новой единице по оси X, а второй параметр <AB> указывает, сколько предыдущих единиц по оси У содержится в одной новой единице по оси У, т.е. новые единицы измерения вычисляются относительно старых, уже существующих единиц в указанном графическом объекте.

Пример:

:50 SCALE AЖ(), 0.2,5

После выполнения оператора 50 новая единица измерения по оси X стала в пять раз мельче, а по оси У - в пять раз крупнее.

2.12.3.6. DOT и DDOT.

DOT <метка символьного массива> ,<AB> ,<AB>

DDOT <метка символьного массива>, <AB> ,<AB>

Операторы используются для построения в указанном графическом объекте точки с координатами, определяемыми первым и вторым парамет-

рами <AB> по оси X и У соответственно. В операторе DOT параметры <AB> определяют координаты точки. В операторе DDOT -приращения значений координат по отношению к последним координатам графического объекта от предыдущих построений.

Примеры:

1) DOT A#(), 50, A+3

2) DDOT A#(), I5,-I0

В первом примере строится точка в объекте A#() с координатами по оси X=50, по оси У=A+3.

Во втором примере строится точка в объекте A#() с координатами по оси X=XI+I5, по оси У=YI-I0, где XI,YI - значения последних построенных координат графического объекта. Если это первый графический элемент в графическом объекте A#(), то эти значения - нулевые. Все координаты берутся с учетом метрики указанного графического объекта.

2.I2.3.7. DRAW и DDRAW.

DRAW <метка символьного массива> ,<AB> , <AB>

DDRAW <метка символьного массива> , <AB> , <AB>

Операторы служат для построения вектора в указанном графическом объекте. Координаты конца вектора по оси X и У определяется первым и вторым параметрами <AB> соответственно , причем, если используется оператор DRAW , то координаты конца вектора совпадают со значениями параметров <AB> , а если используется оператор DDRAW , то значения параметров <AB> задают координаты конца вектора через приращения значений координат по отношению к последним координатам графического объекта.

Примеры:

1) DRAW A#(), -I0,I5

2) DDRAW A#(), 5,-15

В первом примере строится вектор с началом в точке, определяемой последними координатами графического объекта A#() , и концом в точке XK=-I0, YK=I5. Во втором примере строится вектор с началом аналогично первому примеру и концом в точке XK=X1+5, YK=Y1-I5.

2.I2.3.8. NPLOT и DNPLOT.

NPLOT <метка символьного массива> ,<AB> , <AB>

DNPLOT <метка символьного массива > ,<AB> , <AB>

Операторы устанавливают значение последних координат указанного графического объекта, равное значению первого и второго параметров

<AB> по оси X и оси Y соответственно для оператора NPLOT и сумме значений последних координат графического объекта и значений параметров <AB> для оператора DNPLOT.

Примеры:

1) :NPLOT BX(), 100, 300

Последними координатами графического объекта BX() устанавливаются координаты XI=100 и YI=300;

2) пусть последними координатами до выполнения оператора DNPLOT были XI=150, YI=200;

:100 DNPLOT BX(), 50, -100

После выполнения оператора 100 последние координаты определяются таким образом: X2=150+50=200, Y2=200+(-100)=100

Операторы NPLOT и DNPLOT не вызывают построений, поэтому, если указанный в операторе графический объект - экранный, то происходит движение курсора без рисования в заданное новыми последними координатами место экрана.

Примечания:

1. Каждая точка и вектор занимают в ОЗУ по пять байтов.

2. При использовании операторов NPLOT и DNPLOT система автоматически принимает меры по оптимизации использования ОЗУ.

3. Поскольку хранение последних координат графического объекта осуществляется в двоичном коде, то возможна погрешность вычисления координат в операторах DDOT, DDRAW, DNPLOT, равная одной экранной дискрете.

Пример:

:100 DDOT A$ (), 10, 100

:110 DDOT A$ (), -10, -100

Выполнение этих двух операторов для экранного графического объекта может дать визуальное изображение двух несливающихся точек, находящихся на расстоянии одной экранной дискреты.

При использовании операторов построения в приращениях внутри цикла погрешность, если она есть, накапливается.

2.12.3.9. LABEL

LABEL <метка символьного массива>[(<AB>)], [(<AB>)], [(<AB>)]<список LABEL>

<список LABEL>::= <символьное значение> / <AB> / <список LABEL>, <символьное значение> / <список LABEL>, <AB>

Оператор LABEL предназначен для построения надписи в указанном графическом объекте.

Первый параметр <AB> задает масштаб выводимых символов надписи;

при его отсутствии принимается масштаб, равный единице.

Размеры символа равны 5*7 экранных дискрет.

Второй и третий параметры <AB> задают расстояние между символами надписи по осям X и Y соответственно: при отсутствии второго параметра <AB> расстояние по оси X равно двум экранным дискретам, а при отсутствии третьего параметра <AB> - расстояние по оси Y равно нулю.

Значения параметров <AB> должны быть положительными, не равными нулю и не превышать 255.

При выполнении оператора данные из списка элементов надписи заносятся в описание указанного графического объекта в ОЗУ и, в случае, если графический объект - экранный, выводятся на экран в виде последовательности символов. Форматы представления элементов надписи соответствуют форматам представления данных на внешних носителях.

Начальной точкой надписи является точка с текущими координатами, последними перед надписью в процессе построения указанного графического объекта. Конечной точкой при построении надписи является точка с координатами, определяемыми следующим образом:

X2=X1+(5*M+X0)*N Y2=Y1+N*Y0,

где X1, Y1 - последние координаты указанного графического объекта,

N - число символов надписи,

M - масштаб символов,

X0, Y0 - расстояние между символами в экранных дискретах.

Следует иметь в виду, что после построения надписи оператором LABEL последние координаты указанного графического объекта не определены. Чтобы их определить для дальнейших построений в этом графическом объекте, необходимо использовать операторы DOT или MPLOT, а затем производить следующие построения.

Если при построении надписи в экранном графическом объекте оказывается, что в область экрана, задаваемую оператором WINDOW, попала лишь часть надписи (с учетом последних координат надписи), то надпись не выводится на экран, а заносится лишь в описание графического объекта в ОЗУ. Вывести на экран построенную надпись в этом случае можно по оператору WINDOW с параметрами, позволяющими таким образом включить надпись в экранную область.

Пример:

```
:100 *OPEN A#(), B# ()
:200 WINDOW 0, 500, 0, 250
```

:300 LABEL A%() 4,,, B%(I), A+3

В примере строится надпись в экранном графическом объекте A%(), параметры "окна" на экране заданы оператором WINDOW : Если надпись на экране не появилась . (параметры WINDOW заданы неточно), необходимо выполнить следующие операторы:

:400 WINDOW 100, 600, 100,350

:500 XLET A% ()= A% ()

По оператору XLET произойдет перевывод экранного графического объекта A% () на новую область экрана, заданную оператором 400.

Примечания:

1. Если последние координаты надписи находятся за пределами области, определенной по оператору WINDOW , то визуальных изменений надписи на экране, если графический объект - экранный (по операторам стирания, построения, преобразования) не происходит, но в описании графического объекта в ОЗУ все действия выполняются.

2 . Каждая надпись занимает в ОЗУ (N+9) байтов , где N -число символов в надписи.

2.12.3.10 NDOT .

NDOT <метка символьного массива> , <AB> , <AB>

Оператор используется для стирания точки из указанного графического объекта, координаты которой заданы значениями параметров <AB>. Если графический объект является экранным, и точка с координатами, равными значениям параметров <AB> , входит в экранную область, то она стирается с экрана. Если в описании графического объекта несколько точек с одинаковыми координатами, то стирается только первая. Если точка с указанными координатами не найдена, то выдается сообщение об ошибке (NER 18).

Пример:

:NDOT A % (), 5, 317

2.12.3.11. NDRAW.

NDRAW <метка символьного массива> , <AB> , <AB>

Оператор предназначен для стирания вектора из указанного графического объекта, координаты конца вектора заданы параметрами <AB>.

Если графический объект является экранным, и вектор с координатами конца (параметры <AB>) входит в экранную область, то вектор (или его часть, которая помещена внутри экранной области) стирается с экрана. Если в описании графического объекта несколько векторов с одинаковыми координатами конца вектора, то стирается только первый в описании вектор. Если вектор с указанными координатами конца в данном графическом объекте не найден, выдается сообщение об ошибке (ERR 18).

Примечание. Операторы *NDOT* и *NDRAW* не изменяют последних координат указанного графического объекта; в случае стирания точки (вектора) на экране курсор после стирания возвращается на старое место.

Пример:

```
:NDRAW A LET (),  K+7,  A (5) +6
```

2.12.3.12. LET

LET <параметр LET>

<параметр LET> ::= <метка символьного массива> =0 / <ГЭ> =0 / <метка символьного массива> = <список LET>

<список LET> ::= <метка символьного массива> / <ГЭ> / <список LET> + <метка символьного массива> / <список LET> + <ГЭ>

<ГЭ> ::= <идентификатор переменной> (<AB> [,])

Так как графический элемент <ГЭ> может быть частью графического объекта, созданного из двумерного символьного массива, то после номера элемента <AB>, ставится запятая.

Оператор LET служит для построения новых графических объектов из уже построенных графических объектов, их объединения и объединения частей графических объектов (графических элементов).

Под объединением графических объектов понимается подсоединение в области значений графического объекта из левой части оператора последовательной совокупности значений графических объектов из правой части оператора LET.

Если эта совокупность не помещается в отведенное в [illegible] место для графического объекта из левой части оператора LET, то выдается сообщение об ошибке (ERR 18).

Один и тот же графический объект может ставиться в левой и правой частях одного оператора LET. В этом случае графический объект в левой части оператора модифицируется.

Оператор LET используется также для создания одного графического объекта из части другого графического объекта.

Если в левой части оператора употребляется экранный графический

объект, то вся совокупность графических объектов и графических элементов из правой части оператора выводится на экран.

Если в правой части оператора стоит нуль, то значения графического объекта или графического элемента из левой части оператора стираются из ОЗУ, а если этот графический объект - экранный или графический элемент принадлежит экранному графическому объекту, то и с экрана .

По оператору #LET можно осуществлять перевывод экранного графического объекта на новую область экрана, заданную оператором WINDOW . Примеры:

```
1) : #LET A#() = A#() + B#(5)
2) : #LET A#() = B#() + C#() + D#(8)
3): #LET A#() = 0:#LET B#(A2) = 0
4) : 60 #LET A#() = 0
   : 70 FOR A=20 TO 40
   : 80 #LET A#() = A#() + D#(A): #NEXT A
```

В примере 1 к предыдущему (до оператора #LET) значению графического объекта добавится значение пятого элемента из графического объекта B #().

В примере 2, если A#() - экранный графический объект, то на экран выведется содержимое графических объектов B#(), C#() и восьмого элемента из графического объекта D#(), входящее в экранную область, заданную оператором WINDOW .

В примере 3 значения графического объекта A#() стираются из ОЗУ. Значения графического элемента с указанным номером стираются из области значений графического объекта B#(); если графический элемент с указанным номером не найден в данном графическом объекте, то выдается сообщение об ошибке.

В примере 4 после выполнения оператора 60 в графическом объекте A#() отсутствует графический элемент; после выполнения операторов 70 и 80 графический объект A #() заполняется графическими элементами, входящими в графический объект D#() ; номера этих элементов в графическом объекте D#() - с 20 по 40 ; в графическом объекте A #() эти элементы будут иметь номера с 1 по 21 .

Примечания:

1. Использование оператора #LET не меняет метрику графических объектов, входящих в оператор #LET, т.е. метрика (система единиц

измерения координат) сохраняется, даже если в правой части оператора стоит нуль.

2. При построении графического объекта из уже построенных графических элементов такие графические элементы, как вектор и надпись, подсоединяются к области значений графического объекта с учетом начала и конца вектора (надписи). Если графический объект - экранный, местоположение вектора (надписи) на экране остается после подсоединения прежним.

3. Если в правой части оператора ЖLET употребляется тот же графический объект, что и в левой, то он должен стоять сразу после знака равенства и быть единственным, в противном случае выдается сообщение об ошибке (ERR 10). Если в правой части стоит элемент указанного в левой части графического объекта, то порядок следования графического объекта и графического элемента после знака равенства произвольный.

Например:

1) : ЖLET BЖ() = BЖ() + AЖ(), но не ЖLET BЖ()= AЖ()+BЖ()

2) : ЖLET BЖ() = AЖ() + BЖ(I0) и ЖLET BЖ()=BЖ(I0)+AЖ()

2.I2.3.I3. ЖMOVE .

ЖMOVE <метка символьного массива> , <AB> ,<AB>

оператор осуществляет сдвиг графического объекта по осям X и У на величину, определяемую параметрами <AB> соответственно.

2.I2.3.I4. STRETCH .

STRETCH<метка символьного массива> , <AB> , <AB> , <AB> , <AB>

Оператор осуществляет растяжение или сжатие графического объекта относительно точки с координатами, соответствующими значениям первых двух параметров <AB> , в число раз по оси X, заданное значением третьего параметра <AB> ; по оси У - четвертым параметром <AB> .

Если значение третьего и четвертого параметров <AB> больше I, то происходит растяжение графического объекта, если меньше I, то - сжатие.

Значения третьего и четвертого параметров должны быть положительны.

2.I2.3.I5. TURN .

TURN <метка символьного массива>, <AB> , <AB> , <AB>

оператор осуществляет поворот графического объекта относительно точки с координатами, заданными первым и вторым параметрами <AB>

на угол задаваемый третьим параметром <AB>.

Единицы измерения угла задаются оператором SELECT D/ R / G.

Нулевое значение угла - положительное направление по оси X, положительное приращение угла - против часовой стрелки.

Если преобразуемый графический объект - экранный, то по окончании преобразования новый графический объект выводится на экран. Значения координат нового графического объекта не должны выходить за область построения графического объекта, иначе выдается сообщение об ошибке (ERR 57).

Пример:

```
: TURN AЖ (),  I20,  I0,  -20
```

2.I2.3.I6. ЖCOPY .

ЖCOPY [<ФАУ>,] <метка символьного массива> [, <AB> , <AB >]

Оператор служит для вывода построенного графического объекта на графопостроитель, экран дисплея или другое графическое устройство.

Параметры <AB> задают положение начальной точки, измеренное в экранных дискретах.

Значения параметров <AB> должны быть положительными, не равными нулю и не превышать 64000. Область вывода графического объекта задается оператором WINDOW .

Адрес устройства должен быть задан оператором SELECT PLOT или непосредственно в операторе ЖCOPY параметром <ФАУ>.

Пример:

```
:ЖCOPY  / I0 , BЖ (),  20, I0
```

2.I2.3.I7. TRANSFORM .

TRANSFORM <метка символьного массива> TO <метка числового массива> оператор используется для присвоения значений координат графического элемента построенного графического объекта элементам числового массива.

Все нечетные элементы числового массива получают значение координаты X, а четные координаты У. Если число элементов массива недостаточно для присвоения значений всех графических элементов из указанного графического объекта, то выдается сообщение об ошибке (EPR I8).

Пример:

```
:TRANSFORM CЖ ()  TOK ()
```

Примечание. Если число пар элементов числового массива больше числа графических элементов в указанном графическом объекте, то оставшиеся элементы числового массива сохранят старые значения.

2.12.3.18. *PEN.

*PEN <метка символьного массива>

Оператор предназначен для работы со световым пером. При выходе программы на оператор *PEN выполнение текущей программы прекращается, и управление передается пользователю.

Существуют следующие режимы работы со световым пером:

1) ведение графического курсора по экрану;
2) рисование произвольной кривой из точек;
3) рисование векторов;
4) распознавание указанных пером экранных графических элементов;
5) стирание указанных элементов изображения.

Переключение режимов работы с пером производится нажатием клавиш специальных функций (коды клавиш HEX (5B) - HEX(60)). Включение самого пера производят кнопкой, расположенной на пере.

В режиме ведения графического курсора (клавиша HEX(5B)) пользователь, захватив пером курсор, выводит его в требуемое место экрана.

В режиме рисования точечной кривой (клавиша HEX (5C)) пользователь при ведении пером курсора рисует на экране кривую из точек. Если скорость ведения пера высока, то точки располагаются с интервалом 5-7 экранных дискрет; если мала, то с интервалом 1-3 дискреты.

В режиме рисования векторов после первого нажатия клавиши режима (HEX (5D)) устанавливается режим ведения курсора. По следующему нажатию клавиши рисования вектора до текущего положения курсора рисуется вектор, и устанавливается режим ведения курсора; по очередному нажатию клавиши рисования вектора рисуется следующий вектор и т.д.

В режиме распознавания указанных пером элементов экранного графического объекта (клавиша HEX (5E)) пользователю высвечивается на экране следующая информация об элементе:

1) номер графического элемента в экранном графическом объекте;
2) тип графического элемента (точка, вектор, надпись);
3) координаты указанного графического элемента в метрике экранного графического элемента, причем для вектора выдаются конечные координаты, а для надписи - координаты начала надписи.

Например, при нажатии клавиши распознавания (HEX(5E)) световое перо указывает одну из точек кривой - тогда на экране высветится информация вида:

POINTED A Ж(42) DOT X=237.56 Y=-36.639

где число 42 - номер элемента в графическом объекте А Ж() - экранном графическом объекте,

DOT - тип графического элемента - точка,

X и Y - координаты точки, указанной световым пером, их значения - в 42 графическом элементе графического объекта А Ж().

Графический объект, стоящий в операторе Ж PEN, должен быть экранным графическим объектом, открытым оператором Ж OPEN, иначе выдается сообщение об ошибке (ERR 18).

Выход из оператора Ж PEN осуществляется по нажатии клавиши спецфункций HEX(5F) - при этом возобновляется счет по программе, прерванный по оператору Ж PEN.

При подведении курсора к ненужному элементу на экране и нажатии клавиши стирания (HEX(60)) указанный элемент стирается с экрана, а его данные убираются из описания экранного объекта, который хранится в ОЗУ.

Кнопку включения пера следует нажимать только после фиксации положения наконечника пера относительно выбранного элемента изображения при перпендикулярном положении пера относительно плоскости экрана.

В режиме распознавания в последние два байта графического объекта заносится номер графического элемента (в двоичном коде), последнего из распознанных светопером. Если этот номер нужен для работы, то после распознавания следует выйти из оператора Ж PEN, запомнить или обработать эти данные и только после этого вернуться в оператор Ж PEN.

Рекомендуется исполнять оператор Ж PEN, т.е. работать со световым пером, в те промежутки времени, когда не используется таймер текущего времени.

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

БИФ	- блок интерфейсный функциональный
Восьмер.	- восьмеричный
ОЗУ	- оперативное запоминающее устройство
П.	- пункт
ПЭКВМ	- программоуправляемая электронная клавишная вычислительная машина
ПИД	- процессор интерпретирующий диалоговый
РОН	- регистр общего назначения
См.	- смотри
Табл.	- таблицы
т.д.	- так далее
т.к.	- так как
т.е.	- то есть
т.п.	- тому подобное
УП	- управляющая память
ФАУ	- физический адрес устройства

ОПИСАНИЕ ЯЗЫКА
ПРИЛОЖЕНИЕ
00017-01 35 01 - 2

I. ТЕРМИНЫ ЯЗЫКА БЕЙСИК 02 И ПРАВИЛА СИНТАКСИЧЕСКОГО ОПИСАНИЯ

I.I. Правила синтаксического описания.

< > - обозначение элементов в конструкции.
[ ] - обозначение необязательных элементов в конструкции.
/ - "ИЛИ".
::= - равно по определению.

I.2. Основные конструкции.

I.2.I. <символьная переменная> ::= <символьный операнд> / <метка символьного массива>

<символьный операнд> :: =<простая символьная переменная> / <элемент символьного массива> / STR (<символьный элемент> , <AB>[, <AB>])

<символьный элемент> :: = <простая символьная переменная> /<элемент символьного массива> / <метка символьного массива>

<простая символьная переменная> :: = <идентификатор переменной>$

<идентификатор переменной>:: = <буква> / <буква> <цифра>

<буква> :: = A/B/C/D/E/F/G/H/I/J/ K/L/M/N/ O/ P/Q/R/S/ T/U/V/W/X/ Y/ Z

<цифра> :: = 0/1/2/3/4/5/6/7/8/9

<элемент символьного массива>:: = <идентификатор переменной>$(<AB>)/ <идентификатор переменной>$(<AB> , <AB>)

<символьная константа> :: ="<цепочка символов>" / "<цепочка символов>" / HEX (<цепочка элементов HEX>)

<цепочка элементов HEX> :: = <элемент HEX>/ <цепочка элементов HEX> <элемент HEX>

<элемент HEX> :: =<шестнадцатиричная цифра><шестнадцатиричная цифра>

<шестнадцатиричная цифра> :: = 0/1/2/3/4/5/6/7/8/9/A/B/C/D /E/F

<цепочка символов> :: = <символ> / <цепочка символов> <символ>

<символьный аргумент> :: = <символьная переменная> / <символьная константа>

<символьный аргумент 1> :: = <символьный аргумент> / <элемент HEX>

<символьное значение> :: = <символьный операнд> / <символьная константа>.

I.2.2.<числовая переменная> :: =<числовой элемент> /<метка числового массива>

<числовой элемент> :: =<простая числовая переменная> / <элемент числового массива>

<простая числовая переменная>:: =<идентификатор переменной> [%]

<элемент числового массива> :: =<идентификатор переменной>[%](<AB>)/
<идентификатор переменной>[%](<AB>, <AB>).

I.2.3.<метка массива> :: = <метка числового массива> / <метка символьного массива>

<метка числового массива> :: = <идентификатор переменной> [%] ()

<метка символьного массива> :: =<идентификатор переменной> $ ().

I.2.1. <AB> :: = [+]<операнд>[<полином>] / - <операнд>[<полином>]

<полином> :: = <арифметическая операция> <операнд> / <полином> <арифметическая операция><операнд>

<арифметическая операция>:: =<сложение> / <вычитание> / <умножение / <деление> / <возведение в степень>

<сложение> :: = +

<вычитание>:: = -

<умножение>:: = * <деление> ::=/

<возведение в степень>:: = ^

<операнд> :: =<числовая конотанта> / <числовой элемент> /<функция> /(<операнд>[<полином>])

<числовая конотанта> :: =# PI / <число>

<число>:: =<целое>[.<целое>][E<порядок>] /[<целое>].<целое>[E <порядок>]

<целое> :: = <цифра> / <целое> <цифра>

<порядок> :: = + <цифра> / + <цифра><цифра> / -<цифра> /-<цифра><цифра>

<функция> :: = <элементарная функция> / <функция символьного аргумента> / FN(<цифра>(<AB>)) / FN(<буква> <AB>)) / ROUND(<AB>,<AB>)

<элементарная функция> :: =<имя функции>(<AB>)

<имя функции> :: = SIN / COS / TAN / ARCSIN/ ARCCOS / ARCTAN / SQR / SGN / RND / LOG / INT/ EXP/ ABS

<функция символьного аргумента> :: =NUM (<символьный аргумент>) / LEN(<символьный аргумент>) / POS (<символьный аргумент> <операция сравнения> <символьный аргумент I>) / VAL (<символьный аргумент>[,2])

<имя FN>:: =<буква> / <цифра>

<операция сравнения> :: =<меньше>/ <меньше или равно> / <больше> /<больше или равно> / <равно> / <не равно>

<меньше> :: =<

<меньше или равно> :: = <=

<больше> :: = >

<больше или равно> :: = >=

<равно> :: = =

<не равно>:: =<>.

I 4.5. <список переменных>:: = <числовая переменная>/ <символьная переменная>/<список переменных> ,<числовая переменная>/ <список переменных>, <символьная переменная>

<список аргументов>:: = <числовая переменная>/<символьный аргумент>/ <AB><список аргументов>, <числовая переменная>/<список аргументов>,<символьный аргумент>/ <список аргументов>, <AB>

<список для присвоения>:: = <числовой элемент>/ <символьный операнд>/ <список для присвоения>, <числовой элемент>/<список для присвоения>, <символьный операнд>

<диапазон строк>::=[<номер строки>] , <номер строки>/ <номер строки> [.<номер строки>]

<номер строки>:: = <цифра>/ <цифра><цифра>/ <цифра><цифра><цифра>/<цифра><цифра><цифра><цифра>.

I.3. Основные операторы.

I.3.I. CLEAR[<параметр CLEAR>]

<параметр CLEAR>:: = U/N/ P[<диапазон строк>].

I.3.2. COM <список объявлений>

<список объявлений>:: = <объявление> /<список объявлений>, <объявление>

<объявление>:: = <объявление простой переменной>/ <объявление массива>

<объявление простой переменной>:: =<простая числовая переменная>/ <простая символьная переменная>[<длина>]

<длина>:: = <цифра>/ <цифра><цифра> / <цифра><цифра><цифра>

<объявление массива>:: =<объявление целого одномерного массива>/ <объявление целого двумерного массива>/ <объявление действительного одномерного массива>/<объявление действительного двумерного массива>/ <объявление символьного одномерного массива>/<объявление символьного двумерного массива>

<объявление целого одномерного массива>:: =<идентификатор переменной> %(<размерность>)

<объявление целого двумерного массива>:: =<идентификатор переменной> % (<размерность> , <размерность>)

<объявление действительного одномерного массива>:: =<идентификатор переменной> (<размерность>)

<объявление действительного двумерного массива>:: =<идентификатор переменной>%(<размерность> , <размерность>)

<объявление символьного одномерного массива> :: =<идентификатор переменной>$(<размерность>) [<длина>]

<объявление символьного двумерного массива> ::= <идентификатор переменной>$(<размерность> , <размерность>)[<длина>]

<размерность> :: = <цифра> / <цифра><цифра>/ <цифра><цифра><цифра> / <цифра><цифра><цифра><цифра> / <цифра><цифра><цифра><цифра><цифра>

I.3.3. COM CLEAR[<параметр COM CLEAR>]

<параметр COM CLEAR> :: = <числовая переменная> / <символьный элемент>

I.3.4. DATA <список DATA>

<список DATA> :: = <элемент DATA> /<список DATA>, <элемент DATA>

<элемент DATA> :: =<числовая константа> / <числовой элемент> / <символьная константа> / <символьный операнд>

I.3.5. DEFFN <имя FN>(<простая числовая переменная>) = <AB>

<имя FN>:: = <цифра> / <буква>

I.3.6.DEFFN'<номер DEFFN'>[(<список для присвоения>)] / DEFFN'<номер DEFFN'><список символьных конотант>

<номер DEFFN'>:: = <цифра> / <цифра><цифра> / <цифра><цифра><цифра>

<список символьных конотент>:: = <символьная конотанта> /<список символьных конотант>; <символьная константа>

I.3.7. DIM <список объявлений> .

I.3.8. END .

I.3.9 FOR <простая числовая переменная> = <AB> TO <AB>[STEP <AB>].

I.3.10. GOSUB <номер строки> .

I.3.11. GOSUB' <номер DEFFN'>(<список параметров GOSUB'>)

<список параметров GOSUB'>:: = <AB> /<символьное значение> /<список параметров GOSUB'>,<AB> / <список параметров GOSUB'>, <символьное значение> .

I.3.12. GOTO <номер строки> .

I.3.13. IF <условие> THEN <номер строки>

<условие> :: =<выражение>[<список логических элементов>]

<выражение>:: = <AB><операция сравнения> <AB> / <символьный аргумент><операция сравнения> <символьный аргумент>

<список логических элементов> :: = <логическая операция> /<список логических элементов><логическая операция>

<логическая операция> :: = AND <выражение> / OR <выражение> / XOR <выражение>

I.3.I4. INIT (<символьный аргумент>)<список символьных переменных>

<список символьных переменных> :: = <символьная переменная> / <список символьных переменных> , <символьная переменная>

I.3.I5. [LET]<присвоение> .

<присвоение> :: = <список числовых элементов> = <АВ> /<список символьных операнд> = <символьное значение>

<список числовых элементов> :: = <числовой элемент> /<список цифровых элементов> ,<числовой элемент>

<список символьных операнд >:: = <символьный операнд> /<список символьных операнд>, <символьный операнд>

I.3.I6. MATCOPY [-] <символьная переменная> TO [-]<символьная переменная>

I.3.I7. MATSEARCH <символьная переменная> ,<условие> <символьный аргумент> TO <символьная переменная> [STEP <АВ>]

<условие> :: = < /> / <> / >= / <= / = /*/ # <символьный аргумент> , / %

I.3.I8. NEXT <простая числовая переменная> .

I.3.I9. ON <АВ> <параметр перехода> <список строк>

<параметр перехода> :: = GOTO / GOSUB

<список строк> :: = <номер строки> / <список строк> , <номер строки>

I.3.20. ON ERROR[<символьная переменная> ,<символьная переменная> <параметр ON ERROR> <номер строки>

<параметр ON ERROR> :: = GOTO / GOSUB / THEN

I.3.2I. READ <список для присвоения> .

I.3.22. REM [<цепочка символов>] .

I.3.23. RENUMBER [<номер строки>][, <новый номер>]

<новый номер> :: = <номер строка> ,<шаг> / <номер строки> /, (шаг)

<шаг> :: = <цифра> / <цифра><цифра>

I.3.24 REPLACE <числовой элемент> , <символьная переменная> , <символьный аргумент> [, <символьный аргумент>]

I.3.25. RESTORE [<АВ>][,<номер строки>] .

I.3.26. RETURN .

I.3.27. RETURN CLEAR [ALL] .

I.3.28. RUN[<номер строки>] .

I.3.29. SELECT <список SELECT> .

<список SELECT> :: = <параметр SELECT> /<список SELECT> ,<параметр SELECT>

<параметр SELECT> :: = CI <элемент HEX> / CO <элемент HEX> [(<длина строки>)] TAB <элемент HEX> [(<длина строки>)] / LIST <элемент HEX> [(<длина строки>)] / PRINT <элемент HEX> [(<длина строки>)] DISK <элемент HEX> P / DISK <элемент HEX> R / # <цифра> <элемент HEX> [R] / # <цифра> <элемент HEX> [R] / P [<цифра>]/D / R / C

<длина строки> :: = <цифра> /<цифра><цифра>/<цифра><цифра><цифра>/<цифра><цифра><цифра><цифра>

I.3.30. STOP [<символьная константа>][#] .

I.3.31 TRACE [OFF] .

I.4. Операторы символьной обработки .

I.4.1. ADD[C]<логические аргументы> .

<логические аргументы> :: = (<символьная переменная> ,<символьный аргумент I>)

I.4.2. AND <логические аргументы> .

I.4.3. BIN (<символьная переменная> [, 2]) = <AB> .

I.4.4. BOOL <шестнадцатиричная буква> <логические аргументы> .

I.4.5. CONVERT <преобразование> .

<преобразование> :: = <символьная переменная> TO <числовая переменная> / <AB> TO <символьная переменная> ,(<формат>)

<формат> :: =<цепочка символов> / <символьный операнд>

I.4.6. OR <логические аргументы> .

I.4.7. PACK (<формат>)<символьная переменная> FROM <список PACK> .

<список PACK> :: = <AB> / <числовая переменная> / <список PACK>, <AB> / <список PACK> ,<числовая переменная>

I.4.8. ROTATE [C] (<символьная переменная>, <AB>) .

I.4.9. UNPACK (<формат>) <символьная переменная> TO <список UNPACK>

<список UNPACK> :: = <числовая переменная> / <список UNPACK>, <цифровая переменная> .

I.4.10. XOR <логические аргументы> .

I.4.11. XPACK [(<формат упаковки>)] <символьная переменная> [,

<числовая переменная>] FROM <список переменных>
<формат упаковки> :: = F = <символьная переменная> / D = <символьная переменная>

I.4.I2. #TRAN (<символьная переменная>, <символьная переменная>)[<элемент HEX>] [R]

I.4.I3. #UNPACK [(<формат упаковки>)] <символьная переменная>[, <числовая переменная>] TO <список переменных>

I.5. Матричные операторы.

I.5.I. MAT <числовой массив> = <числовой массив>.
<числовой массив> :: = <идентификатор переменной>[%]

I.5.2. MAT <числовой массив> = ZER [(<AB> [, <AB>])].
I.5.3. MAT <числовой массив> = CON [(<AB> [, <AB>])].
I.5.4. MAT <числовой массив> = IDN [(<AB>[, <AB>)].
I.5.5. MAT <числовой массив> = <числовой массив> + <числовой массив>.
I.5.6. MAT <числовой массив> = <числовой массив> - <числовой массив>.
I.5.7. MAT <числовой массив> = <числовой массив> # <числовой массив>.
I.5.8. MAT <числовой массив> = (<AB>)# <числовой массив>.
I.5.9. MAT <числовой массив> = TRN(<числовой массив>).
I.5.I0. MAT <числовой массив> = INV (<числовой массив>) [, <числовой элемент>] ;
I.5.II. MATREAD <список массивов>.
<список массивов> :: = <идентификатор переменной> [%][(<AB> [, <AB>])] / <идентификатор переменной> # [(<AB>[, <AB>])] / <список массивов> , <идентификатор переменной> [%] [(<AB>[, <AB>])] / <список массивов>, <идентификатор переменной># [(<AB>[, <AB>])].

I.5.I2. MATREDIM <список массивов>.

I.5.I3. MATINPUT <список массивов>.
I.5.I4. MATPRINT [<устройство>,] <список MATPRINT> [<разделитель>]
<список MATPRINT> :: = <идентификатор переменной>[%] / <идентификатор переменной># / <список массивов> <разделитель> <идентификатор переменной> [%] / <список массивов> <разделитель> <идентификатор переменной> #

I.6. Операторы сортировки.

I.6.I. MATCONVERT <метка числового массива> TO <метка символьного массива> [(<AB>, <AB>)].

I.6.2. MATMERGE <метка символьного массива> TO <метка символьного массива>, <метка символьного массива>, <метка символьного массива>.

I.6.3. MATMOVE <метка числового массива>, <элемент символьного массива> [, <числовая переменная>] TO <элемент числового массива> / MATMOVE <метка символьного массива> [(<AB>, <AB>)] <элемент символьного массива> [, <числовая переменная>] TO <элемент символьного массива>.

I.6.4. MATSORT <метка символьного массива> TO <метка символьного массива>, <метка символьного массива>.

I.7. Операторы ввода-вывода.

I.7.I. DATA LOAD BT [<устройство>,] <символьная переменная>
<устройство> :: =<логический номер> / <ФАЙ>
<логический номер> :: = #<AB>
<ФАЙ> :: = <символ /> <элемент HEX> / <символ /> <символьная переменная>

I.7.2. DATA SAVE BT [<устройство>,] <символьная переменная>.

I.7.3. HEXPRINT [<устройство>,] <список HEXPRINT> [<разделитель>]
<список HEX PRINT> :: =<элемент HEXPRINT> /<список HEXPRINT> <разделитель> <элемент HEX PRINT>
<разделитель> :: = <символ ,> / <символ ;>
<элемент HEXPRINT> :: = <символьный аргумент> / TAB(<AB>).

I.7.4. IF END THEN<номер строки>.

I.7.5. % <цепочка символов>.

I.7.6. INPUT [<символьная константа>,] <список переменных> / INPUT # <числовой элемент>.

I.7.7. KEYIN <символьная переменная>, <номер строки>, <номер строки>.

I.7.8. LINPUT [<символьная константа>] [-] <символьная переменная>.

I.7.9. LIST <параметр>.
<параметр> :: = [<символ #>] [<устройство>] / [<символ #>] [<устройство>,] <элемент LIST>

<элемент LIST>:: = % /[A][<диапазон строк>]/ <символ'>[<номер DEFFN'>]/V[<диапазон переменных>] /* <символьный аргумент>[<диапазон строк>]

<диапазон переменных> :: =[<простая числовая переменная>,]<простая числовая переменная> /[<простая символьная переменная>,]<простая символьная переменная> /[<метка массива> ,] <метка массива>

I.7.10. PLOT[<устройство> ,] <список PLOT> .

<список PLOT>:: = <элемент PLOT> / <список PLOT>, <элемент PLOT>

<элемент PLOT>:: =[<AB>]<символ <>[<AB>][<AB>],[<параметр PLOT>] <символ >>

<параметр PLOT>:: = <символьное значение> /U/D / R /S / C.

I.7.11. PRINT[<устройство>] / PRINT[<устройство>,][<список PRINT>][<список разделителей>] .

<список PRINT> :: = <элемент PRINT> /<список PRINT> <список разделителей> <элемент PRINT>

<список разделителей> :: = <разделитель> / <список разделителей> <разделитель>

<элемент PRINT> :: = <символьный аргумент> / <числовая переменная>/ <AB>.

I.7.12. PRINT AT[<устройство> ,](<AB>[, <AB>[, <AB>]).

I.7.13. PRINTUSING [<устройство> ,]<формат PU>[<список PRINT>][<список разделителей>]

<формат PU>:: = <символьный аргумент> / <номер строки>.

I.8. Дисковые операторы.

I.8.1. DATA LOAD DC[<логический номер>,] <список переменных> .

I.8.2. DATA LOAD DC OPEN <тип диска>[<устройство> ,] <параметр LOAD OPEN .

<параметр LOAD OPEN :: = <символьный аргумент> / TEMP , <AB>,<AB>.

I.8.3. DATA SAVE DC[K] [<логический номер>,] <параметр SAVE END>

<параметр SAVE END>::= END/ <список аргументов>.

I.8.4. DATA SAVE DC CLOSE[<параметр CLOSE>]

<параметр CLOSE> :: = <логический номер> / ALL

I.8.5. DATA SAVE DC OPEN <тип диска>[K][<устройство>,] <параметр SAVE OPEN>

<параметр SAVE OPEN :: = [<резерв>] <символьный аргумент> / TEMP, <AB>,<AB>

<резерв>:: = <АВ> / <символьный аргумент>.

I.8.6. D BACK S PACE [<логический номер> ,] <параметр BEG>

<параметр BEG> :: = BEG / <АВ>[S].

I.8.7. DSKIP [<логический номер> ,] <параметр END>
<параметр END>:: = END / <АВ>[S].

I.8.8. LIMITS <тип диска> <файл> <числовой элемент>, <числовой элемент>, <числовой элемент>[, <числовой элемент>]
<файл> :: = <ФАЙ>, <символьный аргумент> . /[<логический номер>,] [<символьный аргумент> ,].

I.8.9. LIST DC <тип диска> <параметр LIST DC> .
<параметр LIST DC>:: = [<устройство>] / [<устройство>,] <символьный аргумент>.

I.8.10. LOAD DC <тип диска>[<устройство>,] <символьный аргумент> [<диапазон LOAD>]
<диапазон LOAD> :: = [<номер строки> ,] <номер строки>[,<номер строки>] / <номер строки>[,<номер строки>][,<номер строки>] / [<номер строки>],[<номер строки>], <номер строки>.

I.8.11. MOVE <тип диска>[<устройство> ,] TO <тип диска>[<устройство>]
MOVE <тип диска>[<устройство>,] <символьный аргумент> TO <тип диска> <направление MOVE>
<направление MOVE> :: = <устройство>/ <устройство>, (<резерв>).

I.8.12. MOVE END<тип диска>[<устройство> ,] = <АВ>.

I.8.13. SAVE DC <тип диска>[*] <устройство> ,][<признак>][(<резерв>)] <символьный аргумент>[<диапазон строк>]
<признак> :: = P/T/G.

I.8.14. SCRATCH <тип диска>[<устройство>,] <список имен>
<список имен> :: = <символьный аргумент> / <список имен>, <символьный аргумент>.

I.8.15. SCRATCH DISK <тип диска>[<устройство>][LS = <АВ>,] END = <АВ>.

I.8.16. VERIFY <тип диска> [<устройство>][(<АВ>, <АВ>)]

I.9. Дисковые операторы прямой адресации.

I.9.1. COPY <тип диска>[<устройство>][(<АВ>, <АВ>)] TO <тип диска> [<направление>]

⟨направление⟩ ::= ⟨устройство⟩ / ⟨устройство⟩ ,(⟨AB⟩).

I.9.2. DATA LOAD BA ⟨тип диска⟩[⟨устройство⟩,](⟨адрес сектора⟩) ⟨символьная переменная⟩

⟨адрес сектора⟩ :: = ⟨AB⟩[,⟨переменная⟩] / ⟨символьный аргумент⟩[,⟨переменная⟩].

I.9.3. DATA LOAD DA ⟨тип диска⟩[⟨устройство⟩,] (⟨адрес сектора⟩) ⟨список переменных⟩.

I.9.4. DATA SAVE BA ⟨тип диска⟩[*][⟨устройство⟩,](⟨адрес сектора⟩) ⟨символьная переменная⟩.

I.9.5. DATA SAVE DA ⟨тип диска⟩[*][⟨устройство⟩ ,](⟨адрес сектора⟩) ⟨параметр SAVE END⟩.

I.9.6. LOAD DA ⟨тип диска⟩[⟨устройство⟩,] (⟨адрес сектора⟩) [⟨диапазон LOAD⟩].

I.9.7. SAVE DA ⟨тип диска⟩[*][⟨устройство⟩,][⟨признак⟩](⟨адрес сектора⟩)[⟨диапазон строк⟩].

I.10. Операторы специального назначения.

I.10.1. ASMB ⟨параметр ASMB⟩ .

⟨параметр ASMB⟩ :: =⟨символьная переменная⟩ ,⟨символьная переменная⟩ , ⟨символьная переменная⟩[⟨режим⟩] / ⟨символьная переменная⟩* / , ⟨адрес запуска⟩[,⟨список аргументов⟩]

⟨режим⟩ ::= * /*[⟨диапазон строк⟩]

⟨адрес запуска⟩ :: = ⟨цифра⟩ / ⟨цифра⟩⟨цифра⟩/ ⟨цифра⟩⟨цифра⟩⟨цифра⟩ / ⟨цифра⟩⟨цифра⟩⟨цифра⟩⟨цифра⟩ / ⟨цифра⟩⟨цифра⟩ ⟨цифра⟩ ⟨цифра⟩ ⟨цифра⟩.

I.10.2. LOAD ⟨символьная переменная⟩[⟨диапазон LOAD⟩].

I.10.3. SAVE ⟨символьная переменная⟩[⟨диапазон строк⟩].

I.10.4. *G IO[⟨устройство⟩ ,] (⟨символьный аргумент I⟩ , ⟨символьная переменная⟩)[⟨символьная переменная⟩].

I.10.5.*G IO' ⟨символьный аргумент I⟩[, ⟨переменная⟩].

I.11. Операторы графического взаимодействия.

I.11.1.DDOT ⟨метка символьного массива⟩ , ⟨AB⟩ , ⟨AB⟩.

I.11.2. DDRAW ⟨метка символьного массива⟩ , ⟨AB⟩, ⟨AB⟩.

I.11.3. DNPLOT ⟨метка символьного массива⟩ , ⟨AB⟩, ⟨AB⟩.

I.11.4. DOT ⟨метка символьного массива⟩ , ⟨AB⟩ , ⟨AB⟩.

I.11.5. DRAW ⟨метка символьного массива⟩ , ⟨AB⟩ , ⟨AB⟩.

I.11.6. FRAME ⟨метка символьного массива⟩ , ⟨AB⟩, ⟨AB⟩, ⟨AB⟩,⟨AB⟩.

I.II.7. LABEL <метка символьного массива>[<АВ>][<АВ>],[<АВ>] <список LABEL>

<список LABEL> :: = <символьное значение> / <АВ> / <список LABEL> ,<символьное значение> / <список LABEL>, <АВ>.

I.II.8. NDOT <метка символьного массива> , <АВ> , <АВ>.

I.II.9. NDRAW<метка символьного массива> , <АВ> , <АВ>.

I.II.10. NPLOT <метка символьного массива> , <АВ>, <АВ>.

I.II.II. ORIGIN <метка символьного массива> , <АВ>, <АВ>.

I.II.I2. SCALE <метка символьного массива> , <АВ> , <АВ>.

I.II.I3. STRETCH <метка символьного массива> , <АВ> , <АВ> ,<АВ>, <АВ>.

I.II.I4. TRANSFORM <метка символьного массива> TO <метка цифрового массива>.

I.II.I5. TURN <метка символьного массива> , <АВ> , <АВ> ,<АВ>.

I.II.I6. WINDOW <АВ> , <АВ> , <АВ> , <АВ>.

I.II.I7. * COPY [<ФАУ>,]<метка символьного массива>[, <АВ> , <АВ>].

I.II.I8. * LET <параметр *LET>.

<параметр * LET> :: = <метка символьного массива> =0 /<ГЭ> =0 / <метка символьного массива> = <список* LET>

<ГЭ> :: =<идентификатор переменной> * (<АВ>[,])

<список* LET> :: = <метка символьного массива> / <ГЭ> / <список * LET> + <метка символьного массива> / <список * LET> +<ГЭ>.

I.II.I9. * MOVE <метка символьного массива> , <АВ> , <АВ>.

I.II.20. * OPEN [ ]<список * OPEN> .

<список * OPEN> :: =<метка символьного массива> / <список * OPEN>, <метка символьного массива>.

I.II.2I. * PEN <метка символьного массива>.

I.I2. Операторы работы с КНМЛ.

I.I2.I. BACKSPACE [<устройство>,] <параметр BACK>

<параметр BACK> :: = ВЕВ / <АВ> / <АВ> Р / <АВ> В.

I.I2.2. DATA LOAD [<устройство> ,]<параметр DATA LOAD>

<параметр DATA LOAD> :: = <символьная константа> / <список переменных>.

I.I2.3. DATA RESAVE [<устройство> ,] <тип записи> .

<тип записи> :: = OPEN <символьный аргумент> / END / <список аргументов>.

I.I2.4. DATA SAVE[<устройство>] <тип записи>.

I.I2.5. LOAD[<устройство>][<символьная переменная>][<диапазон LOAD>].

I.I2.6. REWIND [<устройство>].

I.I2.7. SAVE <параметр SAVE>.

<параметр SAVE> :: = <устройство> / <устройство> , [P] [<символьная константа>] [<диапазон строк>].

I.I2.8. SKIP [<устройство> ,] <параметр SKIP>.

<параметр SKIP> :: = END / <AB> / <AB> F / <AB> B.

2. ОТЛИЧИЕ ЯЗЫКА БЭЙСИК 02 ОТ ЯЗЫКА БЭЙСИК 01

2.I. Изменение реализации операторов.

2.I.I. Реализация оператора PRINT обеспечивает вывод в естественной форме значений, находящихся в пределах IE-I2 <X/ <9.999999999999E+I2.

2.I.2. В операторах CLEAR и LOAD исключена индикация ошибки при задании несуществующих строк.

2.I.3. При загрузке программы из символьного аргумента по оператору LOAD допускается использование переменной, не объявленной оператором COM.

2.I.4. Изменены коды, формируемые по оператору KEYIN.

2.I.5. Реализация оператора LINPUT изменена следующим образом:

I) область перемещения курсора ограничена длиной редактируемого поля;

2) введена возможность не высвечивать символ * перед редактируемым текстом;

3) по клавише INSERT производится вставка пробела по текущему положению курсора с исключением последнего символа редактируемого поля;

4) по клавише DELETE производится стирание символа по текущему положению курсора с вставкой пробела на месте последнего символа редактируемого поля;

5) по клавише ERASE производится замена символов в поле редактирования, начиная с текущего положения курсора, на пробелы;

6) по клавише LINE ERASE производится замена символов в поле редактирования на пробелы, курсор устанавливается на первое знакоместо поля редактирования. Символ * заменяется на пробел >.

2.I.6. В операторах : * TRAN , * PACK , * UNPACK, MATSORT , MATSEARCH - исключена конструкция <поле массива>.

2.I.7. В операторе SE/ECT DISK исключена конструкция <длина строки> и указанный адрес устройства заносится в нулевую строку таблицы устройств.

2.I.8. Оператор SE/ECT # заносит указанный адрес устройства в соответствующую строку таблицы устройств, сбрасывая при этом предыдущую информацию в строка.

2.I.9. При указании логического номера (*) в операторах ввода-вывода выбор физического адреса устройства производится из строки таблицы устройств, соответствующей указанному номеру.

2.I.I0. При отсутствии указания устройства в дисковых операторах физический адрес устройства выбирается из нулевой строки таблицы устройств.

2.I.II. Использование оператора ON ERROR в строке непосредственного счета игнорируется.

2.I.I2. В операторе /IST исключена конструкция <номер строки>.

2.I.I3. В дисковых операторах режима адресации секторов может отсутствовать перемещная, в которую заносится очередной номер сектора.

2.2. Реализация дополнительных операторов и расширение функций имеющихся .

2.2.I. Введен оператор REP/ACE , обеспечивающий контекстное изменение символьных переменных.

2.2.2. Введен оператор ASMB, предназначенный для создания программ пользователя на уровне системы инструкций и организации счета по созданной программе.

2.2.3. Операторы DIM и COM дополнены возможностью использования массивов, содержащих 64000 элементов.

2.2.4. Оператор COPY дополнен возможностью использования третьего параметра, указывающего номер начального сектора диска, на который производится запись.

2.2.5. Операторы /OAD, /OAD DC, /OAD DA дополнены возможностью использования номера строки в качестве третьего параметра, с которого необходимо начать счет по программе.

2.2.6. Оператор MATSEARCH дополнен возможностью использования операции отождествления и диапазона.

2.2.7. Оператор ROTATE дополнен возможностью циклического сдвига вправо битов каждого байта символьной переменной.

2.2.8. Функция VAL дополнена возможностью преобразования двух байтов символьного аргумента в числовое значение.

2.2.9. Оператор BIN дополнен возможностью преобразования целой части арифметического выражения в двоичное число, записываемое в первые два байта символьной переменной.

2.2.10. Оператор RETURN CLEAR дополнен возможностью использования параметра ALL, обеспечивающего сброс всего магазина возвратов из цикла и подпрограмм.

2.2.11. Оператор LIMITS в форме с именем файла дополнен возможностью физического адреса устройства и четвертой числовой переменной, в которую заносится состояние файла.

2.2.12. Оператор STOP дополнен возможностью индикации номера программной строки, содержащей оператор STOP.

2.2.13. Оператор IF THEN дополнен возможностью использования логических выражений, содержащих логические операции AND, OR, XOR.

2.2.14. Оператор RESTORE дополнен возможностью указания номера строки программы, содержащей оператор DATA.

2.2.15. Оператор LIST дополнен возможностью вывода программных строк, содержащих указанный контекст, вывода таблицы номеров файлов и указания параметра S.

2.2.16. Оператор MOVE дополнен возможностью указания второго адреса устройства и возможностью переписывать файлы по одному с одного диска на другой.

2.2.17. Оператор HEXPRINT дополнен возможностью использования символьной константы.

2.2.18. Функции NUM, LEN, POS и оператор INIT дополнены возможностью использования конструкций: <метка символьного массива> и <символьная константа>.

2.2.19. Операторы ADD, AND, BOOL, OR, XOR дополнены возможностью использования конструкций: в первом элементе <метка символьного массива>, во втором элементе <метка символьного массива> и <символьная константа>.

2.2.20. Оператор DATA дополнен возможностью использования конструкций в элемент DATA : <числовой элемент> и <символьный операнд>.

2.2.21. Оператор PRINTUSING дополнен возможностью использования конструкции в формате <метка символьного массива>.

2.2.22. Операторы DATA LOAD DC CREM, DATA SAVE DC CREM дополнены возможностью использования конструкции <DAD>, при этом указанный адрес заносится в нулевую строку таблицы устройств.

2.2.23. Оператор ON ERROR дополнен возможностью указания в качестве параметра перехода GOSUB и THEN.

2.2.24. Добавлена возможность объединения программных строк в одну строку в процессе редактирования.

2.2.25. Операторы UNPACK и # UNPACK дополнены возможностью проверки распаковываемых тетрад на наличие цифр.

2.2.26. Операторы LINPUT и INPUT дополнены возможностью останова программы по клавише HALT/STEP после выполнения этих операторов.

2.2.27. В режиме TRACE добавлена возможность вывода служебных символов в виде точек и вывода сообщений при выполнении оператора NEXT.

Примечание. Кроме отличия языка БЭЙСИК 02 от языка БЭЙСИК 01 необходимо учитывать следующие особенности БЭЙСИК-системы:

1. Изменена индикация готовности системы и сбойных ситуаций в системе;

2. Программы, оттранслированные и записанные на носитель по версии системы БЭЙСИК 01, не могут использоваться при работе с версией системы БЭЙСИК 02 (все оттранслированные программы необходимо перетранслировать).

3. СПИСОК КОДОВ ОШИБОК

Таблица

Ошибка	Причина возникновения ошибки
Сбой системы	Переход через нулевой адрес управляющей памяти
Сбой МО	Сбой матобеспечения
Сбой ОЗУ	Сбой оперативной памяти
Сбой УП	Сбой управляющей памяти
ERR 01	Переполнение памяти текстом программы
ERR 02	Переполнение памяти значениями переменных
ERR 03	Деление на нуль
ERR 04	Строку нельзя выполнить из-за противоречия в переменных
ERR 05	-
ERR 06	Синтаксическая ошибка при вводе строки с клавиатуры
ERR 07	Синтаксическая ошибка при вводе строки с диска
ERR 08	Функция FN не определена
ERR 09	Несоответствие параметров DEFFN' и GOSUB'
ERR 10	Синтаксическая ошибка при вводе строки из переменной

Продолжение табл.

Ошибка	Причина возникновения ошибки
ERR 11	Несуществующий номер строки
ERR 12	Оператор с синтаксической ошибкой
ERR 13	В строке отсутствует DATA для RESTORE
ERR 14	Ошибка в трансляции при выполнении оператора ASMB
	Неправильная инструкция 01
	Неверно задан тип константы 02
	Метка описана дважды 04
	Ошибка в задании константы 05
	Использован неописанный идентификатор 06
	Переполнение ОЗУ или УП 07
	Использование константы 'К' больше 15 08
	Константа или номер ячейки больше 255 09
	Недопустимый адрес безусловного перехода 10
	Недопустимый адрес условного перехода 11
	Константа больше 177777 (восьмеричное) 12
	Автокодная программа не заканчивается END 17
	Объем выделенной рабочей области недостаточен 20
	Для трансляции программы объем выделенной области под объектную программу недостаточен 21
ERR 15	Неправильный номер строки или диапазон строк
ERR 16	Нет ассемблера
ERR 17	-
ERR 18	Недопустимая величина (числа, размерности, индекса)
ERR 19	-
ERR 20	Недопустимый формат числа (порядок больше 99)
ERR 21	-
ERR 22	Программу нельзя выполнить из-за противоречия в переменных
ERR 23	Отсутствие программы в памяти
ERR 24	Операторы DATA не объединены в список
ERR 25	Недопустимое использование операторов GOSUB/RETURN, FOR/NEXT
ERR 26	-
ERR 27	Данные исчерпаны или находятся за допустимыми пределами

Продолжение табл.

Ошибка	Причина возникновения ошибки
ERR 28	-
ERR 29	Недопустимый формат данных в операторе INPUT
ERR 30	-
ERR 31	Новый номер строки по оператору RENUMBER больше 9999
ERR 32	Длина выгружаемой программы больше длины переменной
ERR 33	-
ERR 34	-
ERR 35	-
ERR 36	Ошибочный формат в операторах CONVERT, PACK, UNPACK
ERR 37	Для оператора PRINTUSING нет оператора IMAGE
ERR 38	-
ERR 39	-
ERR 40	Нарушена возрастающая последовательность идентификаторов переменных в операторе LIST V
ERR 41	Недопустимый аргумент STR
ERR 42	-
ERR 43	Недопустимое присвоение
ERR 44	Программа защищена
ERR 45	Длина транслируемого текста больше 255 байтов
ERR 46	Новый начальный номер строки в операторе RENUMBER мал
ERR 47	Физический адрес устройства больше 7F (шестнадцатеричное)
ERR 48	Отсутствует оператор DATA с указанным номером
ERR 49	Неквадратная матрица
ERR 50	Неправильные параметры матриц
ERR 51	Неверный матричный аргумент в операторах умножения и транспонирования матриц
ERR 52	Инвертирование сингулярной матрицы
ERR 53	Длина распаковываемого массива мала
ERR 54	Ошибочная длина строки устройства в операторе SELECT
ERR 55	Ошибочный адрес сектора в операторах COPY, VERIFY
ERR 56	Число превышает формат в операторах PACK, CONVERT
ERR 57	-
ERR 58	Переменная в операторе LOAD DA несфор
ERR 59	Длина переменной мала
ERR 60	Файл не открыт или не существует

Продолжение табл.

Ошибка	Причина возникновения ошибки
ERR 61	-
ERR 62	-
ERR 63	-
ERR 64	-
ERR 65	-
ERR 66	Сбой по к.с. при передаче из БМЭ в ПИД
ERR 67	Неправильный тип записи программа-данные
ERR 68	Буфер ввода-вывода мал
ERR 69	Переменная/ массив не помещается в буфер ввода-вывода
ERR 70	В операторе PRINTUSING нет формата
ERR 71	Файл уже есть в каталоге
ERR 72	Файл вычеркнут
ERR 73	Файл отсутствует в каталоге
ERR 74	Адрес сектора вне файла
ERR 75	Конец каталога
ERR 76	Файл заполнен
ERR 77	Временный файл находится не за областью каталога
ERR 78	Указатель каталога заполнен
ERR 79	-
ERR 80	Ошибка ввода-вывода
	Авария ВВУ (диск, лента) 01
	Нарушение последовательности процедур 02
	УВВ не готово к обмену 03
	Резерв 04
	Защита от записи 05
	Маркер начала (конца) ленты 06
	Ошибка по контрольной сумме контрольного чтения 07
	После записи
	Авария БМЭ 08
	Резерв 09
	Ошибка по контрольной сумме при считывании 0A
	Ошибка процедуры (кодирование, формат, контрольная сумма обмена ПИД и БМЭ) 0B
	Сектор с указанным адресом не найден 0C
	Нарушение последовательности интерфейсных 0D

Ошибка	Причина возникновения ошибки
	команд в процедуре
	Недопустимый адрес сектора — OE
	Ошибочная длина массива — OF

4. АДРЕСА УСТРОЙСТВ ВВОДА-ВЫВОДА

ФАУ десятичный	ФАУ шестнадцатиричный	ТИП УВВ
I	0I	Устройство клавишное
5	05	БОСГИ(символьный)
I2	0C	Печать
I3	0D	Печать
I4	0E	Печать
I6	I0	БОСГИ (графический)
20	I4	Графопостроитель "Н-306"
24	I8	НГМД
27	IB	Накопитель на магнитной ленте
28	IC	Накопитель на магнитном диске
32	20	Аналого-цифровой преобразователь
33	2I	"_"_"
34	22	"_"_"
35	23	"_"_"
36	24	Цифро-аналоговый преобразователь
37	25	"_"_"
38	26	"_"_"
39	27	Указатель графической информации
40	28	Фотосчитыватель,перфоратор
45	2D	КРТР
46	2E	Приборный интерфейс
52	34	Передатчик по рангу С2
53	35	Приемник по рангу С2
54	36	Передатчик по рангу КРПС
55	37	Приемник по рангу КРПС

5. АЛФАВИТНЫЙ УКАЗАТЕЛЬ СЛУЖЕБНЫХ СЛОВ ЯЗЫКА БЭЙСИК 02

6. Операторы работы с КНМД.

Информация (программы и данные) на магнитной ленте фиксируется в виде физических записей. Длина физической записи определяется заданной длиной строки устройства.

Элемент информации (под элементом подразумевается символьные и числовые переменные и константы) не может быть расположен сразу в двух физических записях. Если он не помещается целиком в предназначенной для него части физической записи, то полностью переносится в следующую физическую запись. Если длина элемента информации превышает заданную длину физической записи, то выдается сообщение об ошибке.

Запись программы на магнитную ленту не ограничивается записью очередных программных строк. Перед программой автоматически записывается заголовок (открывающая запись), а после нее - закрывающая запись. Заголовок, строки программы и закрывающая запись образуют программный файл.

Заголовок программы (заголовок файла) - это физическая запись, которая может включать в себя название программы.

Если на магнитной ленте записано несколько программ по именам, то их удобно загружать в память машины по присвоенным им именам.

Каждая физическая запись программного файла содержит столько строк программы, расположенных непосредственно друг за другом, сколько позволяет заданная длина физической записи. Очередная строка, непомещающаяся в блоке, переносится в следующую физическую запись.

Закрывающая запись - это также физическая запись.

Программный файл формируется одним оператором вывода программы и содержит только одну логическую запись, которая в свою очередь, состоит из одной или нескольких физических записей.

На магнитной ленте можно пропустить требуемое количество программных файлов (логических записей) и физических записей.

Данные хранятся на магнитной ленте в виде физических записей. Если данные не вмещаются в одну физическую запись, то они занимают две физические записи и т.д.

Данные, выведенные на магнитную ленту одним оператором вывода, образуют логическую запись (кроме оператора DATA SAVE LT).

Подобно программным файлам составляются файлы данных, которые состоят из логических записей, впереди которых помещается заголовок (имя)

файла, а в конце - закрывающая запись файла.

Заголовок и закрывающая запись файлов данных сходны с соответствующими записями программных файлов, но в отличие от программных файлов, в файле данных заголовок и закрывающая запись формируются пользователем с помощью соответствующих операторов.

Имя файла занимает определенное количество байтов, но оно не должно превышать заданную длину физической записи.

Каждая физическая запись файла данных должна содержать столько данных, расположенных непосредственно друг за другом, сколько позволяет заданная длина физической записи. Очередное значение, не помещающееся в записи, переносится в следующую физическую запись.

Организация данных в виде файлов дает возможность устанавливать магнитную ленту в начало или конец файла при пропуске файлов или возврат к ним. Наличие заголовка и закрывающей записи способствует правильной установке магнитной ленты в начало или конец файла.

Машина обеспечивает возможность обновления логических записей файла данных.

6.I. BACKSPACE .

BACKSPACE [<устройство> ,] <параметр BACK>

<параметр BACK> ::= BEG / <AB> / <AB> F / <AB> B

Оператор BACKSPACE предназначен для перемотки магнитной ленты назад. В зависимости от параметров, включенных в оператор BACKSPACE, осуществляется следующая перемотка магнитной ленты назад:

BEG - назад к началу текущего файла и останов после заголовка файла; <AB> - на *N* логических записей в пределах одного файла данных и останов перед *N*+I логической записью;

<AB> F - на *N* файлов и останов перед *N*+I файлом;

<AB> B - на *N* физических записей и останов перед *N*+I физической записью; где *N* - целая часть арифметического выражения, должна быть >=I

Примеры правильного синтаксиса:

1) :20 BACKSPACE BEG
2) :35 BACKSPACE # 3, LOG 5.8
3) :II0 BACKSPACE/09, (X*Y)F
4) :320 BACKSPACE I88

6.2. DATA LOAD.

DATA LOAD [<устройство> ,]<параметр DATA LOAD>
<параметр DATA LOAD>::= <символьная константа> / <список переменных> оператор DATA LOAD предназначен для считывания с магнитной ленты логических записей и присвоения значений переменным и массивам из списка переменных.

Массивы заполняются строка за строкой. Если значения получают не все переменные из списка, то считывается следующая логическая запись. Записанные, но не используемые оператором DATA LOAD данные игнорируются. Переменные из списка переменных, для которых не хватило данных, сохранят прежние значения.

Параметр <символьная константа> определяет имя файла данных и устанавливает магнитную ленту на первую логическую запись файла с данным именем.

Примеры правильного синтаксиса;

1) :40 DATA LOAD "RW"
2) :50 DATA LOAD A,W, C %(10)
3) :10 DATA LOAD #1, A, B %(), C%
4) :60 DATA LOAD/08, B, X1, STR(A%,3,5)

6.3. DATA RESAVE .

DATA RESAVE [<устройство> ,] <тип записи>
<тип записи> ::= OPEN<символьный аргумент> / END/<список аргументов> оператор DATA RESAVE предназначен для обновления логических записей на магнитной ленте. Могут быть изменены все записи файла, включая его имя.

Параметр OPEN предназначен для замены старого заголовка файла новым. Параметр END предназначен для формирования новой закрывающей записи файла.

Для изменения содержимого одной логической записи необходимо:

Установить магнитную ленту на начало файла (оператором DATA LOAD);

Установить магнитную ленту перед обновляемой записью (операторами SKIP или BACKSPACE);

Занести новую запись на место старой (оператором DATA RESAVE).

Число физических записей в старой логической записи и в новой записи должно быть одинаковым. Это выполняется при условии идентичности списка аргументов в операторе DATA RESAVE списку аргументов в старой записи, то есть они не должны различаться по числу,

виду переменных и их последовательности.

Для изменения записи используется не только оператор DATA RESAVE, но и оператор DATA SAVE. Но этот оператор не гарантирует того, что новый файл запишется на нужное место: на магнитной ленте может остаться информация из старой записи.

Примеры правильного синтаксиса:

1) :100 DATA RESAVE #1, OPEN "FILE"
2) :150 DATA RESAVE/08, A, B%, C ()

6.4. DATA SAVE.

DATA SAVE [<устройство> ,] <тип записи>.

Оператор DATA SAVE предназначен для записи на магнитную ленту данных в виде логической записи, либо в виде файла данных.

Параметр OPEN предназначен для записи заголовка в начале файла данных. Параметр END предназначен для формирования закрывающей записи в конце файла.

При записи на одну магнитную ленту нескольких массивов, в начале каждого массива записывают его заголовок, а в конце формируют закрывающую запись. Пользователь может найти каждый файл по имени, а по закрывающей записи проверить, подошла ли магнитная лента к концу файла, используя оператор IF END THEN. Кроме того, заголовок и закрывающая запись позволяют пропустить несколько файлов при перемотке магнитной ленты вперед или назад.

Но массивы данных могут и не иметь заголовка и закрывающей записи, то есть параметры END и OPEN не обязательные.

Запись данных на магнитную ленту выполняется с помощью одного или нескольких операторов DATA SAVE.

Примеры правильного синтаксиса:

1) :20 DATA SAVE A, B, C% ()
2) :100 DATA SAVE #2, END
3) :300 DATA SAVE OPEN "PROGRAM I"

6.5. LOAD .

LOAD [<устройство> ,][<символьная переменная>][<диапазон LOAD>]

оператор LOAD предназначен для считывания программы или ее сегмента с магнитной ленты и их загрузки в память машины.

При использовании оператора CHAIN при счете по программе автоматически выполняются следующие действия:

1) приостанавливается исполнение текущей программы;

2) из памяти машины стираются все строки текущей программы или ее сегмента, ограниченного значениями параметров <номер строки> ;

3) стираются значения всех переменных, кроме общих, объявленных в операторе COM;

4) ранее записанная на магнитную ленту программа загружается в память машины в соответствии с параметрами оператора CHAIN ;

5) продолжается исполнение программы.

Реализация оператора CHAIN в непосредственном счете отличается от его реализации при счете по программе тем, что выполняются действия по пункту 2) и 4), а оператор CHAIN без параметров лишь загружает программу в память машины.

В зависимости от наличия параметров <номер строки> осуществляется стирание строк программы в памяти машины по следующим правилам:

1) если заданы оба параметра <номер строки> , то стирается часть программы, ограниченная этими номерами строк; счет начинается со строки с номером, равным значению первого параметра <номер строки> ;

2) если задан только первый параметр <номер строки> , то стирается часть программы, начиная с заданного номера строки и до конца; счет начинается с заданного номера строки;

3) если задан только второй параметр <номер строки> , то стирается часть программы от ее начала до заданного номера строки; счет начинается с первого оператора загружаемой программы;

4) если параметры <номер строки> не заданы, то стирается вся программа; счет начинается с первого оператора загружаемой программы;

5) при отсутствии в загружаемой программе номера строки, указанного в первом параметре <номер строки> , машина выдает сообщение об ошибке, если оператор использован при счете по программе.

Оператор CHAIN при использовании его при счете по программе позволяет организовать пользователю автоматическое прохождение сегментированных программ. При этом, для передачи значений переменных из одного сегмента программы в другой, эти переменные объявляют как общие.

Оператор CHAIN должен быть последним исполняемым оператором

в строке. Примеры правильного синтаксиса:

1) : LOAD "PROL"
2) : 100 LOAD/08, "WK", 500, 1000
3) :150 LOAD #1,, 300

6.6. REWIND.

REWIND [<устройство>]

Оператор REWIND предназначен для перемотки магнитной ленты и установки ее в начало.

Примеры правильного синтаксиса:

1) : REWIND
2) : 50 REWIND/09
3) :100 REWIND #2

6.7 SAVE.

SAVE <параметр SAVE>.

<Параметр SAVE> ::= <устройство> /<устройство> ,[P] [<символьная константа>][<диапазон строк>]

оператор SAVE предназначен для записи на магнитную ленту программы или ее сегментов.

В зависимости от параметров, включенных в оператор SAVE, осуществляется запись программы на магнитную ленту по следующим правилам:

<символьная константа> - записывается программа с именем;

Если задан только первый параметр <номер строки>, то записывается часть программы, начиная с заданного номера строки и до конца;

если задан только второй параметр <номер строки>, то записывается часть программы от ее начала до заданного номера строки;

если заданы оба параметра <номер строки>, то записывается часть программы, ограниченная заданными номерами строк;

если параметры <номер строки> не определены, то записывается вся программа;

P - записывается программа, которая будет защищена.

Программа, записанная на магнитную ленту по оператору SAVE P, и загруженная в память машины по оператору LOAD не может быть просмотрена или скопирована. Если защищенная программа загружена в память машины, то остальные загружаемые программы можно просматривать

или скопировать только, если перед их вводом в память машины очистить ее, используя кнопку SR, или оператор CLEAR .

Примеры правильного синтаксиса:

1) : SAVE
2) : 50 SAVE/ 08, P "UTILIT" 400,800
3) : 100 SAVE, 500

6.8. SKIP .

SKIP [<устройство> ,] <параметр SKIP>

<параметр SKIP> ::= END/ <AB> / <AB> F / <AB> B

Оператор SKIP предназначен для перемотки магнитной ленты вперед. В зависимости от параметров, включенных в оператор SKIP, осуществляется следующая перемотка магнитной ленты вперед:

END - до конца текущего файла и останов перед записью, закрывающей файл; <AB> - на N логических записей в пределах одного файла и останов перед N+1 логической записью;

<AB> F - на N файлов и останов перед N+1 файлом;

<AB> B - на N физических записей и останов перед N+1 физической записью; где N - целая часть арифметического выражения, должна быть >=1.

Примеры правильного синтаксиса:

1) :20 SKIP #1, END
2) :100 SKIP/08, 2B
3) :200 SKIP (A^2+B/3)F

СОДЕРЖАНИЕ