НПП «НАУЧНЫЙ ЦЕНТР»

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДВК

КНИГА 3

ЯЗЫКИ ПРОГРАММИРОВАНИЯ.
АССЕМБЛЕР, ПАСКАЛЬ, БЕЙСИК

МОСКВА 1990 год

АННОТАЦИЯ

Данная книга содержит сведения о машинно-ориентированном языке АССЕМБЛЕР, о языках высокого уровня ПАСКАЛЬ и БЕЙСИК.

В главе «Ассемблер. Описание языка» приведены общие сведения о структуре и элементах языка, представлениях данных, командах и директивах АССЕМБЛЕРа, а также описание макросредств языка АССЕМБЛЕР.

Глава «Ассемблер. Руководство программиста» описывает транслятор языка, предназначенный для перевода программ с языка АССЕМБЛЕР в программу на машинном языке с информацией для программы редактор связей. Также в эту главу включены сведения об основных характеристиках транслятора и структуре входных и выходных данных.

В главе «Редактор связей» содержится информация о возможностях редактора связей и способе его использования.

Глава «Библиотекарь. Руководство оператора» представляет собой руководство пользователя ОС ФОДОС-2 по работе с библиотекарем. В главе дано описание структуры библиотеки, функциях и способе использования библиотекаря.

Глава «Отладчик. Виртуальный отладчик» содержит информацию о возможностях отладчика и способе его использования.

Две следующие главы описывают язык высокого уровня ПАСКАЛЬ, созданный профессором Н. Виртом. Дано подробное описание языка, его элементов, функций и процедур. Описание сопровождается примерами и содержит всю информацию, необходимую программисту для работы с языком ПАСКАЛЬ в ОС ФОДОС-2.

В главе «Бейсик. Описание языка» приведено описание самого языка, правила составления и выполнения программ на языке БЕЙСИК, описаны основные элементы языка БЕЙСИК, представлены операторы, функции, команды для отладки и запуска программных файлов, описано использование некоторых ключевых команд.

Глава «Бейсик. Руководство программиста» содержит общие сведения о назначении и запуске интерпретатора языка БЕЙСИК, описаны дополнительные функции и показаны возможности включения программ, написанных на языке АССЕМБЛЕР.

АССЕМБЛЕР. ОПИСАНИЕ ЯЗЫКА.

1. ОБЩИЕ СВЕДЕНИЯ

Язык АССЕМБЛЕР предназначен для автоматизации программирования на уровне машинных команд.

Основные характеристики языка АССЕМБЛЕР:

— легкий для использования формат предложений исходной программы;
— символическая адресация элементов программы;
— разнообразные способы представления данных;
— наличие средств деления программы на программные секции;
— наличие макросредств;
— наличие средств трансляции по условию;
— возможность включать в программу имеющиеся программные запросы системной макробиблиотеки.

2. СПОСОБ ОПИСАНИЯ ЯЗЫКА

В данном документе используются следующие соглашения:

— элемент или группа элементов, заключенные в квадратные скобки ([ ]), являются не обязательными;
— элемент или группа элементов, предшествующие многоточию (...), могут быть повторены;
— элементы из латинских букв, являющиеся ключевыми словами (в форматах предложений они подчеркнуты), записываются точно так же, как они записаны в формате предложения;
— элементы, следующие за ключевым словом, заменяются согласно описанию.

В данном документе используются следующие обозначения:

<= меньше или равно

=> больше или равно

3. СТРУКТУРА ЯЗЫКА

Выполняемая программа на языке АССЕМБЛЕР состоит из одного или нескольких программных модулей. Максимальное допустимое число программных модулей — шесть.

Каждый программный модуль состоит из последовательности строк — предложений и/или комментариев.

Предложения подразделяются на четыре основных вида: команды, директивы, макрокоманды и оператор прямого присваивания.

Каждой команде АССЕМБЛЕРа в оттранслированной программе соответствует одна команда в машинном коде.

Директивы используются для управления процессом трансляции и выполнения различных функций: управление печатью листинга, управление распределением памяти, секционирование и объединение программ, резервирование областей памяти, запись данных и т. д. Директивы не порождают команд в машинном коде.

С помощью макрокоманд вызываются макроопределения, написанные на языке АССЕМБЛЕР, модифицируются в соответствии с информацией, заданной в каждой отдельной макрокоманде, и включаются в исходную программу, заменяя собой макрокоманды.

Синтаксическими элементами предложений являются символические имена и операции. Символические имена используются для обозначения команды, макрокоманды, директивы, адреса, регистра и метки. Операции определяют действия над данными. Константы, арифметические и логические операции, а также специальные операции могут быть связаны в выражения. Выражение служит для задания правил вычисления значения: это значение получается в результате выполнения указанных в выражении операций над данными.

АССЕМБЛЕР позволяет разбивать исходную программу на несколько программных секций с помощью директив секционирования и управлять распределением памяти для программы во время ее связывания.

Полученные после трансляции программы могут быть абсолютными или перемещаемыми. Перемещаемые программы могут быть выполнены в любой области оперативной памяти.

АССЕМБЛЕР позволяет записывать одну программную секцию в разных программных модулях и во время трансляции объединять части программной секции в одну секцию.

3.1. Строки

Исходная программа на языке АССЕМБЛЕР состоит из последовательности строк.

Формат строки:

[метка:] операция операнд(ы) [;комментарий] <ВК>

```
          :                              :
          :<-----Предложение----->:
```

или

;комментарий <ВК>

Метка и комментарий являются необязательными компонентами предложения.

Содержимое полей операции и операндов взаимосвязаны. Любое из них может быть опущено в зависимости от содержимого другого.

АССЕМБЛЕР обрабатывает предложения последовательно одно за другим, формируя одно-, двух- или трехсловную команду или слово данных, или управляет процессом трансляции, если предложение является директивой.

Предложение должно содержать информацию в одном или нескольких из указанных выше полей или во всех четырех полях.

Разрешаются и пустые строки (пустое предложение).

Предложения могут иметь в поле операндов один операнд или два операнда, например,

```
CLR R0
MOV #3044,R2
```

Каждое предложение АССЕМБЛЕРа должно быть размещено на одной строке. Продолжение предложения на следующую строку не разрешается.

Строка может содержать до 132(10) символов, не считая символа ВК. Все символы, вводимые сверх этого ограничения, игнорируются, и печатается сообщение об ошибке.

Использование символа ГТ в каждом поле позволяет записать текст исходной программы в формате, удобном для чтения, например:

Поле метки	Поле операции	Поле операнда(ов)	Поле комментария
CHECK:	BIT	#1,R0	; число нечетное?
	BEQ	EVEN	; нет, четное, на выход
	MOV	#—1,ODDFLG	; да, установить признак
EVEN:	RTS	PC	; выход из подпрограммы

3.1.1. Метка

Метка — это определяемое пользователем символическое имя, которому при трансляции присваивается текущее значение счетчика адреса. Значение метки может быть абсолютным или перемещаемым в зависимости от значения счетчика адреса. Метка и соответствующее значение счетчика адреса запоминаются в таблице имен пользователя.

Меткой помечаются строки в программе, к которым осуществляется обращение. Метка в строке должна стоять первой и ограничиваться символом «:» или «: :».

Например, если текущее значение счетчика адреса равно 100, в предложении ABCD: MOV A,B метке ABCD будет присвоено значение 100.

Если в этом примере значение счетчика адреса перемещаемое, окончательное значение ABCD должно быть k+100, где k — адрес начала перемещаемой секции, в которой определена метка ABCD.

В поле метки одного предложения может быть несколько меток, каждой из них присваивается одно и то же значение счетчика адреса. Например, если текущее значение счетчика адреса равно 100, то в предложении

ABC: ABD: MASK: MOV A,B меткам ABC, ABD, MASK будет во время трансляции присвоено значение 100.

Если у двух или более меток совпадают первые шесть символов, то при трансляции в листинге печатается сообщение об ошибке «M».

3.1.2. Операция

Поле операции в предложении следует за полем метки и может содержать имя команды, имя макрокоманды или директиву АССЕМБЛЕРа.

Если в поле операции помещено символическое имя команды, АССЕМБЛЕР транслирует его в машинный код команды.

Если в поле операции помещено имя макрокоманды, АССЕМБЛЕР при трансляции заменяет макрокоманду соответствующим расширением, которое пользователь определил в начале программы через макроопределение.

Если в поле операции записана директива АССЕМБЛЕРа, она обозначает определенную функцию или действие, которое должно быть выполнено во время трансляции.

Поле операции может быть ограничено пробелом, символом ГТ или любым не буквенно-цифровым символом, который не используется для записи символических имен, например:

MOV A,B — операция MOV ограничена символом ГТ

MOV#A,B — операция MOV ограничена символом #

Если в предложении нет операндов или комментария, последним вводится символ ВК.

Если поле операции остается пустым, предложение транслируется как директива АССЕМБЛЕРа .WORD без операндов.

3.1.3. Операнд

Поле операнда в предложении следует за полем операции и может содержать один или более операндов, над которыми должны выполняться действия, определенные операцией.

Операнды могут быть представлены в виде выражений, чисел и параметров в зависимости от того, что записано в по-

ле операции: символическая команда, макрокоманда, системная макрокоманда или директива.

Если предложение содержит несколько операндов, они отделяются друг от друга запятой, пробелом, символом ГТ или символами «<» и «>», в которые заключаются один или несколько операндов.

Операнду может предшествовать операция, метка или другой операнд, после него может следовать комментарий.

Поле операнда заканчивается точкой с запятой, если за ним следует комментарий, или символом ВК, если нет комментария, например:

LABEL: MOV A,B <ВК>

Пробел между символической командой MOV и операндом А ограничивает поле операции и начинает поле операнда, запятая разделяет операнды А и В.

3.1.4. Комментарий

Наличие комментария в предложении необязательно. В поле комментария могут использоваться любые символы терминала за исключением символов ПУС, ЗБ, ВК, ПС, ВТ, ПФ.

Полю комментария могут предшествовать все три описанных выше поля, любые из них или ни одного. Поле комментария должно начинаться с символа «;» и заканчиваться символом ВК. При продолжении комментария на следующую строку каждая строка должна начинаться с символа «;».

Комментарии не оказывают никакого влияния на процесс трансляции и на выполнение программы, но они полезны при распечатке программы для последующего анализа, отладки и документирования.

4. ЭЛЕМЕНТЫ ЯЗЫКА

4.1. Алфавит языка

При записи программы используются только символы, входящие в алфавит языка АССЕМБЛЕРа. Алфавит делится на три группы символов: цифры, буквы и специальные символы.

4.1.1. Цифры. Цифра — это один из восьми символов: 0, 1, 2, 3, 4, 5, 6, 7. Десятичная цифра — это один из десяти символов: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Если не оговорено, то последовательность цифр интерпретируется АССЕМБЛЕРом как число в восьмеричной системе счисления.

4.1.2. Буквы. Буква — это один из двадцати шести символов: 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.

4.1.3. Буквенно-цифровые символы. Буквенно-цифровой символ — это либо буква, либо цифра.

4.1.4. Специальные символы. Специальный символ — это один из символов, приведенных в табл. 1 и в приложении 1.

Таблица 1

Символ	Название
1	2
	Пробел
:	Двоеточие
=	Равно
%	Процент
#	Признак числа
@	Коммерческое ЭТ
—	Минус
(	Круглая скобка левая
)	Круглая скобка правая
.	Точка
,	Запятая
;	Точка с запятой
<	Меньше
>	Больше
+	Плюс
*	Звездочка
/	Дробная черта
&	Коммерческое И
!	Восклицательный знак
"	Кавычки
'	Апостроф
^	Стрелка вверх
\	Обратная дробная черта

4.1.4.1. Разделительные и ограничительные символы

В табл. 2 приведены допустимые разделительные и ограничительные символы.

Таблица 2

Символ	Определение	Назначение
1	2	3
Пробелы	Один или более пробелов и/или символов ГТ	Допустимый разделитель для операндов. Пробелы внутри выражения игнорируются
	Запятая	Допустимый разделитель для выражений и операндов

1	2	3
<...>	Парные символы «<» и «>»	Используются для выделения аргумента, в частности, если в аргументе уже использованы разделительные символы. Символы «<» и «>» могут использоваться в любом месте программы для ограничения выражения, рассматриваемого как терм
Λ\...\	Конструкция с символом Λ	Эта конструкция эквивалентна символам «<» и «>» и обычно используется там, где аргумент уже содержит эти символы

Параметры макрокоманд могут быть представлены в различной форме в зависимости от их использования.

При составлении программ необходимо соблюдать следующие правила:

1) если параметры не содержат разделительные символы, то они отделяются запятыми;

2) если параметры содержат разделительные символы или пробелы, то они могут быть заключены в символы «<» и «>». Внешняя пара символов «<» и «>» аннулируется при использовании параметра;

3) если параметр содержит разделительные символы, включая символы «<» и «>», то параметр можно записать, используя конструкцию вида Λ\...\, где в качестве ограничителя «\» может быть использован любой символ. Символ «Λ» и ограничители при использовании параметра опускаются. Следует иметь ввиду, что независимо от способа записи каждого из параметров, друг от друга параметры необходимо отделять запятыми.

4.1.4.2. Символы управления форматом

Управление форматом текста программы по горизонтали осуществляется с помощью символов пробел и ГТ. Эти символы не влияют на процесс трансляции, если они не являются составной частью имени, числа, аргумента директивы .ASCII, или, если эти символы не используются в качестве ограничителя операции.

Предложение может быть записано следующим образом:

```
LABEL:MOV(SP)+,TAG;выборка данного из стека
```

или, используя символы управления форматом:

```
LABEL: MOV (SP)+,TAG  ; выборка данного из стека
```

Использование символов пробел и ГТ в предложениях позволяет сделать программу более удобной для чтения.

Управление форматом листинга программы по вертикали, т. е. размером страницы в N строк, осуществляется вводом кода символа ПФ после N-ой строки. Если код ПФ отсутствует, страница автоматически заканчивается через каждые 58 строк.

4.1.4.3. Символы операций

В табл. 3 приведены символы одноместных операций (операций с одним членом).

Таблица 3

Символ	Операция	Пример записи
1	2	3
+	Плюс	+A положительное значение, эквивалентно A
—	Минус	—A отрицательное значение A, двоичное дополнение
∧	Указатель специальной операции	∧F3.0 интерпретируется как однословное число с плавающей запятой 3.0
		∧C24 интерпретируется как обратный код числа 24
		∧D127 интерпретируется как десятичное число 127(10)
		∧O34 интерпретируется как восьмеричное число 34
		∧B11000111 интерпретируется как двоичное 11000111

Символы одноместных операций могут быть объединены друг с другом в одном терме, например:

∧C∧O12
—∧O5

В табл. 4 приведены символы двухместных операций (операций с двумя членами).

Таблица 4

Символ	Операция	Пример записи
1	2	3
+	Сложение	A+B
—	Вычитание	A—B

1	2	3
*	Умножение	A*B (результат — 16-разрядное произведение)
/	Деление	A/B (результат — 16-разрядное частное)
&	Логическое И	A&B
!	Логическое ИЛИ	A!B

Все двухместные операции имеют одинаковый приоритет. Деление и умножение выполняются над числами со знаком.

Термы в выражении могут быть сгруппированы и заключены в символы «<» и «>».

Термы, заключенные в символы «<» и «>», вычисляются первыми, остальные операции выполняются последовательно слева направо, например:

```
.WORD 1+2*3      ; 11 восьмеричное
.WORD 1+<2*3>    ; 7 восьмеричное
```

4.1.5. Недопустимые символы

Символы считаются недопустимыми в одном из следующих случаев:

1) символ, который не принадлежит набору символов АССЕМБЛЕРа, всегда является недопустимым и вызывает печать символа «?» В соответствующей позиции текущей строки и печать сообщения об ошибке «1» в листинге, например:

```
LABEL ю: MOV A,B
```

Так как буква «ю» не входит в набор символов АССЕМБЛЕРа, то вся строка транслируется как оператор .WORD LABEL, а в листинге печатается сообщение об ошибке;

2) символ, допустимый для использования в АССЕМБЛЕРе, (т. е. символ, принадлежащий набору символов АССЕМБЛЕРа), при нарушении основных положений и синтаксиса языка может быть в контексте интерпретирован как недопустимый, и в листинге печатается сообщение об ошибке «Q».

4.2. Символические имена

4.2.1. Постоянные имена, имена пользователя и имена макрокоманд

В АССЕМБЛЕРе используются символические имена трех видов: постоянные имена, имена пользователя и имена макрокоманд. Соответственно формируется три типа таблиц: таблица постоянных имен, таблица имен пользователя и таблица макроимен.

Таблица постоянных имен содержит постоянные имена и входит в состав транслятора. Постоянными именами являются символические обозначения кодов команд и директив АССЕМБЛЕРа. Эти имена нет необходимости определять перед использованием в исходной программе.

Таблица имен пользователя и таблица макроимен создаются в процессе трансляции исходной программы.

К именам пользователя относятся имена, используемые в программе в качестве меток, и имена, определяемые с помощью оператора прямого присваивания. Эти имена добавляются в таблицу имен пользователя по мере того, как они встречаются в исходной программе при трансляции.

Имена макрокоманд добавляются в таблицу макроимен при трансляции по мере того, как они встречаются в исходной программе.

Имена, определяемые пользователем, и имена макрокоманд могут записываться только с помощью букв латинского алфавита, цифр, символа «¤» и точки. Любые другие символы для записи имен не допустимы.

Символы «¤» и «.» зарезервированы для имен системных программ.

При формировании имен, определяемых пользователем, и имен макрокоманд необходимо придерживаться следующих правил:

1) первый символ не должен быть цифрой, за исключением локальных меток;

2) каждое имя должно быть единственным по первым шести символам;

3) имя может содержать более шести символов, но седьмой и последующие символы проверяются только на допустимость и не принимаются во внимание при трансляции;

4) пробелы, символы ГТ и недопустимые символы (например, буквы русского алфавита) нельзя включать в символические имена.

Числовое значение имени зависит от его использования в исходной программе.

Имя, встречающееся в поле операции, может относиться к любому из трех описанных выше видов символических имен. В этом случае для определения значения имени АССЕМБЛЕР осуществляет поиск имени в таблицах имен в следующем порядке:

1) таблица макроимен;

2) таблица постоянных имен;

3) таблица имен пользователя.

Поиск имен, встретившихся в поле операнда, осуществляется в следующем порядке:

1) таблица имен пользователя;

2) таблица постоянных имен.

Предполагается, что АССЕМБЛЕР не должен обнаружить имя макрокоманды в поле операнда.

Такой порядок просмотра таблиц позволяет переопределить имя, содержащееся в таблице постоянных имен, как имя, определенное пользователем, или как имя макрокоманды. Одно и то же имя может быть использовано для обозначения и макрокоманды и метки.

4.2.1.1. Глобальные и локальные имена

Имена, определяемые пользователем, могут быть локальными и глобальными.

Глобальные имена определяются:

— с помощью директивы .GLOBL (являются ее аргументами);

— с помощью символов «::» при определении метки;

— с помощью символов «===» или «===:» в операторе прямого присваивания.

Все остальные символические имена, определяемые пользователем в программе, являются локальными.

Среди локальных имен особое место занимают локальные метки (п. 4.2.3).

К глобальным именам происходит обращение для осуществления связи между объектными модулями внутри загрузочного модуля, который получают в результате связывания из нескольких объектных модулей.

Глобальное имя в программе может быть меткой и его абсолютное значение будет определено в процессе связывания объектных модулей.

Глобальное имя в программе может не являться меткой и его значение будет определяться с помощью оператора прямого присваивания как значение выражения. Глобальное имя, определенное как метка, обычно является точкой входа.

4.2.1.2. Внутренние и внешние имена

Глобальные имена делятся на внутренние глобальные и внешние глобальные имена.

Глобальное имя, не определенное в данном объектном модуле, называется внешним. Оно должно быть определено в каком-то другом модуле, для которого уже будет называться внутренним.

Глобальное имя должно быть зафиксировано в качестве аргумента директивы .GLOBL как в объектном модуле, где

это имя определено, так и в модуле, где оно используется как внешнее.

Поскольку АССЕМБЛЕР обеспечивает возможность секционирования программ (п. 7.7), следует также рассматривать две категории глобальных имен:

1) имена, относящиеся только к текущей секции программы;

2) имена, используемые и другими программными секциями.

В обоих случаях имя должно быть определено к моменту трансляции текущего предложения, так как его значение может использоваться при вычислении выражения.

4.2.2. Имена регистров

Восемь регистров общего назначения пронумерованы от 0 до 7 и могут быть обозначены в исходной программе как: '%0 '%1 %2 %3 %4 %5 %6 %7.

Символ '% может использоваться с любым термом или выражением для определения номера регистра.

Например,
CLR %3+1 эквивалентно CLR '%4.

По этой команде очищается регистр 4, тогда как по команде CLR 4 очищается ячейка оперативной памяти 000004.

Рекомендуется использовать символические имена регистров:
R0='%0 R1=%1 R2=%2 R3=%3 R4=%4 R5=%5 SP=%6 PC='%7

Эти имена регистров являются общепринятыми именами и используются во всех стандартных программах.

Регистрам 6 и 7 даны специальные имена вследствие специфики их использования, в то время как регистрам с номерами от 0 до 5 присвоены однотипные имена, чтобы указать, что они являются универсальными регистрами общего назначения.

Стандартные имена регистров можно переопределить, используя директиву .DSABL REG и оператор прямого присваивания. Без указания директивы .DSABL REG переопределять имена регистров не допускается.

4.2.3. Локальные метки

Локальными метками называют символические имена специального формата, используемые как метки в заданном диапазоне (в блоке локальных меток).

Благодаря использованию локальных меток можно получить значительную экономию места в таблице имен пользователя. Для каждой локальной метки блока локальных меток отводится три слова памяти, а для каждой метки в таблице имен пользователя — четыре слова памяти.

Локальные метки удобно использовать для осуществления условных переходов в исходной программе.

К локальным меткам нельзя обращаться из других объектных модулей или из других блоков локальных меток одного и того же объектного модуля. К ним может осуществляться обращение внутри того блока локальных меток, в котором эта локальная метка определена. Поэтому не произойдет конфликтной ситуации, если точно такая же локальная метка будет определена в любом другом блоке локальных меток.

Локальные метки записываются в форме: N¤,
где N — целое число от 1 до 65535 (десятичное) включительно.

Примеры локальных меток:
1¤ 35¤ 107¤ 120¤

Ниже приведен пример листинга исходной программы с использованием локальных меток.

```
     Пример:
1    000000 012700  XCTPRG:  MOV #IMPURE,R0
            0000006
2    000004 005020  1$:      CLR (R0)+
3    000006 020027          CMP R0,#IMPURT
            0000006
4    000012 0013374           BNE 1$
5
6
7    000014 012700  XCTPAS:  MOV #IMPPAS,R0
            0000006
8    000020 005020  1$:      CLR (R0)+
9    000022 020027          CMP R0,#IMPPAT
            0000006
10   000026 001374          BNE 1$
11   000030 000207          RTS PC
```

Рекомендуется использовать локальные метки от 1¤ до 29999¤.

Локальные метки от 30000¤ до 65535¤ могут формироваться автоматически в процессе трансляции макрокоманд (п. 8.3.3).

Блок локальных меток состоит из последовательности предложений и должен ограничиваться одним из следующих способов:

1) начинаться с символической метки и заканчиваться символической меткой;

2) начинаться с символической метки и заканчиваться одной из директив .PSECT .ASECT .CSECT;

3) начинаться с символической метки, следующей за директивой .ENABL LSB, и заканчиваться символической меткой или директивой .CSECT, за которыми в том и в другом случае следует директива .DSABL LSB. Подразумевается, что по умолчанию выполняются функции директивы .DSABL LSB.

ПРИМЕЧАНИЕ. Необходимо иметь ввиду, что предложение типа LABEL=. , являющееся оператором прямого присваивания, не определяет символическое имя как метку и не может быть ограничителем блока локальных меток.

4.2.4. Имя счетчика адреса

В качестве условного обозначения текущего значения счетчика адреса в АССЕМБЛЕРе используется символ точка (.).

Когда точка используется в поле операнда команды, ей ставится в соответствие значение адреса первого слова команды. Когда точка используется в поле операнда директивы АССЕМБЛЕРа, ей присваивается значение адреса текущего байта или слова.

Пример:

```
A: MOV #.,R0 ; обозначает адрес ячейки с меткой A,
             ; т. е. адрес данной команды
```

В начале каждого прохода АССЕМБЛЕР очищает счетчик адреса программы.

В общем случае каждому байту данных, транслируемых АССЕМБЛЕРом, присваиваются последовательные адреса. Однако, текущее значение счетчика адреса можно изменить с помощью оператора прямого присваивания: . = Выражение.

Как и любое символическое имя, имя текущего значения счетчика адреса должно быть определено либо в абсолютной, либо в перемещаемой секции. Текущий (абсолютный или перемещаемый) вид значения счетчика адреса может быть изменен на противоположный только при переходе из одной секции в другую, т. е. после директив .ASECT или .CSECT соответственно.

Примеры:

```
        .ASECT
.=500                   ;значение счетчика адреса уста-
                        ;навливается равным 500 (абсо-
                        ;лютное)
FIRST: MOV .+10,COUNT   ;метке FIRST ставится в соответ-
                        ;ствие значение 500 (абсолют-
                        ;ное).
                        ;.+10 эквивалентно 510 (абсолют-
                        ;ное). Содержимое ячейки 510 за-
                        ;писывается в ячейку с меткой
                        ;COUNT
.=520                   ;значение счетчика адреса уста-
                        ;навливается равным 520 (абсо-
                        ;лютное)
SECOND: MOV .,INDEX     ;значение метки SECOND равно
                        ;520 (абсолютное)
                        ;содержимое ячейки 520, т. е. дво-
                        ;ичное значение команды «MOV
                        ;.,INDEX» записывается в ячейку
                        ;с меткой INDEX
INDEX: .WORD 0          ;
        .CSECT          ;значение счетчика адреса уста-
.=.+20                  ;навливается равным значению
                        ;счетчика адреса перемещаемой
                        ;неименованной секции плюс 20
THIRD: .WORD 0          ;значение метки THIRD равно 20
                        ;(перемещаемое)
```

Выражение, определяющее значение счетчика адреса, не должно содержать ссылки **вперед и имена**, значения которых изменяются при трансляции **от одного** прохода к другому.

С помощью оператора **прямого присваивания можно** зарезервировать определенную **область** памяти. **Например**, если значение текущего адреса **равно** 1000, оператор .=.+100 резервирует 100 байт оперативной памяти. Следующая команда запишется, начиная с ячейки 1100.

Область памяти можно также резервировать с помощью директив .BLKW и .BLKB (п. 7.5.3).

4.3. Данные

Величины, над которыми осуществляются действия в процессе выполнения программы, называются данными. В качестве данных языка используют константы и выражения.

4.3.1. Константы

Константа является данным, которое всегда определено и не меняется в процессе выполнения программы.

Константа всегда имеет абсолютное значение.

Допускаются следующие виды констант:

— целые;

— вещественные;

— символьные.

Константа без знака или со знаком плюс означает положительное число, а со знаком минус — отрицательное.

АССЕМБЛЕР все числа в исходной программе интерпретирует как восьмеричные, если не задана другая система счисления.

Система счисления может быть изменена директивой .RADIX.

Если число не определено как десятичное, но содержит цифры 8 и 9, в листинге печатается сообщение об ошибке «N» и число рассматривается как десятичное.

Если число слишком большое для размещения в 16-разрядной ячейке, оно усекается слева, и в листинге печатается сообщение об ошибке «T».

4.3.1.1. Целая константа

Целые константы подразделяются на восьмеричные, десятичные, двоичные.

Восьмеричные константы могут быть заданы в обратном коде.

Восьмеричная константа записывается в виде непустой последовательности цифр 0—7. По умолчанию все константы обрабатываются как восьмеричные.

Восьмеричная константа в исходной программе может задаваться с помощью указателя ^O.

Пример:

```
2540            ; 2540 восьмеричное число
^O 47           ; 47 восьмеричное число
^O <A+13>       ; A+13 имеет восьмеричное значение
```

Десятичная константа записывается в виде непустой последовательности цифр 0—9.

Десятичная константа в исходной программе задается с помощью точки (.) или указателя ^D.

Пример:

```
1376.           ; 1376 десятичное число
^D 123          ; 123 десятичное число
```

Двоичная константа записывается в виде последовательности цифр 0 и 1.

Двоичная константа в исходной программе задается с помощью указателя ∧B.

Пример:

∧B 11000111 ; 11000111 двоичное число

Восьмеричная константа может быть задана в обратном коде с помощью указателя ∧C.

Пример:

.WORD ∧C 151 ; в ячейке запоминается 177626

4.3.1.2. Вещественная константа

Вещественная константа задается с помощью указателя ∧F и занимает одно слово. Формат вещественной константы приведен на рисунке.

Формат числа с плавающей запятой: +N.NE+P или —N.NE+P или +N.NE—P или —N.NE—P.

где N — десятичные цифры, определяющие целую и дробную часть числа;

P — десятичная цифра, определяющая порядок числа (степень 10);

E — основание степени 10;

. — отделяет целую часть числа от дробной.

Формат числа с плавающей запятой, занимающего одно слово

15 14 7 6 0

: :	Порядок	:	Мантисса	:

:<——————знак числа

Примеры:

∧F 1.0 = 040200

∧F —1.0 =140200

—∧F 1.0 =137600

—∧F —1.0 = 037600

ПРИМЕЧАНИЕ. Для записи вещественных констант в два и четыре слова используются директивы .FLT2 и .FLT4 (п. 7.3.7).

4.3.1.3. Символьная константа

Символьная константа может состоять из одного или двух символов КОИ-7.

В качестве значения константы используется 16-разрядный код.

Для преобразования одного символа КОИ-7 в 16-разрядное значение используется апостроф. При этом в младший

байт записывается 7-разрядный код символа, а в старший байт — ноль.

Пример:

```
MOV #'A,R0 ; код символа A (000101) пересылается в R0
```

Два символа КОИ-7 преобразуются в 16-разрядный код с помощью кавычек. При этом 7-разрядные коды символов записываются в одно слово: код первого символа — в младший байт, код второго символа — в старший байт.

Пример:

```
MOV #"AB,R0 ; коды символов AB (041101) пересылаются в
            ; R0
```

Символьное данное, состоящее из трех символов, может быть преобразовано в код RADIX-50 с помощью указателя ^R. Результат записывается в одно слово. Если указано более трех символов, то четвертый и последующие символы игнорируются.

Набор символов, допустимых для преобразования в код RADIX-50, приведен в приложении 1.

Для преобразования последовательности более чем из трех символов в код RADIX-50 может быть использована директива .RAD50 (п. 7.3.5).

Пример:

```
MOV #^RMAC,R0 ; символы M,A,C преобразуются в код
              ; RADIX-50,
              ; который пересылается в R0
```

4.3.2. Выражения

Выражение — это терм или несколько термов, соединенных вместе знаками двухместных операций (см. табл. 4). Термом может быть:

— константа;
— символическое имя;
— выражение, заключенное в символы «<» и «>»;
— символическое имя или константа, которой предшествует знак одноместной операции.

Значением выражения является 16-разрядная величина.

Выражения обрабатываются слева направо без соблюдения правил приоритета выполнения операций, за исключением одноместных операций, которые выполняются первыми по сравнению с двухместными операциями.

Отсутствующий терм, выражение или внешнее имя интерпретируется как ноль. Пробелы внутри выражения игнорируются. Пропущенное или недопустимое предложение приводит к прекращению анализа выражения и вызывает печать в листинге сообщений об ошибке «A» и/или «Q».

4.3.2.1. Одноместные и двухместные операции

По своему назначению операции АССЕМБЛЕРа делятся на одноместные и двуместные.

В зависимости от использования в выражениях операции могут быть арифметические, логические и специальные.

Одноместные операции (см. табл. 3) действуют только на один терм (компонент выражения) и определяют, какое действие необходимо выполнить над данным термом. Терм, которому предшествует одноместная операция, рассматривается как содержащий данную операцию. Плюс, минус и специальные операции относятся к одноместным операциям. Специальные операции — это операции преобразования данных из одной формы представления в другую, записываемые со знаком «Λ».

Двухместные операции указывают действия, выполняемые над несколькими термами в выражении (см. табл. 4). К двухместным операциям относятся арифметические и логические операции.

Двухместные операции имеют одинаковый приоритет.

4.3.2.2. Виды выражений

Выражения могут быть абсолютными, перемещаемыми, внешними и составными перемещаемыми.

Абсолютное выражение определяется как:

— терм или несколько термов, являющихся константами;

— перемещаемое выражение или терм минус перемещаемый терм;

- несколько термов, являющихся метками, определенными в абсолютной программной секции.

Перемещаемым выражением является выражение, значение которого фиксировано по отношению к базовому адресу перемещаемой программной секции, в которой находится это выражение. При связывании значение выражения меняется.

Перемещаемое выражение определяется как:

- перемещаемый терм;

- перемещаемый терм (метка или имя счетчика адреса перемещаемой программной секции), связанный знаком арифметической операции с абсолютным выражением.

Внешним выражением является выражение, которое содержит внешнее имя.

Внешнее выражение определяется как:

- внешний терм;

- внешний терм, связанный знаком арифметической операции с абсолютным термом;

— абсолютное выражение, связанное знаком операции сложения с внешним выражением.

Составное перемещаемое выражение содержит несколько перемещаемых или внешних термов.

Выражение является составным перемещаемым, если выполняется любое из следующих условий:
— выражение содержит глобальное имя и перемещаемое имя;
— выражение содержит более одного глобального имени;
— выражение содержит перемещаемые термы, принадлежащие различным программным секциям;
— значение, получающееся в результате вычисления выражения, имеет более одного уровня перемещения. Например, если перемещаемые имена TAG1 и TAG2, принадлежащие одной и той же программной секции, указаны в выражении вида TAG1+TAG2, то вводятся два уровня перемещения, так как при оценке каждого имени принимается во внимание смещение, получающееся в результате перемещения данной программной секции;
— для неопределенного глобального имени указана операция, отличная от сложения;
— для перемещаемого значения указана операция, отличная от сложения, вычитания, отрицания или дополнения.

Вычисление перемещаемых, внешних и составных перемещаемых выражений завершается во время связывания программных секций.

ПРИМЕЧАНИЕ. Пробелы внутри выражений могут иметь значение только между символами. Другими словами, выражения
A + B и A+B
являются тождественными, но символические имена B17 и B 17 не тождественны.

Примеры:

```
   .ASECT                ;значение имени ABBSYM абсо-
 .=100                   ;лютное, так как оно определено
ABBSYM=.                 ;в абсолютной секции
   .CSECT MAIN           ;начало перемещаемой програм-
                         ;мной секции
   .GLOBL EXTVAL         ;имя EXTVAL является внешним
                         ;глобальным, оно определено в
                         ;другом программном модуле,
                         ;его значение будет оставаться
                         ;неизменным до связывания
BEGSYM: .BLKW 4          ;значения имен BEGSYM и
   .ASCII /ABCD/         ;ENDSYM являются перемещае-
```

```
                        ; мыми, так как адрес, начиная с
    .EVEN               ; которого будет загружаться про-
    ENDSYM=.            ; граммная секция с именем
                        ; MAIN, будет определен лишь во
                        ; время связывания
SIZE=ENDSYM—BEGSYM      ; значение имени SIZE становится
                        ; уже известным (оно равно
                        ; 12(10)) во время трансляции и
                        ; является абсолютным
RELEXP=                 ; значение имени RELEXP (рав-
=ENDSYM—BEGSYM+.        ; ное .+12) является перемещае-
                        ; мым
EXTEXP:.WORD EXTVAL+4   ; выражение EXTVAL+4 явля-
                        ; ется внешним глобальным, так
                        ; как имя EXTVAL определено в
                        ; другом программном модуле
CHARA='A                ; значение имени CHARA, равное
                        ; 'A, является абсолютным
```

4.4. Оператор прямого присваивания

Оператор прямого присваивания вычисляет выражение и присваивает полученное значение символическому имени.

Формат оператора прямого присваивания:

символическое имя = выражение или
символическое имя == выражение или
символическое имя =: выражение или
символическое имя ==: выражение

Символическое имя принимает абсолютное или перемещаемое значение в зависимости от определяющего его выражения.

Выражение в операторе прямого присваивания может содержать только один уровень ссылки вперед и не должно содержать ссылку на внешнее имя.

Операторы прямого присваивания, содержащие символы «==» и «==:», определяют имя как глобальное.

После того, как символическое имя определено с помощью оператора прямого присваивания, оно включается в таблицу имен пользователя.

Значение символического имени может быть переопределено последующим оператором прямого присваивания, если в операторе прямого присваивания используются символы «=» и «==». Если в операторе прямого присваивания используются символы «=:» или «==:», любая попытка изменить значение символического имени вызывает печать в листинге сообщения об ошибке «M».

При использовании оператора прямого присваивания должны соблюдаться следующие правила:

1) символы «=», «==», «=:», «==:» отделяют имя от выражения, значение которого присваивается имени;

2) оператор прямого присваивания обычно помещается в поле операции, перед ним может быть метка, а после него — комментарий;

3) одним оператором прямого присваивания может быть определено только одно имя;

4) допускается только один уровень ссылки вперед, например, X=Y Y=1;

5) оператор прямого присваивания, определяющий глобальное имя, не должен содержать ссылку вперед.

5. ПЕРЕМЕЩЕНИЕ И СВЯЗЫВАНИЕ

Результатом работы АССЕМБЛЕРа является объектный модуль, который должен быть обработан редактором связей до его загрузки и выполнения.

Редактор связей фиксирует (т. е. делает абсолютными) значения имен в перемещаемых секциях и преобразует объектный модуль в загрузочный модуль. Загрузочный модуль — это программа, готовая к загрузке в основную память для выполнения.

Для того, чтобы редактор связей имел информацию, необходимую для фиксирования значений выражений и имен, АССЕМБЛЕР выдает в объектном модуле ряд указаний с требуемыми параметрами.

Если выражение перемещаемое, редактор связей прибавляет базовый адрес соответствующей перемещаемой программной секции к величине выражения, определенного АССЕМБЛЕРом при трансляции программы.

Если выражение внешнее, редактор связей определяет значение внешнего терма выражения и прибавляет его к значению выражения, полученного при трансляции программы.

Все слова, которые должны быть модифицированы, при распечатке листинга программы отмечаются символами «'», «G» или «C»:

«'» — требуется простое перемещение;

«G» — к абсолютной части выражения должно быть прибавлено значение внешнего символического имени;

«C» — для фиксирования выражения редактор связей должен выполнить анализ сложного перемещения.

Пример:

```
005065 CLR RELOC(R5)      ;предполагается, что значение
000040'                   ;имени RELOC является перемещаемым. Редактор связей прибавит к нему смещение от перемещения модуля
005065 CLR EXTERN(R5)     ;внешнему глобальному имени
000000G                   ;EXTERN при трансляции ставится в соответствие значение
                          ;ноль. Действительное значение
                          ;будет определено редактором
                          ;связей
005065 CLR EXTERN+6(R5)   ;абсолютная часть выражения
000006G                   ;(000006) будет прибавлена к значению внешнего имени EXTERN
                          ;в процессе связывания
005065 CLR                ;выражение является составным
—<EXTERN+                 ;перемещаемым, так как оно со-
+RELOC>(R5)               ;держит глобальное имя EX-
000000C                   ;TERN и перемещаемый терм,
                          ;взятые со знаком минус
```

6. КОМАНДЫ

Команды АССЕМБЛЕРа делятся на безадресные, одноадресные и двухадресные. Безадресные команды содержат только символическое имя команды. Одноадресные и двухадресные команды содержат символическое имя команды, метод(ы) адресации, регистр(ы) общего назначения и/или адрес.

Набор команд, допустимых в языке АССЕМБЛЕР, приведен в приложении 3.

6.1. Методы адресации

Счетчик команд PC (R7) является одним из восьми регистров общего назначения и всегда содержит адрес следующего слова, т. е. адрес следующей команды, которую нужно выполнить, или адрес второго или третьего слова текущей команды.

Всякий раз, когда процессор использует счетчик команд для выборки слова из памяти, содержимое счетчика команд увеличивается на 2, что равносильно указанию на следующее слово в памяти, т. е. когда команда выбрана из памяти, содержимое счетчика команд увеличивается на 2, чтобы указать следующее слово в памяти.

Например, если команда использует индексный метод адресации, то после выборки команды и увеличения PC на 2 процессор снова обращается к памяти за индексным словом, после чего еще раз увеличивает на 2 содержимое PC.

Введем следующие обозначения:

1) E — выражение, как оно определено выше (см. п. 4.3.2);

2) R — выражение для регистра. Это любое выражение, содержащее терм, которому предшествует символ %, или имя, предварительно присвоенное такому терму.

Примеры:

```
R0=%0   ; регистр общего назначения 0
R1=%1   ; регистр общего назначения 1
R2=%2   ; регистр общего назначения 2
```

3) ER — выражение для идентификации регистра, т. е. выражение, значение которого заключено в пределах от 0 до 7;

4) A — выражение для указания метода адресации, занимающего 6-разрядное поле операнда;

Выражение A может быть представлено термами, обозначенными: E, R, ER. В примерах для иллюстрации одноадресных команд используется команда CLR, для двухадресных — команда MOV.

Методы адресации приведены в приложении 4.

6.1.1. Регистровый метод

Формат: R

Регистр содержит операнд

Пример:

```
CLR R3 ; очистить регистр 3
```

6.1.2. Косвенно-регистровый метод

Формат: @R или (ER)

Регистр содержит адрес операнда

Примеры:

```
CLR  @R1    ;очистить ячейку, адрес которой  находится
CLR  (R1)   ;в R1
CLR  (%1)
```

6.1.3. Автоинкрементный метод

Формат: (ER)+

Регистр содержит адрес операнда. Содержимое регистра после его использования как адреса операнда автоматически увеличивается на 1 или 2.

Примеры:

```
CLR  (R0)+   ;каждая из этих команд очищает ячейку по
CLR  (R4)+   ;адресу, содержащемуся в указанном регист-
CLR  (R2)+   ;ре 0, 4, 2, и затем увеличивает содержимое
             ;этого регистра на 2
```

6.1.4. Косвенно-автоинкрементный метод

Формат: @(ER)+

Регистр содержит указатель адреса операнда, т. е. адрес адреса операнда. Содержимое регистра после его использования как указателя адреса операнда автоматически увеличивается на 2.

Пример:

```
CLR @(R3)+  ; регистр 3 содержит указатель адреса ячейки,
            ; которая очищается. Затем содержимое реги-
            ; стра 3 увеличивается на 2
```

6.1.5. Автодекрементный метод

Формат: —(ER)

Регистр содержит адрес операнда. Содержимое регистра автоматически уменьшается на 1 или 2, а затем используется как адрес операнда.

Примеры:

```
CLR —(R0)   ; содержимое регистров 0, 3, 2 уменьшается на
CLR —(R3)   ; 2 перед использованием каждого из них в
CLR —(R2)   ; качестве адреса операнда
```

6.1.6. Косвенно-автодекрементный метод

Формат: @—(ER)

Регистр содержит указатель адреса операнда. Содержимое регистра автоматически уменьшается на 2, а затем используется как указатель адреса операнда.

Пример:

```
CLR @—(R2)  ; содержимое регистра 2 уменьшается перед
            ; использованием его как указателя адреса
            ; операнда
```

6.1.7. Индексный метод

Формат: E(ER)

Значение выражения E (индексное слово) запоминается во втором или третьем слове команды. Исполнительный адрес операнда вычисляется как сумма значений E и содержимого регистра ER.

Примеры:

```
CLR X+2(R1) ; исполнительный адрес равен X+2 плюс со-
            ; держимое регистра 1
MOV R0,—2(R3) ; исполнительный адрес равен —2 плюс содер-
            ; жимое регистра 3
```

6.1.8. Косвенно-индексный метод

Формат: @E(ER)

Значение выражения E и содержимое регистра, определяемое выражением ER, складываются и сумма используется как указатель адреса операнда.

Пример:

```
CLR @114(R4); если регистр 4 содержит значение 100 и со-
            ; держимое ячейки 214 равно 2000, то ячейка
            ; с адресом 2000 очищается
```

6.1.9. Непосредственный метод

Формат: #E

Операнд E запоминается во втором или третьем слове команды. Непосредственный метод формируется как автоинкрементный метод адресации с использованием счетчика команд, т. е. PC. **Пример:**

```
MOV #100,R0  ; занести 100 в регистр 0
```

Символ # используется как указатель непосредственного метода адресации. Команда MOV #100,R0 транслируется как двухсловная команда, операнд помещается во втором слове команды: (при использовании FPP непосредственный операнд транслируется как число с плавающей точкой)

```
012703
000100
```

Непосредственно перед тем как эта команда будет выбрана и исполнена, PC указывает на первое слово команды. Процессор выбирает первое слово и увеличивает PC на 2. Метод адресации второго операнда — 27 (автоувеличение PC). Таким образом, PC используется как указатель операнда (второе слово команды) перед тем, как его содержимое будет увеличено на 2, т. е. будет содержать адрес следующей команды.

6.1.10. Абсолютный метод

Формат: @#E

Абсолютный адрес операнда, задаваемый выражением @#A, запоминается во втором или третьем слове команды. Это достигается использованием косвенно-автоинкрементного метода адресации применительно к счетчику команд.

Примеры:

```
MOV@#100,R0; запомнить содержимое ячейки 100 в регист-
           ; ре 0
CLR @#X    ; очистить ячейку, адрес которой равен значе-
           ; нию символа X
```

6.1.11. Относительный метод

Формат: E

Относительный метод адресации представляет собой индексную адресацию с использованием PC. Индексное слово хранится во втором или третьем слове команды, а адрес операнда в этом случае вычисляется относительно текущего значения PC как сумма PC и индексного слова.

Относительный метод адресации удобно использовать для обращения к любой ячейке памяти.

Примеры:

```
CLR 100        ; очистить ячейку с адресом 100
MOV 100,R3     ; занести содержимое ячейки 100 в регистр 3
```

Используя обозначение счетчика адреса и счетчика команд, второй операнд можно записать в следующем виде:

```
MOV 100—.—4(PC),R3
```

Этот метод адресации называется относительным, так как адрес операнда вычисляется относительно текущего значения счетчика адреса.

При трансляции команды, использующей относительный метод адресации, индексное слово определяется как разность между адресом операнда и текущим значением счетчика адреса (.) и записывается во второе или третье слово команды.

Например, если оператор «MOV 100,R3» помещен по абсолютному адресу 20, то в результате трансляции получим:

```
000020 016703
000022 000054
```

При выполнении команды «MOV 100,R3» будет происходить следующее:

1) выборка команды из ячейки 20;

2) увеличение содержимого PC на 2 (PC=22), т. е. получение адреса ячейки, содержащей индексное слово;

3) выборка индексного слова из ячейки с адресом 22;

4) увеличение содержимого PC на 2 (PC=24) и исполнение команды, при котором адрес операнда источника будет определен как сумма содержимого PC и индексного слова, т. е. 24+54=100.

Использование относительного метода адресации позволяет получить программу позиционно независимую, т. е. работоспособность такой программы сохраняется при перемещении ее в памяти ЭВМ.

6.1.12. Косвенно-относительный метод

Формат: @E

Указатель адреса операнда задается с помощью PC и индексного слова. Этот метод адресации аналогичен относительному методу адресации, только сформированный адрес является указателем адреса операнда.

Это достигается использованием косвенно-индексного метода адресации применительно к PC.

Пример:

```
MOV @X,R0      ; занести в регистр 0 содержимое ячейки, ад-
               ; рес которой находится в X
```

6.1.13. Форматы методов адресации

Методы адресации, приведенные в табл. 5, не увеличивают длину команды.

Таблица 5

Формат метода адресации	Код метода адресации	Наименование
R	0N	Регистровый
@R или (ER)	1N	Косвенно-регистровый
(ER)+	2N	Автоинкрементный
@(ER)+	3N	Косвенно-автоинкрементный
—(ER)	4N	Автодекрементный
@—(ER)	5N	Косвенно-автодекрементный

Методы адресации, приведенные в табл. 6, увеличивают длину команды на одно слово.

Таблица 6

Формат метода адресации	Код метода адресации	Наименование
E(ER)	6N	Индексный
@E(ER)	7N	Косвенно-индексный
#E	27	Непосредственный
@#E	37	Абсолютный
E	67	Относительный
@E	77	Косвенно-относительный

ПРИМЕЧАНИЯ:

1. В графе «Код метода адресации» N — один из регистров общего назначения.

2. Последние четыре метода адресации в качестве регистра общего назначения используют счетчик команд.

3. С помощью директивы .ENABL AMA можно в процессе трансляции заменить относительный метод адресации на абсолютный метод адресации в командах программы (п. 7.2.1).

6.2. Адресация в командах ветвления

Команды ветвления вызывают передачу управления по адресу, являющемуся суммой текущего содержимого PC и смещения, умноженного на 2.

За текущее содержимое PC берется адрес следующей команды.

Смещение записывается в команде ветвления в разрядах 0—7 и указывает после умножения на 2, на сколько слов нужно перейти от ячейки, адрес которой в данный момент содержится в PC. Седьмой разряд смещения является знако-

ным. Если он установлен, то смещение отрицательное и ветвление происходит в направлении уменьшения адресов, а если этот разряд очищен, то смещение положительное и ветвление происходит в направлении увеличения адресов программы.

Исполнительный адрес в команде ветвления аппаратно вычисляется следующим образом:

1) формируется 16-разрядное слово, младший байт которого является смещением, а разрядам 15—8 присваивается значение знакового разряда смещения;

2) полученное слово умножается на 2, т. е. образуется смещение, выраженное в словах, а не в байтах;

3) результат складывается с содержимым PC для формирования исполнительного адреса.

АССЕМБЛЕР выполняет обратную операцию для формирования смещения в байтах относительно заданного адреса.

При использовании команд ветвления необходимо следить за тем, чтобы не было:

— перехода из одной программной секции в другую;

— перехода по адресу, который определен как внешнее имя;

— перехода по адресу, который находится за пределами действия команды ветвления, т. е. смещение перехода должно быть в пределах от —128(10) до +128(10). В случае нарушения одного из этих условий в соответствующей позиции текущей строки будет напечатано сообщение об ошибке «A» и установлено смещение 377 (восьмеричное).

6.3. Адресация в системных командах EMT и TRAP

В командах EMT и TRAP старший байт слова содержит код команды, а младший байт предназначен для передачи информации драйверам внутренних прерываний.

Если EMT (или TRAP) сопровождается выражением, то при трансляции значение этого выражения запоминается в младшем байте слова. Если значение выражения превышает значение 377(8), оно усекается до 8 бит и печатается сообщение об ошибке «T».

7. ДИРЕКТИВЫ

В данном разделе описываются директивы языка АССЕМБЛЕР (см. приложение 5), подразделяющиеся на:

1) директивы управления листингом;
2) директивы режима трансляции;
3) директивы задания данных;
4) директива управления системой счисления;
5) директивы управления счетчиком;

6) директива окончания;
7) директивы секционирования;
8) директивы описания имен;
9) директивы условной трансляции;
10) директивы управления файлами.

7.1. Директивы управления листингом. Директивы управления листингом управляют содержанием, форматом и формированием страниц листинга, выдаваемых на терминал и построчно-печатающее устройство.

К директивам управления листингом относятся: .LIST .NLIST .TITLE .SBTTL .IDENT .PAGE .REM

7.1.1. Директивы .LIST и .NLIST. Директивы .LIST и .NLIST используются для управления печатью определенных аргументами полей листинга: — .LIST разрешить печать, .NLIST — запретить печать.

Формат:

.LIST [A]

.NLIST [A]

где A — один или несколько аргументов (табл. 7), разделенных запятыми, пробелами или символами ГТ.

Директивы .LIST и .NLIST могут быть заданы без аргументов. В этом случае данные директивы изменяют значение счетчика уровня печати. Счетчик уровня печати может принимать отрицательное, положительное и нулевое значение. При отрицательном значении счетчика уровня печать листинга запрещается (за исключением строк, содержащих ошибки), при положительном — разрешается, при нулевом — строка листинга печатается или не печатается в зависимости от других управляющих параметров, которые заданы в данный момент в программе.

Счетчик уровня печати увеличивается на 1 по директиве .LIST и уменьшается на 1 по директиве .NLIST. Первоначальное значение счетчика уровня печати равно нулю.

Основное назначение счетчика уровня печати — обеспечить выборочную распечатку макрорасширений, при этом значение счетчика уровня печати при выходе должно совпадать с тем, каким оно было до обращения к макрокоманде.

Директивы .LIST и .NLIST с аргументами управляют печатью полей листинга, задаваемых аргументами. Аргументы могут использоваться индивидуально или в комбинации друг с другом. Для любого аргумента, не включенного явно в директиву управления печатью листинга, используется соответствующее значение аргумента по умолчанию.

Таблица 7

СИМВОЛИЧЕСКИЕ АРГУМЕНТЫ ДИРЕКТИВ .LIST и .NLIST

Аргумент	Действие по умолчанию	Функция
SEQ	Печать	Управление печатью порядковых номеров строк исходной программы. Строки исходной программы нумеруются последовательно, начиная с 1. Директива .NLIST SEQ запрещает нумерацию строк в листинге, при этом поле порядкового номера строки выделяется с помощью пробелов. Расположение других полей листинга не нарушается
LOC	»	Управление печатью значений счетчика адреса. Директива .NLIST LOC запрещает печать значений счетчика адреса. Поле счетчика адреса не выделяется с помощью пробелов и происходит выравнивание слева всех последующих полей листинга
BIN	»	Управление печатью объектных кодов в восьмеричном виде. Директива .NLIST BIN запрещает печать указанных кодов, происходит выравнивание слева всех последующих полей листинга
BEX	»	Управление печатью объектных кодов в восьмеричном виде, расположенных на нескольких строках. Директива .NLIST BEX запрещает печать указанных кодов, кроме кодов первой строки
SRC	»	Управление печатью предложений исходной программы
COM	»	Управление печатью комментария. Директива .NLIST COM сокращает время распечатки и объем листинга в случае, когда комментарии не требуются
MD	»	Управление печатью макроопределений и областей повторений
MC	»	Управление печатью макрокоманд
ME	Печать запрещена	Управление печатью макрорасширений и расширений областей повторений
MEB	»	Управление печатью предложений макрорасширений, порождающих объектные коды
LD	»	Управление действием директив .LIST и .NLIST без аргументов
CND	Печать	Управление печатью блоков условной трансляции с невыполненными условиями
TOC	»	Управление печатью оглавления
SYM	»	Управление печатью таблицы имен и таблицы перекрестных ссылок
PAG	»	Управление форматом печати объектных кодов, таблицы имен и таблицы перекрестных ссылок

ПРИМЕЧАНИЕ. Директива .NLIST SEQ, LOC, BIN, SRC запрещает печать всех полей листинга, при этом пустая строка игнорируется.

Директивы .LIST и .NLIST с аргументами не изменяют значение счетчика уровня печати, однако данные директивы могут использоваться для переопределения действия директив управления печатью листинга.

Пример:

```
.MACRO XX


        .LIST           ; печатать следующую строку
X=.
        .NLIST          ; не печатать оставшуюся часть
                        ; макрорасширения

        .NLIST ME       ; не печатать макрорасширение
        XX
X=.
```

Управлять печатью листинга можно с помощью переключателей командной строки АССЕМБЛЕРа (см. [1]). Действия переключателей аналогичны действию директив .LIST, .NLIST и позволяют изменить действия этих директив, используемых в исходной программе.

7.1.2. Директива .TITLE. Директива .TITLE используется для присваивания имени объектному модулю.

Формат:

```
.TITLE C                ; комментарий
```

где C — имя объектного модуля.

Имя может состоять из шести символов, допустимых в коде RADIX-50. Пробелы, следующие за директивой .TITLE, в имя не включаются. Символы, следующие за первыми шестью, проверяются на соответствие алфавиту языка, но не включаются в имя объектного модуля, но будут включаться вместе с именем объектного модуля в заголовок каждой страницы листинга.

Если в исходной программе встречается несколько директив .TITLE, то объектному модулю присваивается имя, указанное в последней директиве .TITLE. Если директива .TITLE отсутствует, то объектному модулю АССЕМБЛЕР присваивает имя .MAIN.

7.1.3. Директива .SBTTL. Директива .SBTTL используется для формирования оглавления листинга и для обозначения каждой страницы листинга.

Формат:

```
.SBTTL C
```

где С — текст, который печатается в оглавлении и в заголовке листинга.

В оглавление листинга включаются порядковый номер строки, номер страницы и текст, сопровождающий каждую директиву .SBTTL. Текст, заданный в .SBTTL, печатается в заголовке на каждой странице листинга до появления следующей директивы .SBTTL, изменяющей заголовок страницы.

ПРИМЕЧАНИЕ. Печать оглавления запрещается по директиве .NLIST TOC.

Пример:

```
.SBTTL CONDITIONAL ASSEMBLIES
```

Текст «CONDITIONAL ASSEMBLIES» будет печататься в заголовке каждой страницы листинга.

7.1.4. Директива .IDENT. Директива .IDENT используется для дополнительного обозначения объектного модуля, создаваемого АССЕМБЛЕРом.

Формат:

```
.IDENT /С/
```

где С — последовательность не более чем из шести символов, допустимых в коде RADIX-50;

/ / — ограничители (любые символы за исключением символов ; , =, <).

В дополнение к имени, присвоенному объектному модулю по директиве .TITLE (см. п. 7.1.2), указывается последовательность символов, допустимых в коде RADIX-50, которая может быть использована для указания номера версии программы.

ПРИМЕЧАНИЯ:

1. Номер версии программы, заданный в директиве .IDENT, упаковывается в код RADIX-50 и записывается в словарь глобальных имен объектного модуля. Номер версии печатается в карте загрузки и в листинге каталога библиотеки.

2. Принимается во внимание только первая директива .IDENT в исходной программе.

Пример:

```
.IDENT /V05A/
```

Директива .IDENT определяет номер версии программы V05A.

7.1.5. Директива .PAGE. Директива .PAGE вызывает печать следующего за ней текста с новой страницы листинга.

Формат:

```
.PAGE
```

Если директива .PAGE используется в макроопределении, то во время трансляции макроопределения она игнорируется, но при распечатке макрорасширения происходит формирование *новой страницы листинга.*

Формирование страницы листинга осуществляется не только по директиве .PAGE, но если счетчик строк станет равным 58 или в исходной программе встретится символ ПФ. Если символ ПФ появляется в макроопределении, то формирование страницы листинга осуществляется во время трансляции макроопределения. Формирование страницы листинга осуществляется также по началу файла, в том числе при директиве .INCLUDE.

7.1.6. Директива .REM. Директива .REM позволяет ввести комментарий в исходную программу без использования символа «;». Комментарий может содержать любое число строк.

Формат:

.REM /коммент/

где коммент — текст комментария;
/ / — ограничители (любые допустимые символы).

Пример:

```
.TITLE REMARK EXAMPLE
.REM &
COMMENT&
CLR PC
.END
```

7.2. Директивы режима трансляции. Директивы режима трансляции используются для управления функциями трансляции и печатью таблицы перекрестных ссылок.

К директивам режима трансляции относятся: .ENABL; .DSABL; .CROSS; .NOCROSS

7.2.1. Директивы .ENABL и .DSABL. Директивы .ENABL и .DSABL используются для управления функциями трансляции: .ENABL — разрешить выполнение функции, .DSABL — запретить выполнение функции.

Формат:

.ENABL A

.DSABL A

где A — один или несколько символических аргументов

(табл. 8), разделенных запятыми, пробелами или символами ГТ.

Таблица 8

СИМВОЛИЧЕСКИЕ АРГУМЕНТЫ ДИРЕКТИВ .ENABL и .DSABL

Аргумент	Действие по умолчанию	Функция
ABS	Запрещение	Вывод модуля в абсолютном двоичном формате
AMA	»	Замена относительного метода адресации (67) на абсолютный (37)
CRF	Разрешение	Включение символических имен, используемых в исходном модуле, в таблицу перекрестных ссылок Функция имеет смысл в случае, если в командной строке транслятора задан переключатель /C арг
FPT	Запрещение	Усечение (ENABL FPT) или округление (DSABL FPT) представление чисел с плавающей запятой
LC	Разрешение	Преобразование вводимых символов КОИ 7 нижнего регистра в символы верхнего регистра Если эта функция запрещена, то весь текст вводится без преобразования
LCM	Запрещение	Использование символов верхнего/нижнего регистра в директивах .IF IND и IF DIF
LSB	»	Открытие или закрытие блока локальных меток
MCL	»	Поиск неопределенных символических имен в макробиблиотеке пользователя и системной макробиблиотеке
PNC	Разрешение	Включение объектных кодов в объектный модуль
REG	»	Переопределение имен регистров, назначенных по умолчанию
GBL	Запрещение	Обработка неопределенных символических имен как внешних

Блок локальных меток обычно устанавливается при появлении символической метки, директивы .PSECT или директивы .RESTORE.

Директива .ENABL LSB устанавливает новый блок локальных меток, который оканчивается при появлении директивы DSABL LSB

Основное применение эта директива находит тогда, когда требуется временно выйти из программной секции для записи данных, после чего последует возврат в данную програм-

мную секцию. Временный выход из программной секции может быть выполнен с помощью директивы .SAVE или .RESTORE (пп. 7.7.3, 7.7.4).

Попытка использовать локальные метки, определенные в другой программной секции, вызывает печать в листинге сообщения об ошибке «P». В случае изменения на втором проходе трансляции значения метки будет напечатано сообщение об ошибке «P».

7.2.2. Директивы .CROSS и .NOCROSS. Директивы .CROSS и .NOCROSS управляют печатью и содержанием таблицы перекрестных ссылок: .CROSS — разрешить указанное действие, .NOCROSS — запретить указанное действие.

Формат:

.CROSS [S1, S2, ..., SN]

.NOCROSS [S1, S2, ..., SN]

где S1,S2, ..., SN — символические имена, разделенные запятыми, пробелами или символами ГТ.

Директивы используются с переключателем командной строки /C[R] или /CROSS. По умолчанию таблица перекрестных ссылок содержит все определения и обращения ко всем символическим именам в модуле.

Директива .NOCROSS (эквивалентна .DSABL.CRF) без аргументов запрещает включение символических имен в таблицу перекрестных ссылок до появления директивы .CROSS без аргументов.

Директива .NOCROSS со списком аргументов запрещает включение указанных имен в таблицу перекрестных ссылок. Директива .CROSS со списком аргументов разрешает включение указанных имен в таблицу перекрестных ссылок.

Если таблица перекрестных ссылок всех символов в модуле запрещена по директиве .NOCROSS без аргументов, директива .CROSS со списком аргументов не будет иметь действие до повторного разрешения таблицы перекрестных ссылок по директиве .CROSS без аргументов.

Директива .CROSS без списка аргументов эквивалентна директиве .ENABL CRF, а директива .NOCROSS без списка аргументов эквивалентна директиве .DSABL CRF.

Примеры:

```
1)          .NOCROSS
LABEL1:  MOV LOC1,LOC2
            .CROSS
```

В данном примере имя LABEL1 и ссылка на LOC1 и LOC2 не включены в таблицу перекрестных ссылок.

```
2)      .NOCROSS
LABEL2: MOV LOC1,LOC2
        .CROSS LOC1
```

В данном примере определение и ссылка на LOC2 включены в таблицу перекрестных ссылок, но ссылка на LOC1 не включена в таблицу перекрестных ссылок.

7.3. Директивы задания данных. Директивы задания данных используются для записи данных в различных видах.

К директивам задания данных относятся: .BYTE .WORD .ASCII .ASCIZ .RAD50 .PACKED .FLT2 .FLT4.

7.3.1. Директива .BYTE. Директива .BYTE используется для записи данных (операндов) в двоичном виде в последовательно расположенных байтах объектного модуля.

Формат:

.BYTE [E1,...,EN]

где E1,...,EN — одно или несколько допустимых выражений, разделенных запятыми.

Каждое выражение имеет 8-разрядное значение.

Операнды директивы .BYTE вычисляются сначала как выражения длиной в одно слово, а затем усекаются до восьми младших двоичных разрядов. Шестнадцатиразрядное значение указанного в директиве выражения должно иметь в старшем байте (который отбрасывается) либо все нули, либо все единицы, в противном случае в листинге печатается сообщение об ошибке «Т».

Возможно, во время связывания модулей значение перемещаемого выражения превысит восемь двоичных разрядов. В этом случае редактор связей печатает сообщение об ошибке.

Директива .BYTE без аргументов или директива .BYTE, за которой следуют запятые, записывает нули в последовательные байты. **Примеры:**

```
1)      SAM=5
.=410
        .BYTE ^D48,SAM      ;060 (восьмеричный код деся-
                            ;тичного 48) запоминается в
                            ;ячейке 410
        .PSECT              ;перемещаемая программная
A:
        .BYTE A             ;секция A имеет относитель-
                            ;ное значение
2) .=420
        .BYTE ,,,           ;значение 0 запоминается в
                            ;ячейках 420, 421, 422, 423
```

7.3.2. Директива .WORD. Директива .WORD используется для записи данных (операндов) в последовательно расположенных словах объектного модуля.

Формат:

.WORD [E1,..:,EN]

где E1,...,EN — одно или несколько допустимых выражений, разделенных запятыми.

Каждое выражение имеет 16-разрядное значение.

Директива .WORD без аргументов или директива .WORD, за которой следуют запятые, записывает нули в последовательные слова.

Примеры:

```
1)      SAL=0
        .=500
        .WORD  177535,.+4,SAL ; значения 177535, 506 и 0
                              ; запоминаются в ячейках
                              ; 500, 502 и 504 соответст-
                              ; венно
2)  .=500
        .WORD ,5,             ; значения 0, 5 и 0 запо-
                              ; минаются в ячейках 500,
                              ; 502 и 504 соответственно
```

7.3.3. Директива .ASCII. Директива .ASCII используется для записи последовательности символов в коде КОИ—7.

Формат:

.ASCII /C1/.../CN/

где C1,...,CN — последовательности символов КОИ-7;
/ / — ограничители (любые символы, за исключением символов, указанных в операнде директивы).

Непечатные символы могут записываться в операнде директивы .ASCII только путем ограничения кода каждого символа «<» и «>». Символы «<» и «>», используемые внутри последовательности символов, не являются ограничителями.

Пример:

```
.ASCII /адрес/          ; коды символов а, д, р, е, с за-
                        ; поминаются в последовательных
                        ; байтах
.ASCII                  ; коды символов A,B,C,ВК,ПС,D,
/ABC/<15><12>/DEF/      ; E,F запоминаются в последова-
                        ; тельных байтах
.ASCII /A<15>B/         ; коды символов A, <, 1, 5, >, B
                        ; запоминаются в последователь-
                        ; ных байтах
```

```
.ASCII <15>/ABC/          ; код 15 и коды символов A, B, C
                          ; запоминаются в последователь-
                          ; ных байтах
```

Следует соблюдать осторожность при использовании символов «;», «<» и «=» в качестве ограничителей в директиве .ASCII, так как эти символы имеют специальное назначение.

Пример:

```
ASCII ;ABC;/DEF/          ; коды символов A, B, C, D, E, F за-
                          ; поминаются в последовательных
                          ; байтах
.ASCII /ABC/;DEF;         ; коды символов A,B,C запоминают-
                          ; ся в последовательных байтах,
                          ; символы D, E, F, ; рассматрива-
                          ; ются как комментарий
ASCII /ABC/=DEF=          ; коды символов A, B, C, D, E, F за-
                          ; поминаются в последовательных
                          ; байтах
ASCII =DEF=               ; воспринимается как оператор пря-
                          ; мого присваивания .ASCII=DEF
```

7.3.4. Директива .ASCIZ. Директива .ASCIZ эквивалентна директиве .ASCII за исключением того, что автоматически добавляется нулевой байт как последний символ операнда директивы. Нулевой байт определяет конец последовательности символов.

Формат:

```
ASCIZ /C1/.../CN/
```

где C1,...,CN — то же, что в п. 7.3.3;
/ / — то же, что в п. 7.3.3.

Пример:

```
          CR=15
          LF=12
HELLO:    .ASCIZ <CR><LF>/сообщение/<CR><LF>
          .EVEN
                .
                .
                .
          MOV #HELLO,R1
          MOV #LINBUF,R2
10¤:      MOVB (R1)+,(R2)+
          BNE 10¤
                .
                .
```

7.3.5. Директива .RAD50. Директива .RAD50 используется для упаковки последовательности символов в код RADIX-50. Каждые три символа упаковываются в одно слово.

Формат.

.RAD50 /C1/.../CN/

где C1,...CN — упаковываемые символы, допустимые в RADIX-50 (табл. 9);

/ / — ограничители (любые символы, за исключением символов, указанных в операнде директивы).

Если в одно слово упаковывается менее чем три символа, то недостающие символы дополняются пробелами.

В директиве .RAD50 для записи кодов символов должны использоваться символы «<» и «>».

Таблица 9

Символ	Код RADIX—50
1	2
Пробел	0
A—Z	1—32
¤	33
.	34
Не определен	35
0—9	36—47

Код RADIX-50 для трех последовательно расположенных символов C1, C2, C3 определяется по формуле:

код RADIX-50 = ((C1*50(8))+C2)*50(8)+C3

Например, последовательность символов ABC в коде RADIX-50 имеет значение:

((1*50(8))+2)*50(8)+3=3223(8)

Примеры:

```
.RAD50 /ABC/        ; упаковка ABC в одно слово
.RAD50 /AB/         ; упаковка AB и пробела в одно слово
 RAD50 /ABCD/       ; упаковка ABC в первое слово, D и двух
                    ; пробелов во второе слово
.RAD50 /AB/<35>     ; упаковка 3255 в одно слово
CHR1=1
CHR2=2
CHR3=3
      .
      .
      .
```

.RAD50 , эквивалентно RAD50 /ABC/
<CHR1><CHR2><CHR3>

7.3.6. Директива .PACKED. Директива PACKED используется для упаковки десятичных данных по две цифры в байт

Формат
PACKED DS [,C]

где DS — последовательность десятичных цифр (до 31);
C — символическое имя, принимающее значение, равное числу цифр в заданной последовательности

Последовательность десятичных цифр может иметь знак, но не должна использоваться как число Каждая цифра в заданной последовательности имеет значение от 0 до 9

Примеры:

PACKED —12,PACK	PACK получает значение 2
PACKED +500	, упаковать 500
PACKED 0	, упаковать 0
PACKED —0,SUM	SUM получает значение 1
PACKED 1234E6	недопустимая упаковка E6 будет , рассматриваться как переменная и , принимать значение 4

7 3 7. Директивы .FLT2 и .FLT4. Директивы FLT2 и FLT4 используются для записи десятичных чисел в форме с плавающей запятой

Формат
FLT2 A1, ,A
FLT4 A1, ,A

где A1, ,AN — одно или несколько десятичных чисел, разделенных запятои

По директиве FLT2 каждое число записывается с одинарной точностью — в два слова (рис 1), по директиве FLT4 — с двойной точностью — в четыре слова (рис 2) Формат числа с плавающей запятои, занимающего два слова

31 30 23 22 0

Порядок	Мантисса

ЗНАК ЧИСЛА

Рис 1

Формат числа с плавающей запятой, занимающего четыре слова

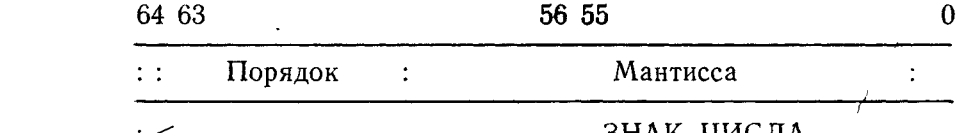


Рис. 2

7.4. Директива управления системой счисления: .RADIX. Константы и значения выражений обрабатываются АССЕМБЛЕРом в восьмеричной системе счисления.

Директива .RADIX позволяет установить десятичное, восьмеричное или двоичное основание системы счисления для данных всей исходной программы или ее части.

Формат:

.RADIX [N]

где N — выражение, задающее основание системы счисления (2,8,10). Оно вычисляется в десятичной системе счисления, независимо от ранее установленного основания.

Если аргумент в директиве не задан, то по умолчанию предполагается N=8. Директива .RADIX сохраняет действие до появления другой директивы .RADIX.

В макроопределениях для задания системы счисления отдельных данных рекомендуется использовать специальные операции (см. п. 4.3.1).

Пример:

```
.RADIX 10 ; устанавливается десятичная система счисления
.RADIX    ; устанавливается восьмеричная система счисления.
```

7.5. Директивы управления счетчиком. Директивы управления счетчиком используются для управления значением счетчика текущего адреса и для резервирования области памяти в объектном модуле.

К директивам управления счетчиком относятся: .EVEN .ODD .BLKB .BLKW .LIMIT

7.5.1. Директива .EVEN. Директива .EVEN используется для установления четного значения счетчика адреса путем добавления единицы в случае его нечетности.

Формат:

.EVEN

Директива .EVEN применяется обычно после использования директив АССЕМБЛЕРа, приводящих к нечетному значению счетчика адреса: .ASCII, .ASCIZ, .BYTE.

Если текущее значение счетчика адреса четное, то директива .EVEN не выполняет никакого действия.

Пример:

```
.ASCII /текст/
.EVEN
.WORD XYZ
```

7.5.2. Директива .ODD. Директива .ODD используется для установления нечетного значения счетчика адреса путем добавления единицы в случае его четности.

Формат:

```
.ODD
```

Если текущее значение счетчика адреса нечетно, то директива .ODD не выполняет никакого действия.

7.5.3. Директивы .BLKB и .BLKW. Директивы .BLKB и .BLKW используются для резервирования области памяти в объектном модуле: по директиве .BLKB резервируется заданное число байтов, по директиве .BLKW резервируется заданное число слов.

Формат:

```
.BLKB E
.BLKW E
```

где E — выражение, имеющее абсолютное значение и определяющее число байтов или слов, резервируемых в объектном модуле. По умолчанию E=1.

Пример:

```
 1
 2
 3
 4 000000
 5
 6 000000    COUNT:  .BLKW 1  ;резервировать 1 слово
 7
 8 000002    MESSAG: .BLKW 80.;буфер для текста сообще-
 9                            ;ния
10 000122    CHRSAV: .BLKB    ;резервировать 1 байт
11
12 000123    FLAG:   .BLKB
13
14 000124    MSGPTR: .BLKB
```

ПРИМЕЧАНИЕ. Действие директивы .BLKB аналогично действию оператора прямого присваивания:

. = выражение

По оператору прямого присваивания значение выражения приоавляется к текущему значению счетчика адреса. Однако, для увеличения значения счетчика адреса предпочтительно использовать директиву .BLKB.

7.5.4. Директива .LIMIT. Директива .LIMIT используется для резервирования двух слов памяти, первое из которых предназначено для записи младшего адреса загрузочного модуля, второе — для записи адреса первого свободного слова, следующего за загрузочным модулем (старший адрес загрузочного модуля + 2).

Формат:

.LIMIT

Запись значений в зарезервированные слова выполняется во время связывания модулей по программе «Редактор связей».

Действие директивы .LIMIT аналогично действию директивы .BLKW 2.

7.6. Директива окончания: .END. Директива .END используется для указания логического конца исходного модуля.

Формат:

.END [E]

где E — выражение, определяющее точку входа программы.

Предложения, расположенные за директивой .END в текущей исходной программе, игнорируются.

Если исходная программа состоит из нескольких модулей, каждый из которых транслируется отдельно, то только один модуль должен оканчиваться директивой .END с аргументом, а все остальные — директивой .END без аргумента.

Директива .END не должна использоваться в макроопределениях и блоках условной трансляции, но может использоваться в директиве непосредственной условной трансляции (п. 7.9.3).

7.7. Директивы секционирования. Директивы секционирования дают возможность управлять распределением памяти для программы во время ее связывания, так как все признаки, введенные с помощью директив секционирования, передаются редактору связей.

К директивам секционирования относятся: .PSECT .ASECT .CSECT .SAVE .RESTORE.

7.7.1. Директива .PSECT. Директива .PSECT позволяет пользователю создавать программные секции и распределять команды и данные между программными секциями.

Формат:

.PSECT S,A1,...,AN

где S — символическое имя программной секции (табл. 10); A1,...,AN — один или несколько аргументов, разделенных запятыми, пробелами или символами ГТ (табл. 10).

Программная секция — это блок программы, который может быть отредактирован программой «Редактор связей» независимо от других блоков программы. Программная секция обычно начинается с директивы .PSECT, .CSECT или .ASECT.

Редактор связей использует признаки программной секции, задаваемые с помощью аргументов директивы .PSECT.

Таблица 10

Аргумент	Действие по умолчанию	Значение
1	2	3
S	Нет	Устанавливает имя программной секции, состоящее не более чем из шести символов, допустимых в коде RADIX-50. Имена программных секций могут совпадать с внутренними именами. Если имя отсутствует, то следующему аргументу должна предшествовать запятая
RO/RW	RW	Определяет признак доступа к данным программной секции: RO — разрешается только чтение; RW — разрешается чтение и запись
I D	I	Определяет признак содержания программной секции: I — программная секция содержит команды; D — программная секция содержит данные
GBL/LCL	LCL	Определяет признак размещения программной секции: GBL — глобальная секция. Такая секция компонуется из модулей всей программы. Если глобальная секция размещена в модулях, входящих в разные сегменты, то редактор связей помещает ее в сегмент с меньшим номером; LCL — локальная секция. Для каждого сегмента эта секция компонуется из модулей, входящих в данный сегмент. Аргументы GBL/LCL используются только для оверлейных программ При построении односегментных неоверлейных программ аргументы GBL/LCL не имеют смысла, так как полное распределение памяти для этой программы осуществляется внутри корневого сегмента
	—	Определяет область размещения программной секции: SAV — программная секция размещается в корневом сегменте;

1	2	3
ABS/REL	REL	Определяет признак перемещаемости программной секции: ABS — абсолютная секция. Размещение данных в абсолютной секции должно быть осуществлено внутри сегмента, содержащего программную секцию. Базовый адрес программной секции не изменяется, он равен нулю; REL — перемещаемая секция. Базовый адрес программной секции зависит от перемещения программной секции во время связывания
CON/OVR	CON	Определяет признак размещения программной секции: CON — все части программной секции редактор связей размещает в загрузочном модуле последовательно друг за другом. Размер отводимой области памяти равен сумме размеров всех частей данной секции; OVR — адрес загрузки для всех частей данной секции, находящихся в разных модулях, будет один и тот же. Размер отводимой области памяти равен размеру наибольшей части этой секции

В табл. 10 символ «/» в графе «Аргумент» означает, что должен быть использован один из данных аргументов. Если в директиве .PSECT указывается аргумент, отличный от перечисленных в табл. 10, то в листинге печатается сообщение об ошибке «А».

Признаки, определенные с помощью директивы .PSECT, действительны для всех последующих директив .PSECT, которые встречаются в данной программе с тем же именем.

АССЕМБЛЕР разрешает создание 256. программных секций:

— одну абсолютную секцию .ABS., определяемую по умолчанию;

— одну неименованную перемещаемую программную секцию, определяемую по умолчанию;

— 254. именованных программных секций.

Для каждой программной секции АССЕМБЛЕР ведет следующую информацию:

— имя программной секции;

— содержимое счетчика текущего адреса;

— максимальное встретившееся значение счетчика адреса;

— признаки программной секции, т. е. аргументы директивы .PSECT.

7.7.1.1. Создание программных секций

АССЕМБЛЕР начинает трансляцию исходных предложе-

ний с относительного нулевого адреса неименованной программной секции. Таким образом, первым предложением исходной программы всегда является подразумеваемая директива .PSECT.

Первое появление директивы .PSECT с заданным именем предполагает, что счетчик текущего адреса устанавливается на относительное нулевое значение. Действие этой директивы распространяется до появления другой директивы секционирования. Последующее появление директивы .PSECT с уже встречающимся именем возобновляет трансляцию программы с места окончания соответствующей программной секции.

Пример:

```
    .PSECT          ;неименованная программная секция,
A:  .WORD 0         ;транслируемая по относительным адре-
B:  .WORD 0         ;сам 0, 2 и 4
    .PSECT ALPHA    ;именованная программная секция,
X:  .WORD 0         ;транслируемая по относительным адре-
Y:  .WORD 0         ;сам 0 и 2
    .PSECT          ;продолжение неименованной програм-
D:  .WORD 0         ;мной секции, транслируемой по относи-
                    ;тельному адресу 6
```

Программная секция описывается полностью первой директивой .PSECT. После этого на секцию может быть сделана ссылка с указанием только ее имени.

Например, программная секция может быть описана директивой .PSECT ALPHA,ABS,OVR и в последующем на нее может быть сделана ссылка с помощью директивы .PSECT ALPHA.

Если директива секционирования не задана, то АССЕМБЛЕР предполагает заданной директиву .PSECT.

Использование отдельных счетчиков адреса для каждой программной секции дает возможность записывать предложения, которые не являются физически последовательными внутри программы. Данные предложения могут быть загружены последовательно после трансляции.

Пример:

```
    .PSECT SEC1,REL,R0  ;начинается перемещаемая про-
A:  .WORD 0             ;граммная секция, транслируе-
B:  .WORD 0             ;мая по относит. адресам 0, 2 и 4
C:  .WORD 0             ;коды транслируются по относи-
ST: CLR A               ;тельным адресам с 6 по 12
      CLR B
```

```
.PSECT SECA,ABS      ; начинается абсолютная програм-
.WORD .+2,A          ; мная секция SECA. Коды транс-
                     ; лируются по абсолютным адре-
                     ; сам 0 и 2
.PSECT SEC1          ; возобновляется перемещаемая
INC A                ; программная секция SEC1. Ко-
BR T                 ; ды транслируются по относи-
                     ; тельным адресам 14 и 16
```

Имя счетчика адреса (.) является относительным или абсолютным в зависимости от того, в какой программной секции оно используется. Все метки в абсолютной секции являются абсолютными. Все метки в перемещаемой секции являются относительными.

Меткам, появляющимся в строке с директивой .ASECT, .CSECT, .PSECT, присваивается текущее значение счетчика адреса предыдущей секции.

Пример:

Если первая строка программы есть A: .PSECT ALT,REL, то метке A присваивается значение относительного нуля.

7.7.1.2. Распределение памяти

АССЕМБЛЕР не фиксирует ошибку, если модуль заканчивается нечетным адресом. Это позволяет поместить нечетное число данных в конце модуля. Однако, если несколько модулей содержат части одной и той же программной секции с аргументом CON, то модули нечетной длины (кроме последнего) могут привести к тому, что редактор связей свяжет модули, начиная с нечетных адресов, в результате чего получится невыполнимая программа. Чтобы избежать этого, команды и данные следует поместить в разные именованные программные секции. Это позволит редактору связей начинать каждую программную секцию с четного адреса.

7.7.2. Директивы .ASECT и .CSECT. Директива .ASECT используется для определения абсолютной программной секции, директива .CSECT — именованной или неименованной перемещаемой программной секции. Эти директивы сохранены для совместимости с предыдущими версиями АССЕМБЛЕРа.

Формат:

.ASECT

.CSECT [S]

где S — имя программной секции.

АССЕМБЛЕР транслирует директивы .ASECT и .CSECT

как директиву .PSECT с признаками, определяемыми по умолчанию (табл. 11).

Таблица 11

Директива	Значение признака директивы .PSECT					
1	2					
	Имя	Доступ	Тип	Размещение	Перемещение	Распределение
.ASECT	.ABS	RW	I	GBL	ABS	OVR
.CSECT	—	RW	I	LCL	REL	CON
.CSECT S	S	RW	I	GBL	REL	OVR

7.7.3. Директива .SAVE. Директива .SAVE используется для сохранения текущего контекста программной секции в верхней части стека. При этом текущий контекст программной секции остается в действии. Контекст программной секции включает текущее значение счетчика адреса и имя секции.

Формат:

.SAVE

Допускается использование до 16 директив .SAVE. Если стек полон, печатается сообщение об ошибке «A».

7.7.4. Директива .RESTORE. Директива .RESTORE используется для восстановления из стека контекста программной секции.

Формат:

.RESTORE

По директиве .RESTORE текущее значение счетчика адреса и имя секции устанавливаются в значения, которые они имели к моменту выполнения директивы .SAVE.

Если при выполнении директивы .RESTORE стек пуст, печатается сообщение об ошибке «A».

Пример:

В данном примере показано использование директив SAVE и .RESTORE.

```
1                       .MACRO DS NAME,SIZE
2                       .SAVE
3                       .PSECT IMPURE,D,GBL
4                 NAME: .BLKW SIZE
5                       .RESTORE
```

```
 6                                          .ENDM
 7
 8  000000 016701 000000'    SCANSY:MOV SYMBAS,R1
 9  000004 010167 000002'           MOV R1,CURSYM
10  000010 066701 000004'           ADD SYMSIZ,R1
11  000014 010167 000006'           MOV R1,SYMTOP
12
13  000020 000207                   RETURN
14
15  000022                          DS SYMBAS
16  000022                          DS CURSYM
17  000022                          DS SYMSIZ
18  000022                          DS SYMSIZ
19  000022 016701 000006'     SSORT:MOV SYMTOP,R1
20         000001                   .END
```

7.8. Директивы описания имен. Директивы описания имен используются для описания имен как глобальных и внешних.

К директивам описания имен относятся: .GLOBL .WEAK.

7.8.1. Директива .GLOBL. Директива .GLOBL используется для описания имен как глобальных. Глобальные имена используются для организации связи между модулями.

Формат:

.GLOBL S1,S2,...,SN

где S1,S2,...,SN — символические имена, разделенные запятыми, пробелами или символами ГТ.

Директива .GLOBL A,B эквивалентна по своему действию следующим предложениям:

A==:выражение
B==:выражение или
A==выражение
B==выражение или
A::
B::

В конце первого прохода АССЕМБЛЕР проверяет, определено ли данное глобальное имя внутри текущего программного модуля. Если имя не определено в текущем модуле, то оно рассматривается как внешнее имя. Все внешние имена, появляющиеся внутри данной программы, должны быть определены в конце первого прохода, в противном случае они рассматриваются как глобальные ссылки по умолчанию. Описание разрешения/запрещения глобальных ссылок приведено в п. 7.2.1.

Пример подпрограммы с двумя точками входа. Подпрограмма вызывает другую подпрограмму.

```
        .PSECT
        .GLOBL A,C
A:      MOV @(R5)+,R0
        MOV #X,R1
X:      JSR PC,C
        RTS R5
B: :    MOV (R5)+,R1
        CLR R2
        BR X
```

7.8.2. Директива .WEAK. Директива .WEAK используется для описания имен как внешних. Поиск указанных имен в объектной библиотеке запрещается.

Формат:

.WEAK S1,...,SN

где S1,...,SN — символические имена, разделенные запятыми, пробелами или символами ГТ.

Имя, определенное как внешнее по директиве .WEAK, является глобальным. Если редактор связей находит определение имени в другом модуле, то используется найденное определение, в противном случае имени присваивается нулевое значение.

Пример:

```
.WEAK SUB1,SUB2
```

7.9. Директивы условной трансляции. Директивы условной трансляции используются для получения разных вариантов объектной программы из одной исходной программы.

К директивам условной трансляции относятся: .IF .ENDC .IFF .IFT .IFTF .IIF.

7.9.1. Директивы .IF и .ENDC. Директивы .IF и .ENDC используются для создания блока условной трансляции, который позволяет во время трансляции включать или не включать в объектный модуль блоки программы в зависимости от выполнения заданного условия.

Формат блока условной трансляции:

```
.IF условие, A1[,A2,...,AN]
.
.
.
Блок
.
```

.ENDC

где .IF — директива начала блока условной трансляции;
условие — условие согласно табл. 12;
, — разделительный символ: запятая, пробел или символ ГТ;
A1,[A2,...,AN] — символические аргументы и/или выражения для проверки указанных условий;
блок — блок исходной программы;
.ENDC — директива окончания блока условной трансляции.

Блок программы транслируется, если условие выполняется, и не транслируется в противном случае.

Условия, отличные от перечисленных в табл. 12, недопустимый аргумент или отсутствие аргументов в директиве .IF вызывает печать в листинге сообщения об ошибке «А».

Таблица 12

Условие	Аргумент	Блок транслируется, если
1	2	3
EQ/NE	Выражение	выражение равно нулю / не равно нулю
GT/LE	»	выражение больше нуля / меньше или равно нулю
LT/GE	»	выражение меньше нуля / больше или равно нулю
DF/NDF	Символическое имя	имя определено / не определено
B/NB	Макропараметр	параметр есть / параметра нет
IND/DIF	Два макропараметра	параметры одинаковы / различны

ПРИМЕЧАНИЕ. В табл. 12 символ «/» в графе «Условие» означает, что должно быть использовано одно из данных условий.

Параметр макрокоманды, который используется как аргумент директивы условной трансляции, должен заключаться в символы «<» и «>» или отмечаться символом «^».

Например,
<A,B,C>
^/124/

При проверке условий DF и NDF символические аргументы могут группироваться с помощью логических операций «&» и «!» (см. табл. 4).

Пример:

```
.IF DF SYM1&SYM2
     .
     .
     .
.ENDC
```

В данном примере блок транслируется, если определены два аргумента SYM1 и SYM2.

АССЕМБЛЕР допускает вложение директив условной трансляции на глубину 16.. Каждый блок условной трансляции должен заканчиваться директивой .ENDC. Внутренние директивы условной трансляции игнорируются, если внешнее условие не выполнено.

Формат вложенных директив условной трансляции:

```
.IF условие, A1
.IF условие, A1
     .
     .
     .
.ENDC
.ENDC
```

Использование директивы .ENDC вне блока условной трансляции или попытка превзойти допустимую глубину уровней вложения вызывает печать в листинге сообщения об ошибке «О».

Пример:

```
.IF DF SYM1
.IF DF SYM2
     .
     .
     .
.ENDC
.ENDC
```

7.9.2. Директивы .IFF, .IFT, .IFTF. Директивы .IFF, .IFT, .IFTF используются только внутри блока условной трансляции. Назначение директив приведено в табл. 13. Часть программы, следующая за одной из данных директив до следующей директивы условной трансляции или до конца блока условной трансляции, транслируется или нет в зависимости от результата проверки условия в директиве .IF.

Таблица 13

НАЗНАЧЕНИЕ ДИРЕКТИВ

Директива	Назначение
1	2
.IFF	Блок транслируется, если условие не выполнено
.IFT	Блок транслируется, если условие выполнено
.IFTF	Блок транслируется независимо от значения условия

Использование директив условной трансляции .IFF, .IFT, .IFTF вне блока условной трансляции вызывает печать в листинге сообщения об ошибке «О».

Если для вложенных директив условной трансляции не выполняется внешнее условие, то внутренние директивы условной трансляции игнорируются.

Пример 1:

Предполагается, что имя SYM определено.

```
.IF DF SYM          ; условие выполнено, транслировать блок
    .
    .
    .
.IFF                ; условие выполнено, не транслировать
    .               ; блок.
    .
    .
.IFT                ; условие выполнено, транслировать блок
    .
    .
    .
.IFTF               ; транслировать блок независимо от вы-
    .               ; полнения условия
    .
    .
.IFT                ; условие выполнено, транслировать блок
    .
    .
    .
.ENDC
```

Пример 2:

Предполагается, что имя X определено, а Y — не определено.

```
.IF DF X            ; условие выполнено
.IF DF Y            ; условие не выполнено
.IFF                ; условие не выполнено, транслировать
    .               ; блок
    .
    .
.IFT                ; условие не выполнено, не транслировать
    .               ; блок
    .
    .
.ENDC
.ENDC
```

Пример 3:

Предполагается, что имя A определено, а имя B — не определено.

```
.IF DF A            ; условие выполнено, транслировать блок
MOV A,R1
    .
    .
    .
.IFF                ; условие выполнено, не транслировать
                    ; блок
MOV R1,R0
    .
    .
    .
.IF NDF B           ; директива условной трансляции игнори-
    .               ; руется
    .
    .
.ENDC
.ENDC
```

Пример 4:

Предполагается, что X — определено, а Y — не определено.

```
.IF DF X            ; условие не выполнено, не транслировать
    .               ; блок
    .
    .
.IF DF Y            ; директива условной трансляции игнори-
    .               ; руется
    .
    .
```

```
.IFT                    ; директива игнорируется
    .
    .
    .
.IFT                    ; директива игнорируется
    .
    .
    .
.ENDC
.ENDC
```

7.9.3. Директива .IIF. Директива .IIF является директивой непосредственной условной трансляции. Блок условной трансляции содержит одно предложение и находится на одной строке с директивой. При использовании данной директивы не требуется директива окончания блока условной трансляции .ENDC.

Формат:

.IIF условие,A1,предложение

где условие — условие согласно табл. 12;

, — разделительный символ: запятая, пробел или символ ГТ. Если в качестве аргумента используется выражение, то разделительным символом должна быть запятая;

A1 — аргумент директивы непосредственной условной трансляции (см. табл. 12);

Предложение — предложение исходной программы, которое транслируется, если условие выполнено.

Пример:

.IIF DF FOO,BEQ ALPHA

Предложение BEQ ALPHA транслируется, если имя FOO определено в исходной программе.

7.10. Директивы управления файлами. Директивы управления файлами используются для указания имени макробиблиотеки, просматриваемой во время трансляции, и для ввода исходного файла в транслируемый исходный файл.

7.10.1. Директива .LIBRARY. Директива .LIBRARY используется для включения имени файла макробиблиотеки пользователя в список просматриваемых АССЕМБЛЕРом макробиблиотек.

Формат:

.LIBRARY /C/

где C — спецификация файла макробиблиотеки;

/ / — ограничители (любые символы, за исключением символов, используемых в спецификации файла).

Макробиблиотеки, включенные в список, просматриваются АССЕМБЛЕРом в случае задания в исходной программе директив .MCALL или .ENABL MCL или появления неопределенного символического имени.

Макробиблиотеки, включенные в список просматриваемых библиотек, просматриваются в порядке, обратном их заданию. Если спецификация файла макробиблиотеки задана не полностью, по умолчанию принимаются следующие значения: имя устройства — DK: тип файла — .MLB.

ПРИМЕЧАНИЯ:

1. Драйвер устройства, на котором находится файл макробиблиотеки, должен быть резидентным.

2. Максимальное число файлов макробиблиотеки определяется числом 12 минус N, где N — число одновременно транслируемых файлов.

3. Директива .LIBRARY записывается в исходной программе до появления первой описанной в ней макрокоманды.

Примеры:

```
.LIBRARY /DK:USLIB.MLB/
.LIBRARY ?DK:SYSDEF.MLB?
.LIBRARY \CURRENT.MLB\
```

7.10.2. Директива .INCLUDE. Директива .INCLUDE используется для ввода исходного файла в транслируемый исходный файл.

Формат:

.INCLUDE /C/

где C — спецификация вводимого исходного файла;
/ / — ограничители (любые символы, за исключением символов, используемых в спецификации файла).

При появлении директивы .INCLUDE текущий файл записывается в стек и транслируется вводимый файл. При достижении конца вводимого файла текущий файл удаляется из стека и трансляция продолжается со строки, следующей за директивой.

Для вложенных директив .INCLUDE текущий файл и первый вводимый файл записываются в стек и транслируется второй вводимый файл. При достижении конца второго вводимого файла первый вводимый файл удаляется из стека и его трансляция возобновляется со строки, следующей за директивой. При достижении конца первого вводимого файла текущий файл удаляется из стека и его трансляция возобновляется со строки, следующей за директивой .INCLUDE.

Максимальный уровень вложения исходных файлов для директив .INCLUDE равен 5.

Если спецификация вводимого файла задана не полностью, по умолчанию принимаются следующие значения: имя устройства — DK: тип файла — .MAC.

ПРИМЕЧАНИЕ. При использовании директивы .INCLUDE драйвер устройства, на котором находится вводимый файл, должен быть резидентным.

Примеры:

```
.INCLUDE /DR3:[1,2]MACROS/
.INCLUDE ?DK:SYSDEF?
.INCLUDE \CURRENT.MAC\
```

8. МАКРОСРЕДСТВА ЯЗЫКА АССЕМБЛЕРа

При составлении программ на языке АССЕМБЛЕР часто возникает необходимость использовать некоторую последовательность предложений в программе несколько раз. Макросредства языка АССЕМБЛЕРа позволяют не переписывать каждый раз такую последовательность предложений, а вызывать ее в программу с помощью одного предложения — макрокоманды. При этом можно изменить некоторые предложения этой последовательности и вызывать не все предложения, а только некоторые из них. Таким образом, макросредства позволяют:

— вставлять в исходный модуль последовательность предложений;

— изменять порядок следования предложений;

— изменять отдельные части предложений.

К макросредствам языка АССЕМБЛЕР относятся макроопределения, макрокоманды и директивы определения характеристик макропараметров.

8.1. Макроопределение. Макроопределение — это последовательность предложений языка АССЕМБЛЕР, которая может быть неоднократно включена в исходную программу с помощью макрокоманды.

Макроопределение состоит из директивы указания начала макроопределения, последовательности предложений, составляющих собственно макроопределение, и директивы указания конца макроопределения.

Макроопределение должно быть задано в исходной программе раньше, чем появится макрокоманда, вызывающая данное макроопределение.

Одно и то же макроопределение может быть использовано в нескольких программах. Для этого следует поместить макроопределение в макробиблиотеку пользователя или систем-

ную макробиблиотеку. Макроопределение хранится в макробиблиотеке с именем, которое указано в директиве .MACRO.

Параметры в макроопределении называются формальными параметрами, а в макрокомандах — фактическими.

8.1.1. Директива .MACRO. Директива .MACRO используется для указания начала макроопределения.

Формат:

.MACRO S,A1,A2,...,AN

где S — символическое имя, называемое именем макроопределения;

, — допустимый разделительный символ: запятая, пробел или символ ГТ;

A1,A2,...,AN — список формальных параметров, представляющих собой допустимые символические имена.

Параметры отделяются друг от друга разделительными символами: запятой, пробелом или символом ГТ. За параметрами может следовать комментарий.

Имя макроопределения и символические имена, использующиеся в качестве формальных параметров, могут совпадать с метками в исходной программе, но не должны совпадать с метками в макроопределении.

ПРИМЕЧАНИЕ. В директиве .MACRO разрешается использование метки, но вводить ее не рекомендуется, особенно для вложенных макроопределений, так как неправильные метки и метки, создаваемые с символом конкатенации, приводят к игнорированию данной директивы. Данное ПРИМЕЧАНИЕ относится также к директивам .IRP, .IRPC, .REPT.

8.1.2. Директива .ENDM. Директива .ENDM используется для указания конца макроопределения.

Формат:

.ENDM [S]

где S — имя макроопределения.

Указание имени в директиве .ENDM дает возможность транслятору обнаружить пропущенные директивы .ENDM и макроопределения с неправильными вложениями.

Имя S в директиве .ENDM должно совпадать с именем, указанным в соответствующей директиве .MACRO.

За директивой .ENDM может следовать комментарий.

Директива .ENDM, используемая вне макроопределения, вызывает печать сообщения об ошибке «O».

Если имя макроопределения и имя, указанное в директиве .ENDM, не совпадают, то в листинге печатается сообщение об ошибке «A».

ПРИМЕЧАНИЯ:

1. В директиве .ENDM метка игнорируется.

2. Директива .ENDM с недопустимой меткой игнорируется.

Пример:

```
.MACRO TYPMSG MESSAGE          ;напечатать сообщение
JSR R5,TYPMSG
.WORD MESSAGE
.ENDM
```

8.1.3. Директива .MEXIT. Директива .MEXIT используется в макроопределении для завершения макрорасширения до того, как встретится директива окончания макроопределения .ENDM.

Формат:

.MEXIT

Директива .MEXIT используется в блоках повторений, вложенных макроопределениях и блоках директив условной трансляции.

Пример:

```
.MACRO ALTR N,A,B      ;начало макроопределения
    .
    .
    .
.IF EQ N
    .
    .
    .
.MEXIT
.ENDC
    .
    .
    .
.ENDM
```

Если в макрокоманде ALTR фактический параметр, соответствующий формальному параметру N макроопределения, равен нулю, то блок условной трансляции транслируется и директива .MEXIT будет завершать и блок условной трансляции и макрорасширение. Для вложенных макрокоманд директива .MEXIT осуществляет переход к макрокоманде внешнего уровня.

Директива .MEXIT, используемая вне макроопределения, вызывает печать в листинге сообщения об ошибке «О».

8.1.4. Форматирование макроопределений

Символ ПФ (перевод формата), используемый в макроопределении, вызывает переход на новую страницу листинга во время трансляции макроопределения. Однако, переход на новую страницу не осуществляется при расширении макрокоманды.

Если в макроопределении указывается директива .PAGE, то она игнорируется во время трансляции макроопределения, но при расширении данной макрокоманды осуществляется переход на новую страницу.

8.2. **Макрокоманда.** Макрокоманда — предложение на языке АССЕМБЛЕР, которое во время трансляции заменяется макрорасширением.

Макрорасширение — макроопределение, в котором формальные параметры заменены на соответствующие им фактические параметры.

Формат макрокоманды:

S фактические параметры

где S — имя макрокоманды;

фактические параметры — символические имена, которые соответствуют формальным параметрам, указанным в директиве .MACRO, разделенные запятой, пробелом или символом ГТ.

Имя макрокоманды должно совпадать с именем макроопределения.

Если имя макрокоманды совпадает с именем метки, то в поле операции это имя означает макрокоманду, а в поле операнда — метку.

ПРИМЕЧАНИЕ. Макроопределение должно быть введено директивой .MACRO до того, как будет выполнена макрокоманда и осуществлено макрорасширение.

Пример:

```
ABS: MOV (R0) ,R1          ; ABS — метка
       .
       .
       .
     BR ABS                ; переход к метке ABS
       .
       .
       .
     ABS #4,ENT,LAR        ; ABS — макрокоманда
```

8.3. **Параметры в макроопределениях и макрокомандах.** Параметры в макроопределениях и макрокомандах могут быть позиционными и ключевыми.

Позиционные параметры находятся в строго позиционной

зависимости: первый фактический параметр в макрокоманде соответствует первому формальному параметру в макроопределении.

Ключевые параметры — параметры, содержащие ключевые слова.

Формат ключевого параметра:

имя = последовательность символов

где имя — формальный параметр (ключевое слово);

Последовательность символов — значение ключевого слова, соответствующее фактическому параметру.

При задании ключевого параметра в списке формальных параметров макроопределения указанная последовательность символов становится фактическим параметром в макрокоманде по умолчанию.

При включении ключевого параметра в список фактических параметров указанная последовательность символов становится значением формального параметра, который точно совпадает с указанным именем, независимо от того, был ли указан формальный параметр с ключевым словом или без него.

Ключевой параметр может быть указан в любом месте списка формальных параметров макроопределения и является частью позиционно-упорядоченного списка. С другой стороны, ключевой параметр может быть указан в любом месте списка фактических параметров макрокоманды и не влияет на позиционное соответствие остальных параметров.

Параметры, содержащие разделительные символы, должны быть ограничены символами «<» и «>» (пример 1).

Параметры, содержащие специальные символы, ограничивать символами «<» и «>» не обязательно (пример 2).

Символ «^» позволяет использовать символы «<» и «>» как часть параметра (пример 3).

Параметры могут включать специальные символы без ограничения их символами «<» и «>», если только данный параметр не содержит разделительных символов (запятой, точки с запятой, пробела или символа ГТ).

Примеры:

1)

```
.MACRO REN A,B,C
      .
      .
      .
.ENDM
      .
```

```
REN <MOV X,Y>,#44,WEN
```

Фактический параметр MOV X,Y соответствует формальному параметру A.

2)
```
.MACRO PUSH ARG
MOV ARG,—(SP)
.ENDM
PUSH X+3(%2)
```

Макрорасширение команды PUSH будет следующим:

```
MOV X+3(%2),—(SP)
```

3) В макрокоманде

```
REN ^/<MOV X,Y>/,#44,WEN
```

последовательность символов <MOV X,Y> является одним из параметров макрокоманды.

При записи параметров, ограниченных символами «<» и «>», могут использоваться пробелы для наглядности.

Если в макрокоманде задано больше параметров, чем в макроопределении, то в листинге программы печатается сообщение об ошибке «Q». Если в макрокоманде задано меньше параметров, чем в макроопределении, то отсутствующим параметрам присваиваются пустые значения. Директивы условной трансляции .IF B и .IF NB могут быть использованы для обнаружения отсутствующих параметров. Число параметров может быть определено также по директиве .NARG.

ПРИМЕЧАНИЕ. Макрокоманда может быть задана без каких-либо параметров.

Пример:

В данном примере показано использование ключевых параметров.

```
 1                      .GLOBL A,B,C,TEMP,VAR,ALB,BAK
 2                      .MACRO TECT CON=
                        =1,BLOCK,ADDR=TEMP
 3                      .WORD CON
 4                      .WORD BLOCK
 5                      .WORD ADDR
 6                      .ENDM
 7
 8 000000               TECT A,B,C
   000000 000000G       .WORD A
   000002 000000G       .WORD B
   000004 000000G       .WORD C
 9
10 000006               TECT ADDR=20,BLOCK=30,CON=40
   000006 000040        .WORD 40
```

```
   000010 000030    .WORD 30
   000012 000020    .WORD 20
11
12
13 000014           TECT BLOCK=5
   000014 000001    .WORD 1
   000016 000005    .WORD 5
   000020 000000G   .WORD TEMP
14
15 000022           TECT CON=5,ADDR=VAR
   000022 000005    .WORD 5
   000024 000000    .WORD 0
   000026 000000G   .WORD VAR
16
17 000030           TECT
   000030 000001    .WORD 1
   000032 000000    .WORD
   000034 000000G   .WORD TEMP
18
19 000036           TECT ADDR=ALB!BAK
   000036 000001    .WORD 1
   000040 000000    .WORD
   000042 000000G   .WORD ALB!BAK
20
21         000001   .END
```

8.3.1. Конкатенация параметров

Символ «'» (апостроф), который стоит перед формальным параметром в макроопределении или следует за ним, означает, что в макрорасширении фактический параметр, соответствующий данному формальному параметру, будет соединяться с символами, стоящими рядом с формальным параметром.

Пример:

```
      .MACRO DEF A,B,C
A'B:  .ASCIZ /C/
      .BYTE "A,"B
      .ENDM
            .
            .
            .
      DEF X,Y,<MACRO B03.00>
            .
            .
            .
```

Макрокоманда DEF X,Y,<MACRO B03.00> вызывает макрорасширение:

```
XY:     .ASCIZ /MACRO B03.00/
        .BYTE 'X,'Y
```

В макрорасширении метке A'B соответствует метка XY, так как апостроф опускается и формальные параметры заменяются на фактические. В операндах директивы .BYTE первый символ апостроф остается в макрорасширении, так как перед ним и после него не следует формальный параметр в макроопределении. Второй апостроф опускается и подставляется фактический параметр. В результате директива .BYTE записывается в виде .BYTE 'X,'Y.

8.3.2. Арифметический параметр

Символ «\», который стоит перед параметром в макрокоманде, определяет его как арифметический параметр, т. е. в макрорасширении этот параметр заменяется числовым значением, в текущей системе счисления, которая устанавливается директивой .RADIX.

Пример:

```
 1                                          .MACRO INC A,B
 2                                          CON A,\B
 3                                          B=B+1
 4                                          .ENDM
 5
 6                                          .MACRO CON A,B
 7                                  A'B:    .WORD 4
 8                                          .ENDM
 9
10
11 000000      160 162          157        .ASCIZ /программа/
   000003      147 162          141
   000006      155 155          141
   000011      000
12                                          .EVEN
13
14          000000                          C=0
15 000012                                   ING X,C
   000012                                   CON X,\C
   000012 000004                    X0:     .WORD 4
          000001                            C=C+1
16
17 000014 000010                            .WORD 10
19 000016                                   INC X,C
```

```
    000016                          CON X,\C
    000016 000004           X1:     .WORD 4
           000002                   C=C+1
20
21  000020     007                  .BYTE 7
22                                  .EVEN
23
24  000022                          INC X,C
    000022
    000022 000004           X2:     .WORD 4
           000003                   C=C+1
25
26         000001                   .END
```

Арифметические параметры можно использовать при обозначении различных вариантов исходной программы.

Пример:

```
ID=6
.MACRO IDT SYM
.IDENT /V05A'SYM/
.ENDM
   .
   .
   .
IDT \ID
```

Макрорасширение IDT \ID будет:

```
.IDENT /V05A6/
```

8.3.3. Локальные метки в макрорасширениях.

В качестве меток в макрорасширениях используются локальные метки, создаваемые АССЕМБЛЕРом автоматически. Такие метки записываются в возрастающем порядке в виде N¤ (30000. <=N<=65535.).

Для задания автоматического создания локальных меток необходимо в макроопределении перед формальным параметром указать символ «?». При вызове макрокоманды соответствующий фактический параметр должен отсутствовать или быть равным 0. Если же в макрокоманде фактический параметр определен, автоматического создания локальных меток не осуществляется и выполняется обычная замена формального параметра на фактический.

Использование локальных меток, создаваемых автоматически, позволяет избежать многократного определения метки.

Автоматическое создание локальных меток возможно только для первых 16. параметров макроопределения.

При использовании автоматической генерации локальных меток необходимо быть уверенным, что генерируемые локальные метки и ссылки на них будут находиться в одном блоке локальных меток.

Пример:

```
        .MACRO ALRHA A,?B
        TST A
        BEQ B
        ADD #5,A
      B:.WORD 100
        .ENDM
        ALPHA R1              ;локальная метка создается
        TST R1
        BEQ 30000¤
        ADD #5,R1
30000¤:.WORD 100
        ALPHA R2,XYZ          ;локальная метка не созда-
        TST R2                ;ется
        BEQ XYZ
        ADD #5,R2
    XYZ:.WORD 100
```

8.3.4. Уровни макрокоманд

Каждое макроопределение может содержать любое число внутренних макрокоманд. Макрокоманда, используемая в некотором макроопределении, называется внутренней макрокомандой, а макрокоманда, соответствующая данному макроопределению, — внешней макрокомандой.

Внешние макрокоманды, используемые в исходной программе, называются макрокомандами первого уровня. Если макроопределение, соответствующее макрокоманде первого уровня, содержит внутренние макрокоманды, то последние называются макрокомандами второго уровня и т. д. Число уровней макрокоманд зависит от объема памяти, занимаемого исходной программой, которая будет транслироваться.

Формальный параметр внутренней макрокоманды должен быть ограничен символами «<» и «>», если соответствую-

щий ему фактический параметр содержит разделительные символы. Символы «<» и «>» для каждого уровня макрокоманд используются в макроопределении, но не в макрокоманде.

Пример:

```
.MACRO LEVEL1 DUM1,DUM2
LEVEL2 <DUM1>
LEVEL2 <DUM2>
.ENDM
.MACRO LEVEL2 DUM3
DUM3
ADD #10,R0
MOV R0, (R1)+
.ENDM
```

Для макрокоманды: LEVEL1 <MOV X,R0>,<MOV R2,R0> макрорасширение имеет вид:

```
MOV X,R0
ADD #10,R0
MOV R0,(R1)+
MOV R2,R0
ADD #10,R0
MOV R0,R1
```

Макроопределение может содержать внутри другое макроопределение. В этом случае нельзя вызывать внутреннее макроопределение до тех пор, пока не будет вызвано и расширено внешнее макроопределение.

Пример:

```
.MACRO LV1 A,B
     .
     .
     .
.MACRO LV2 C
     .
     .
     .
.ENDM
.ENDM
```

Макроопределение LV2 не может быть вызвано и расширено до тех пор, пока не будет вызвано макроопределение LV1.

8.4. Директивы определения характеристик макропараметров.

Директивы определения характеристик макропараметров позволяют определить чйсло параметров в макрокоманде, число символов в указанной последовательности символов, метод адресации указанного параметра макрокоманды.

К данной группе директив относятся: .NARG, .NCNR, .NTYPE.

8.4.1. Директива .NARG. Директива .NARG используется для определения числа параметров в макрокоманде.

Формат:

.NARG S

где S — допустимое символическое имя.

После выполнения директивы имя S получает значение, равное числу параметров в макрокоманде, для которой выполняется макрорасширение.

Директива .NARG записывается только в макроопределении. Если директива .NARG используется вне макроопределения, то в листинге печатается сообщение об ошибке «A».

Пример:

```
 1                      .MACRO NOPP,NUM
 2                      .NARG SYM
 3                      .IF EQ,SYM
 4                      .MEXIT
 5                      IFF
 6                      REPT NUM
 7                      NOP
 8                      ENDM
 9                      ENDC
10                      ENDM
11 000000               NOP
                        NARG SYM
                        IF EQ,SYM
                        MEXIT
                        IFF
                        REPT
                        NOP
                        ENDM
                        .ENDC
12 000000               NOP
          000001        .NARG SYM
```

```
                                .IF EQ,SYM
                                .MEXIT
                                .IFF
                        000006  .REPT 6
                                NOP
                                .ENDM
                000000  000240  NOP
                000002  000240  NOP
                000004  000240  NOP
                000006  000240  NOP
                000010  000240  NOP
                000012  000240  NOP
13                              .ENDC
14                      000001  .END
```

8.4.2. Директива .NCHR. Директива .NCHR используется для определения числа символов в указанной последовательности символов.

Формат:

.NCHR S,<C>,

где S — допустимое символическое имя;

, — разделительный символ: запятая, пробел или символ ГТ;

<C> — последовательность печатных символов. Данная последовательность символов должна быть ограничена символами «<» и «>» или символами «^», если она содержит разделительные символы: запятую, пробел или символ ГТ.

В результате выполнения директивы .NCHR имя S получает значение, равное числу символов в указанной последовательности. Директива .NCHR может записываться в любом месте исходной программы.

Директива .NCHR может быть использована для определения длины макропараметров.

Если символическое имя S не указано, то в листинге печатается сообщение об ошибке «A». Данное сообщение печатается также в случае, если заданы непарные ограничители последовательности символов или если конечный ограничитель не может быть найден из-за синтаксической ошибки в последовательности символов (происходит преждевременное прекращение обработки последовательности символов).

Пример:

```
1                               .MACRO CHAR MESS
2                               .NCHR SYM, MESS
3                               .WORD SYM
4                               .ASCIZ /MESS/
```

```
5                                       .EVEN
6                                       .ENDM
7
8 000000                          MSG:  CHAR <ОШИБКА>
          000006                        .NCHR SYM,ошибка
  000000  000006                        .WORD SYM
  000002      157 173 151               .ASCIZ /ошибка/
  000005      142 153 141
  000010      000
9         000001                        .EVEN
                                        .END
```

8.4.3. Директива .NTYPE. Директива .NTYPE используется для определения метода адресации указанного параметра макрокоманды.

Формат:

.NTYPE S,E,

где S — допустимое символическое имя;

, — разделительный символ: запятая, пробел или символ ГТ;

E — допустимый параметр, метод адресации которого необходимо определить.

В макрорасширении имени S присваивается значение, равное значению 6-разрядного метода адресации указанного параметра.

Если параметр E не указан, то результат будет равен нулю.

Директива .NTYPE записывается только в макроопределении. Если она появится в другом месте, то в листинге печатается сообщение об ошибке «A».

Пример:

```
 1                                      .MACRO SAVE,ARG
 2                                      .NTYPE SYM,ARG
 3                                      .IF EQ,SYM&70
 4                                       MOV ARG,—(SP)
 5                                      .IFF
 6                                       MOV #ARG,—(SP)
 7                                      .ENDC
 8                                      .ENDM
 9
10 000000 000000                 TEMP: .WORD 0
11
12 000002                              .SAVE R1
          000001                       .NTYPE SYM,R1
                                       .IF EQ,SYM&70
```

```
                                    MOV R1,—(SP)
    000002 010146                   .IFF
                                    MOV #R1,—(SP)
                                    .ENDC
13
14
15 000004                           .SAVE TEMP
           000067                   .NTYPE SYM,TEMP
                                    .IF EQ,SYM&70
                                    MOV TEMP,—(SP)
                                    .IFF
    000004 012746 000000'           MOV #TEMP,—(SP)
16                                  .ENDM
17         000001                   .END
```

8.5. Директивы .ERROR и .PRINT. Директивы .ERROR и .PRINT используются для печати в листинге программы предусмотренных сообщений. Если вывод листинга не задан, сообщения выводятся на терминал.

Формат:

.ERROR [E];текст

.PRINT [E];текст,

где E — допустимое выражение, значение которого выводится на терминал при появлении директивы;
текст — текст указанного сообщения, которое будет печататься на терминале.

Сообщения, выводимые по директиве, содержат:
— сообщение об ошибке «P» (печатается только для директивы .ERROR);
— порядковый номер строки, содержащей директиву .ERROR;
— текущее значение счетчика адреса;
— значение выражения, указанного в директиве;
— исходную строку, содержащую директиву .ERROR.

Директива .ERROR может быть использована для печати сообщения о невыполненном или об ошибочном вызове макрокоманды, о существовании недопустимых условий, указанных в директиве условной трансляции.

Директива .PRINT аналогична директиве .ERROR, за исключением того, что она не выдает сообщение об ошибке «P».

Пример:

Директива .ERROR A ; INVALID ARGUMENT вызывает печать следующего сообщения:

```
P  512 005642 000076 .ERROR A ; INVALID ARGUMENT.
```

8.6. Директивы задания области неопределенных повторений .IRP и .IRPC.

Область неопределенных повторений по своей структуре аналогична макроопределению, которое имеет только один формальный параметр. Расширение области происходит в том месте программы, где она определена. При каждом расширении области неопределенных повторений формальный параметр заменяется последовательными элементами из указанного списка фактических параметров. Эта область может находиться как в основной части программы, так и в макроопределении, в области неопределенных повторений и в области повторений.

8.6.1. Директива .IRP. Директива .IRP используется для указания начала области неопределенных повторений. Во время расширения области неопределенных повторений происходит последовательная замена формального параметра фактическими параметрами.

Формат области неопределенных повторений:

```
.IRP S,<A1,A2,...,AN>
      .
      .
      .
Область
неопределенных
повторений
      .
      .
      .
.ENDM
```

где S — формальный параметр, представляющий собой допустимое символическое имя;

, — разделительный символ: запятая, пробел или символ ГТ;

<A1,A2,...,AN> — список фактических параметров, ограниченный символами «<» и «>». Фактическим параметром может быть одиночный символ или последовательность символов. Если список содержит несколько параметров, то они отделяются разделительными символами: запятой, пробелом или символом ГТ;

Область неопределенных повторений — блок программы, который может быть повторен один раз для каждого параметра из списка фактических параметров;

.ENDM — директива окончания области неопределенных повторений.

Область неопределенных повторений может включать другие макроопределения и области повторений. Директива .MEXIT допустима в пределах области неопределенных повторений.

Пример использования директивы .IRP приведен в п. 8.6.2.

8.6.2. Директива .IRPC. Директива .IRPC определяет начало области неопределенных повторений и используется для замены отдельных символов.

При каждом расширении области неопределенных повторений формальный параметр последовательно заменяется одним символом из указанной последовательности символов.

Формат:

```
.IRPC S,<C>
      .
      .
      .
Область
неопределенных
повторений
      .
      .
      .
.ENDM,
```

где S — формальный параметр, представляющий собой допустимое символическое имя;

,— разделительный символ: запятая, пробел или символ ГТ;

<C> — последовательность символов, ограниченная символами «<» и «>»;

область неопределенных повторений — блок программы, который должен быть повторен один раз для каждого символа из последовательности;

.ENDM — директива окончания области неопределенных повторений.

Область неопределенных повторений может содержать макроопределения и области повторения. Директива .MEXIT допустима в пределах области неопределенных повторений

Если формальный параметр S не задан, в листинге печатается сообщение об ошибке «А».

ПРИМЕЧАНИЕ. Символы «<» и «>» используются для наглядности, а также в тех случаях, когда параметр «С» содержит разделительные символы.

Пример:

В данном примере показано использование директив .IRP и .IRPC.

```
1                       .TITLE IRPTST
2                       .LIST ME
3 000000    REGS:       .IRP REG,<PC,SP,R5,R4,R3,R2,R1,R0>
4                       .RAD50 /REG/
5                       .ENDR

  000000 062170                 .RAD50 /PC/
  000002 074500                 .RAD50 /SP/
  000004 072770                 .RAD50 /R5/
  000006 072720                 .RAD50 /R4/
  000010 072650                 .RAD50 /R3/
  000012 072600                 .RAD50 /R2/
  000014 072530                 .RAD50 /R1/
  000016 072460                 .RAD50 /R0/
6 000020            REGS2:      .IRPC NUM,<76543210>
7                               .RAD50 /R'NUM/
8                               .ENDR
  000020 073110                 .RAD50 /R7/
  000022 073040                 .RAD50 /R6/
  000024 072770                 .RAD50 /R5/
  000026 072720                 .RAD50 /R4/
  000030 072650                 .RAD50 /R3/
  000032 072600                 .RAD50 /R2/
  000034 072530                 .RAD50 /R1/
  000036 072460                 .RAD50 /R0/
9         000001                .END
```

8.7. Директивы .REPT и .ENDR. Директивы .REPT и .ENDR определяют область повторений и используются для указания блока программы, повторяемого несколько раз.

Формат области повторений:

```
.REPT E
      .
      .
      .
Область
повторений
      .
      .
      .
.ENDR
```

где E — абсолютное выражение, значение которого определяет число повторений области повторений;

Область повторений — блок программы, который должен быть повторен заданное число раз;

.ENDR — директива окончания области повторений.

Конец области повторений может быть указан директивой .ENDM.

Область повторений может включать макроопределения, области неопределенных повторений или другие области повторений. Директива .MEXIT допустима в пределах области повторений.

Если выражение E меньше или равно нулю, то область повторений не транслируется. Если E не является абсолютным выражением, то в листинге печатается сообщение об ошибке «A».

8.8. Директива .MCALL. Директива .MCALL используется для указания имен макроопределений из макробиблиотек, которые используются в исходной программе. Директива .MCALL позволяет указать имена макроопределений системной макробиблиотеки и макробиблиотек пользователя, которые не определены в текущей программе, но необходимы для трансляции программы. Директива .MCALL записывается в исходной программе до появления первой упомянутой в ней макрокоманды.

Формат:

.MCALL A1,A2,...,AN,

где A1,A2,...,AN — имена макроопределений из макробиблиотек.

Имена макроопределений разделяются запятыми, пробелами или символами <ГТ>.

Если после просмотра макробиблиотек макроопределение с указанным именем не будет обнаружено, то в строке листинга, содержащей директиву .MCALL, печатается сообщение об ошибке «U», а в строке, содержащей макрокоманду с неопределенным именем, печатается сообщение об ошибке «O».

8.9. Директива .MDELETE. Директива .MDELETE стирает указанные макроопределения и освобождает виртуальную память.

Формат:

.MDELETE S1,S2,...,SN,

где S1,S2,...,SN — допустимые имена макроопределений.

Если указано несколько имен, то они разделяются любым допустимым разделителем: запятой, пробелом или символом <ГТ>.

При обращении к стертым макроопределениям печатается сообщение об ошибке «O».

Пример:

.MDELETE .EXIT,EXIT¤S

ПРИЛОЖЕНИЕ 1

СПЕЦИАЛЬНЫЕ СИМВОЛЫ ЯЗЫКА АССЕМБЛЕР

В таблице данного приложения приведены специальные символы языка АССЕМБЛЕР.

СПЕЦИАЛЬНЫЕ СИМВОЛЫ

Таблица

Символ	Назначение символа
1	2
:	Ограничитель метки
=	Оператор прямого присваивания
%	Указатель регистра
ГТ	Ограничитель аргументов или ограничитель поля
Пробел	Ограничитель аргументов или ограничитель поля
#	Указатель непосредственной адресации
@	Указатель косвенной адресации
(	Начальный указатель регистра
)	Конечный указатель регистра
,	Разделитель поля операндов
;	Указатель поля комментария
+	Знак арифметической операции сложения или указатель автоинкрементного метода адресации
—	Знак арифметической операции вычитания или указатель автодекрементного метода адресации
*	Знак арифметической операции умножения
/	Знак арифметической операции деления
&	Знак логической операции «И»
!	Знак логической операции «ИЛИ»
»	Указатель двух символов КОИ-7
'	Указатель одиночного символа КОИ-7 или признак конкатенации
.	Указатель счетчика адреса
<	Указатель начала параметра
>	Указатель конца параметра
^	Указатель специальной операции или указатель параметра
\	Указатель арифметического параметра макрокоманды
ВТ	Ограничитель строки программы

ПРИЛОЖЕНИЕ 2

ПРЕДСТАВЛЕНИЕ СИМВОЛОВ В КОДЕ КОИ-7 И RADIX-50

В данном приложении приведено представление символов в 7-разрядном коде для обмена информацией КОИ-7 Н0 (ASCII) (табл. 1), КОИ-7 Н1 (табл. 2), RADIX-50 (табл. 3) и позиционное значение кода RADIX-50 (табл. 4).

7-БИТНЫЙ КОД ASCII

Таблица 1

000	NUL	020	DLE	040	SP	060	0	100	@	120	P	140	`	160	p	
001	SOH	021	DC1	041	!	061	1	101	A	121	Q	141	a	161	q	
002	STX	022	DC2	042	"	061	2	102	B	122	R	142	b	162	r	
003	ETX	023	DC3	043	#	063	3	103	C	123	S	143	c	163	s	
004	EOT	024	DC4	044	$	064	4	104	D	124	T	144	d	164	t	
005	ENQ	025	NEK	045	%	065	5	105	E	125	U	145	e	165	u	
006	ACK	026	SYN	046	&	066	6	106	F	126	V	146	f	166	v	
007	BEL	027	ETB	047	'	067	7	107	G	127	W	147	g	167	w	
010	BS	030	CAN	050	(	070	8	110	H	130	X	150	h	170	x	
011	HT	031	EM	051	)	071	9	111	I	131	Y	151	i	171	y	
012	LF	032	SUB	052	*	072	:	112	J	132	Z	152	j	172	z	
013	VT	033	ESC	053	+	073	;	113	K	133	[	153	k	173	{	
014	FF	034	FS	054	,	074	<	114	L	134	\	154	l	174	|	
015	CR	035	GS	055	-	075	=	115	M	135	]	155	m	175	}	
016	SO	036	RS	056	.	076	>	116	N	136	^	156	n	176	~	
017	SI	037	US	057	/	077	?	117	O	137	_	157	o	177	DEL	

КОИ-7 Н1

Таблица 2

000 NUL	020 DLE	040 SP	060 0	100 ю	120 п	140 Ю	160 П
001 SOH	021 DC1	041 !	061 1	101 а	121 я	141 А	161 Я
002 STX	022 DC2	042 "	061 2	102 б	122 р	142 Б	162 Р
003 ETX	023 DC3	043 #	063 3	103 ц	123 с	143 Ц	163 С
004 EOT	024 DC4	044 $	064 4	104 д	124 т	144 Д	164 Т
005 ENQ	025 NEK	045 %	065 5	105 е	125 у	145 Е	165 У
006 ACK	026 SYN	046 &	066 6	106 ф	126 ж	146 Ф	166 Ж
007 BEL	027 ETB	047 '	067 7	107 г	127 в	147 Г	167 В
010 BS	030 CAN	050 (	070 8	110 х	130 ь	150 Х	170 Ь
011 HT	031 EM	051)	071 9	111 и	131 ы	151 И	171 Ы
012 LF	032 SUB	052 *	072 :	112 й	132 з	152 Й	172 З
013 VT	033 ESC	053 +	073 ;	113 к	133 ш	153 К	173 Ш
014 FF	034 FS	054 ,	074 <	114 л	134 э	154 Л	174 Э
015 CR	035 GS	055 -	075 =	115 м	135 щ	155 М	175 Щ
016 SO	036 RS	056 .	076 >	116 н	136 ч	156 Н	176 Ч
017 SI	037 US	057 /	077 ?	117 о	137 ъ	157 О	177 Ъ

Таблица 3

КОД RADIX50

Обозначение символа	Восьмеричный код	Код RADIX50
1	2	3
Пробел	40	0
A—Z	101—132	1—32
¤	44	33
	56	34
Не используется		35
0—9	60—71	36—47

Пример преобразования последовательности символов X,2,B в код RADIX-50:

.RAD50 /X2B/

В коде RADIX-50 получим следующее значение (арифметические действия выполняются в восьмеричной системе):

```
  X=113000
+ 2=002400
  B=000002
  --------
X2B=115402
```

Таблица 4

Символ	Код одиночного или первого символа	Код второго символа	Код третьего символа
1	2	3	4
Пробел	000000	000000	000000
A	003100	000050	000001
B	006200	000120	000002
C	011300	000170	000003
D	014400	000240	000004
E	017500	000310	000005
F	022600	000360	000006
G	025700	000430	000007
H	031000	000500	000010
I	034100	000550	000011
J	037200	000620	000012
K	042300	000670	000013
L	045400	000740	000014
M	050500	001010	000015
N	053600	001060	000016

1	2	3	4
O	056700	001130	000017
P	062000	001200	000020
Q	065100	001250	000021
R	070200	001320	000022
S	073300	001370	000023
T	076400	001440	000024
U	101500	001510	000025
V	104600	001560	000026
W	107700	001630	000027
X	113000	001700	000030
Y	116100	001750	000031
Z	121200	002020	000032
¤	124300	002070	000033
	127400	002140	000034
Не использу-ется	132500	002210	000035
0	135600	002260	000036
1	140700	002330	000037
2	144000	002400	000040
3	147100	002450	000041
4	152200	002520	000042
5	155300	002570	000043
6	160400	002640	000044
7	163500	002710	000045
8	166600	002760	000046
9	171700	003030	000047

ПРИЛОЖЕНИЕ 3

КОМАНДЫ АССЕМБЛЕРа

В данном приложении приведены команды АССЕМБЛЕРа: команды центрального процессора (табл. 1) и команды процессора с плавающей запятой (табл. 2).

Таблица 1

КОМАНДЫ ЦЕНТРАЛЬНОГО ПРОЦЕССОРА

Мнемоника	Код команды	Операция
1	2	3
ADC	005500	Прибавление переноса
ADCB	105500	Прибавление переноса к байту
ADD	060000	Сложение
ASH	072000	Арифметический сдвиг
ASHC	073000	Арифметический сдвиг двойного слова
ASL	006300	Арифметический сдвиг влево
ASLB	106300	Арифметический сдвиг влево байта
ASR	006200	Арифметический сдвиг вправо
ASRB	106200	Арифметический сдвиг вправо байта
BCC	103000	Ветвление, если нет переноса
BCS	103400	Ветвление, если перенос
BEQ	001400	Ветвление, если равно (нулю)
BGE	002000	Ветвление, если больше или равно (нулю)
BGT	003000	Ветвление, если больше (нуля)
BHI	101000	Ветвление, если больше
BHIS	103000	Ветвление, если больше или равно
BIC	040000	Очистка разрядов
BICB	140000	Очистка разрядов байта
BIS	050000	Логическое сложение
BISB	150000	Логическое сложение байтов
BIT	030000	Проверка разрядов
BITB	130000	Проверка разрядов байта
BLE	003400	Ветвление, если меньше или равно (нулю)
BLO	103400	Ветвление, если меньше
BLOS	101400	Ветвление, если меньше или равно
BLT	002400	Ветвление, если меньше (нуля)
BMI	100400	Ветвление, если минус
BNE	001000	Ветвление, если не равно (нулю)
BPL	100000	Ветвление, если плюс
BPT	000003	Командное прерывание для отладки
BR	000400	Ветвление безусловное

1	2	3
BVC	102000	Ветвление, если нет арифметического переполнения
BVS	102400	Ветвление, если арифметическое переполнение
CALL	004700	Обращение к подпрограмме (JSR PC,XXX)
CALLR	000100	Безусловная передача управления
CCC	000257	Очистка всех разрядов (N,Z,V,C)
CLC	000241	Очистка C
CLN	000250	Очистка N
CLR	005000	Очистка
CLRB	105000	Очистка байта
CLV	000242	Очистка V
CLZ	000244	Очистка Z
CMP	020000	Сравнение
CMPB	120000	Сравнение байтов
COM	005100	Инвертирование
COMB	105100	Инвертирование байта
DEC	005300	Вычитание единицы
DECB	105300	Вычитание единицы из байта
DIV	071000	Деление
EMT	104000	Программное прерывание для системных программ
FADD	075000	Сложение с плавающей запятой
FDIV	075030	Деление с плавающей запятой
FMUL	075020	Умножение с плавающей запятой
FSUB	075010	Вычитание с плавающей запятой
HALT	000000	Останов
INC	005200	Прибавление единицы
INCB	105200	Прибавление единицы к байту
IOT	000004	Командное прерывание для ввода-вывода
JMP	000100	Безусловная передача управления
JSR	004000	Обращение к подпрограмме
MARK	006400	Восстановление УС
MFPI	006500	Пересылка из предыдущей области команд
MFPS	106700	Чтение ССП
MFPT	000007	Модель процессора
MOV	010000	Пересылка
MOVB	110000	Пересылка байта
MTPI	006600	Пересылка в предыдущую область команд
MTPS	106400	Запись ССП
MUL	070000	Умножение
NEG	005400	Изменение знака
NEGB	105400	Изменение знака байта
NOP	000240	Нет операции
RESET	000005	Сброс внешних устройств
RETURN	000207	Возврат из подпрограммы (RTC PC)
ROL	006100	Циклический сдвиг влево
ROLB	106100	Циклический сдвиг влево байта
ROR	006000	Циклический сдвиг вправо
RORB	106000	Циклический сдвиг вправо байта
RTI	000002	Возврат из прерывания
RTS	000200	Возврат из подпрограммы

1	2	3
RTT	000006	Возврат из прерывания
SBC	005600	Вычитание переноса
SBCB	105600	Вычитание переноса из байта
SCC	000277	Установка всех разрядов (N,Z,V,C)
SEC	000261	Установка C
SEN	000270	Установка N
SEV	000262	Установка V
SEZ	000264	Установка Z
SOB	077000	Вычитание единицы и ветвление
SUB	160000	Вычитание
SWAB	000300	Перестановка байтов
SXT	006700	Расширение знака
TRAP	104400	Командное прерывание
TST	005700	Проверка
TSTB	105700	Проверка байта
TSTSET	007200	
WAIT	000001	Ожидание
WRTLCK	007300	
XOR	740000	Исключающее ИЛИ

Таблица 2

КОМАНДЫ ПРОЦЕССОРА С ПЛАВАЮЩЕЙ ЗАПЯТОЙ

Мнемоника	Код команды	Операция
1	2	3
ABSD	170600	Вычисление абсолютного значения числа с двойной точностью
ABSF	170600	Вычисление абсолютного значения числа с одинарной точностью
ADDD	172000	Сложение с двойной точностью
ADDF	172000	Сложение с одинарной точностью
CFCC	170000	Копирование признаков ПЗ
CLRD	170400	Очистка с двойной точностью
CLRF	170400	Очистка с одинарной точностью
CMPD	173400	Сравнение с двойной точностью
CMPF	173400	Сравнение с одинарной точностью
DIVD	174400	Деление с двойной точностью
DIVF	174400	Деление с одинарной точностью
LDCDF	177400	Загрузка и преобразование числа с двойной точностью в число с одинарной точностью
LDCFD	177400	Загрузка и преобразование числа с одинарной точностью в число с двойной точностью
LDCID	177000	Загрузка и преобразование короткого числа в число с двойной точностью
LDCIF	177000	Загрузка и преобразование короткого числа в число с одинарной точностью
LDCLD	177000	Загрузка и преобразование длинного числа в число с двойной точностью
LDCLF	177000	Загрузка и преобразование длинного числа в число с одинарной точностью
LDD	172400	Загрузка с двойной точностью
LDEXP	176400	Загрузка порядка
LDF	172400	Загрузка с одинарной точностью
LDFPS	170100	Загрузка слова состояния программы ПЗ
MFPD	106500	Пересылка из предыдущей области данных
MODD	171400	Умножение и целочисленное представление с двойной точностью
MODF	171400	Умножение и целочисленное представление с одинарной точностью
MTPD	106600	Пересылка в предыдущую область данных
MULD	171000	Умножение с двойной точностью
MULF	171000	Умножение с одинарной точностью
NEGD	170700	Изменение знака числа с двойной точностью
NEGF	170700	Изменение знака числа с одинарной точностью
SETD	170011	Установка режима двойной точности
SETF	170001	Установка режима одинарной точности
SETI	170002	Установка режима короткого числа
SETL	170012	Установка режима длинного числа
SPL	000230	Установка уровня приоритета
STA0	170005	
STB0	170006	

1	2	3
STCDF	176000	Запись с преобразованием числа с двойной точностью в число с одинарной точностью
STCDI	175400	Запись с преобразованием числа с двойной точностью в короткое число
STCDL	175400	Запись с преобразованием числа с двойной точностью в длинное число
STCFD	176000	Запись с преобразованием числа с одинарной точностью в короткое число
STCFI	175400	Запись с преобразованием числа с одинарной точностью в короткое число
STCFL	175400	Запись с преобразованием числа с одинарной точностью в длинное число
STD	174000	Запись с двойной точностью
STEXP	175000	Запись порядка
STF	174000	Запись с одинарной точностью
STFPS	170200	Запись слова состояния программы
STST	170300	Запись состояния ПЗ
SUBD	173000	Вычитание с двойной точностью
SUBF	173000	Вычитание с одинарной точностью
TSTD	170500	Проверка с двойной точностью
TSTF	170500	Проверка с одинарной точностью

ПРИЛОЖЕНИЕ 4

МЕТОДЫ АДРЕСАЦИИ

В таблице данного приложения приведены методы адресации языка АССЕМБЛЕР.

Таблица

Формат	Адресация	Метод адресации
1	2	3
R	0N	Регистровый
@R или (ER)	1N	Косвенно-регистровый
(ER)+	2N	Автоинкрементный
@(ER)+	3N	Косвенно-автоинкрементный
—(ER)	4N	Автодекрементный
@—(ER)	5N	Косвенно-автодекрементный
E(ER)	6N	Индексный
@E(ER)	7N	Косвенно-индексный
#E	27	Непосредственный
@#E	37	Абсолютный
E	67	Относительный
@E	77	Косвенно-относительный

В вышеприведенной таблице N определяет используемый регистр.

ПРИЛОЖЕНИЕ 5

ДИРЕКТИВЫ АССЕМБЛЕРа

В таблице данного приложения приведены директивы АССЕМБЛЕРа.

Таблица

Мнемоника	Назначение
1	2
.ASCII	Используется для записи последовательности символов в коде КОИ-7
.ASCIZ	Используется для записи последовательности символов в коде КОИ-7 с добавлением нулевого байта в конце последовательности символов
.ASECT	Используется для определения абсолютной программной секции
.BLKB	Используется для резервирования области памяти (в байтах) в объектной программе
.BLKW	Используется для резервирования области памяти (в словах) в объектной программе
.BYTE	Используется для записи данных в двоичном коде в последовательно расположенных байтах
.GROSS	Используется для управления печатью и содержанием таблицы перекрестных ссылок
.CSECT	Используется для определения перемещаемой именованной или неименованной программной секции
.DSABL	Используется для запрещения выполнения заданных функций трансляции
.ENABL	Используется для разрешения выполнения заданных функций трансляции
.END	Используется для указания логического конца исходного модуля
.ENDC	Используется для указания конца блока условной трансляции
.ENDM	Используется для указания конца макроопределений или блока повторений
.ENDR	Используется для указания конца блока повторений
.ERROR	Используется для печати предусмотренных сообщений в листинге программы
.EVEN	Используется для установления четного значения счетчика адреса
.FLT2	Используется для записи десятичных чисел в форме с плавающей точкой с обычной точностью
.FLT4	Используется для записи десятичных чисел в форме с плавающей точкой с двойной точностью
.GLOBL	Используется для описания имен как глобальных

1	2
.IDENT	Используется для дополнительного обозначения объектного модуля
.IF	Используется для указания начала блока условной трансляции
.IFF	Используется только внутри блока условной трансляции и указывает на начало блока, который будет транслироваться, если значение условия ложное
.IFT	Используется только внутри блока условной трансляции и указывает на начало блока, который будет транслироваться, если значение условия истинное
.IFTF	Используется только внутри блока условной трансляции и указывает на начало блока, который будет транслироваться независимо от значения условия
.IIF	Используется для создания блока условной трансляции, содержащего одно предложение
.INCLUDE	Используется для ввода исходного файла в транслируемый файл
.IRP	Используется для указания начала области неопределенных повторений с заменой формального параметра на фактические
.IRPC	Используется для определения области неопределенных повторений с заменой отдельных символов
.LIBRARY	Используется для включения имени файла макробиблиотеки пользователя в список просматриваемых АССЕМБЛЕРом библиотек
.LIMIT	Используется для резервирования двух слов памяти для записи младшего адреса загрузочного модуля и старшего адреса загрузочного модуля плюс два
.LIST	Используется для разрешения печати полей листинга, определенных аргументами
.MACRO	Используется для указания начала макроопределения
.MCALL	Используется для указания имен макроопределений из макробиблиотек, которые используются в исходной программе
.MDELETE	Используется для стирания указанных макроопределений и освобождения виртуальной памяти
.MEXIT	Используется для завершения текущего макрорасширения или области неопределенных повторений
.NARG	Используется для определения числа параметров в макрокоманде
.NCHR	Используется для определения числа символов в указанной последовательности символов
.NLIST	Используется для запрещения печати полей листинга, определенных аргументами
.NTYPE	Используется для опреления метода адресации указанного параметра макрокоманды
.NOCROSS	Используется для управления печатью и содержанием таблицы перекрестных ссылок
.ODD	Используется для установления нечетного значения счетчика адреса
.PAGE	Используется для печати текста с новой страницы листинга

1	2
.PACKED	Используется для упаковки десятичных данных в одно слово
.PRINT	Используется для печати в листинге программы предусмотренных сообщений
.PSECT	Используется для создания именованных или неименованных перемещаемых или абсолютных программных секций и распределения данных между программными секциями
.RADIX	Используется для установления основания системы счисления всей исходной программы или ее части
.RAD50	Используется для упаковки последовательности символов в код RADIX-50
.REPT	Используется для указания начала области повторений
.REM	Используется для ввода комментария в исходную программу
.PESTORE	Используется для восстановления из стека контекста программной секции
.SBTTL	Используется для формирования оглавления и обозначения каждой страницы листинга
.SAVE	Используется для сохранения в стеке контекста программной секции
.TITLE	Используется для присваивания имени объектному модулю
.WORD	Используется для записи данных в последовательно расположенных словах
.WEAK	Используется для описания имен как внешних

ПЕРЕЧЕНЬ ССЫЛОЧНЫХ ДОКУМЕНТОВ

1. Операционная система ФОДОС-2
АССЕМБЛЕР
Руководство программиста

АССЕМБЛЕР

РУКОВОДСТВО ПРОГРАММИСТА

1. НАЗНАЧЕНИЕ И УСЛОВИЯ ПРИМЕНЕНИЯ ПРОГРАММЫ

Данный документ является руководством программиста при трансляции программ, написанных на языке АССЕМБЛЕР операционной системы ФОДОС-2.

Результатом трансляции может быть:

1) объектный модуль (доступный машине логический эквивалент исходного модуля);
2) листинг исходного модуля;
3) листинг файла таблицы перекрестных ссылок;
4) листинг таблицы содержаний;
5) листинг таблицы имен.

Для работы с АССЕМБЛЕРом (MACRO) пользователь должен знать, как:

1) вызвать и закончить работу с транслятором MACRO, а также общий формат командной строки транслятора;
2) выводить временные рабочие файлы на устройства, которые не приняты по умолчанию, если необходимо;
3) используются переключатели спецификации файла для запрещения действия директив управления файлом в исходном модуле;
4) интерпретируются коды ошибок.

АССЕМБЛЕР (MACRO) предназначен для автоматизации программирования на уровне машинно-ориентированного языка. Транслятор переводит программу, написанную на языке АССЕМБЛЕР (исходный модуль), в объектную программу. Трансляция осуществляется за два прохода. Для трансляции программ с помощью АССЕМБЛЕРа необходимо, чтобы исходные модули находились на устройствах с файловой структурой. Минимальный объем оперативной памяти для работы АССЕМБЛЕРа — 32К байт.

2. ХАРАКТЕРИСТИКИ ПРОГРАММЫ

2.1. Режим работы. Режим трансляции задается введением с терминала командной строки. Формат командной строки:
обспф,[листспф],[пспф] [/прк] = входспф,...,входспф [/прк]

где обспф — спецификация объектного файла (устройство, имя и тип файла). Устройства TT: и LP: для получения объектного файла не используются;

листспф — спецификация файла листинга и таблицы имен;

пспф — спецификация временного файла перекрестных ссылок (CREF). Если пспф не указан, листинг таблицы перекрестных ссылок все равно создается;

/прк — один или несколько переключателей (табл. 2);

входспф — спецификация входного файла (исходного или файла макробиблиотеки пользователя).

В командной строке можно указать до шести входных файлов.

Значения элементов спецификаций файлов по умолчанию приведены в табл. 1.

Таблица 1

ЗНАЧЕНИЯ ЭЛЕМЕНТОВ СПЕЦИФИКАЦИЙ ФАЙЛОВ ПО УМОЛЧАНИЮ

Файл	Устройство	Имя файла	Тип файла
1	2	3	4
Объектный	DK:	Необходимо указать	.OBJ
Листинга	Устройство, назначенное для объектного модуля	То же	.LST
Перекрестных ссылок	DK:	CREF	.TMP
Исходный	Для первого файла DK: Для последующих — то же, что для предыдущего	Необходимо указать	.MAC
Системная макробиблиотека	SY:	SYSMAC	.SML
Макробиблиотека пользователя	Для первого файла DK: Для последующих — то же, что для предыдущего	Необходимо указать	.MLB

Спецификации выходных файлов (объектного файла, файла листинга, временного файла перекрестных ссылок) не обязательны, т. е. система не создает выходной файл, если командная строка не содержит спецификацию данного файла. Часть спецификации файла можно не указывать — см. табл. 1. Система определяет, какой выходной файл должен быть создан при трансляции по положению спецификации файла в командной строке. Если какой-либо выходной файл не надо создавать, в командной строке на месте спецификации данного файла следует указать запятую, но не требуется ставить запятую после спецификации последнего нужного выходного файла.

2.2. Временный рабочий файл.

Если при трансляции таблице имен необходима область памяти большего размера, чем ей отводится, система автоматически создает временный рабочий файл WRK.TMP для расширения области таблицы имен.

По умолчанию файл WRK.TMP создается на устройстве DK:. Чтобы назначить другое устройство по умолчанию для создания временного рабочего файла, следует подать команду:

.ASSING уст: WF

где уст: — постоянное имя устройства файловой структуры, куда будет выводиться файл WRK.TMP.

После данной команды система создает файл WRK.TMP на указанном устройстве.

2.3. Переключатели.

Используя переключатели программы АССЕМБЛЕР, пользователь может запретить действие директив языка АССЕМБЛЕР исходной программы.

Переключатели, допустимые в командной строке, приведены в табл. 2.

Таблица 2

ПЕРЕКЛЮЧАТЕЛИ ПРОГРАММЫ АССЕМБЛЕР

Переключатель	Назначение
1	2
/L:apr	Определяет формат и содержание листинга; запрещает директиву .NLIST исходной программы
/N:apr	Определяет формат и содержание листинга; запрещает директиву .LIST исходной программы

1	2
/E:арг	Разрешает определенные функции при трансляции и при формировании объектного модуля; запрещает директиву .DSABL исходной программы
/D:арг	Запрещает определенные функции при трансляции и при формировании объектного модуля; запрещает директиву .ENABL исходной программы
/M	Указывает, что входной файл является файлом макробиблиотеки
/C:арг	Определяет содержание таблицы перекрестных ссылок

ПРИМЕЧАНИЕ. Переключатель /M действует только на тот файл, за спецификацией которого он следует в командной строке. Все другие переключатели программы АССЕМБЛЕР могут быть указаны в любом месте командной строки.

Примеры:

1. *DK:BINF.OBJ,LP:=DK:SRC.MAC.

По данной командной строке транслируется файл SRC.MAC и создается объектный файл DK:BINF.OBJ, листинг выводится на LP:.

2. *,LP:/C=DK:SRC.MAC

По данной командной строке на LP: выводится листинг, включающий временный файл перекрестных ссылок.

Ниже описаны все переключатели программы АССЕМБЛЕР.

2.3.1. Переключатели управления листингом (/L:арг и /N:арг).

Переключатели /L:арг и /N:арг определяют формат и содержание листинга. Аргументы переключателей приведены в табл. 3. Используя данные переключатели, можно запретить во время трансляции действие директив исходной программы .LIST и .NLIST с определенными аргументами: переключатель /L:арг запрещает действие директивы .LIST арг. Аргументы переключателя и аргументы директивы в этом случае должны совпадать. Описание директив .LIST и .NLIST приведено в документе [2].

Переключатели управления листингом могут использоваться без аргументов: по /L запрещается действие директив .LIST и .NLIST, которые не имеют аргументов; по /N система включает в листинг только таблицу имен, таблицу содержания и сообщения об ошибках.

Пример:

*I,LP:/L:MEB/N:SYM=FILE

По данной командной строке транслируется файл FILE.MAC и на LP: выводится листинг (с шириной строки 132 символа), без таблицы имен (по /N:SYM), но включая операторы макрорасширений, порождающие объектные коды (по /L:MEB), и создается объектный файл I.OBJ.

Таблица 3

АРГУМЕНТЫ ПЕРЕКЛЮЧАТЕЛЕЙ /L:арг и /N:арг

Аргумент	Действие по умолчанию	Назначение
1	2	3
SEQ	Разрешает	Включать в листинг порядковые номера строк исходной программы
LOC	»	Печать значения счетчика адреса
BIN	»	Печать объектных кодов в восьмеричном виде
BEX	»	Печать объектных кодов в восьмеричном виде, расположенных на нескольких строках
SRC	»	Печать операторов исходной программы
COM	»	Печать комментария
MD	»	Печать макроопределений и расширений областей повторений
MC	»	Печать макрокоманд и расширений областей повторений
ME	Запрещает	Печать макрорасширений
MEB	Запрещает	Печать операторов макрорасширений, порождающих объектные коды
CND	Разрешает	Печать блоков условной трансляции с невыполненными условиями
LD	Запрещает	Действие директив .LIST и .NLIST исходной программы, в которых нет аргументов
TOC	Разрешает	Печать таблицы содержания
TTM	Печать в строку	Управлять форматом печати объектных кодов и таблицы имен (см. рис. 1) Приведен формат, заданный переключателем /L:TTM
SYM	»	Печать таблицы имен

2.3.2. Переключатели управления функциями (/D:арг и /E:арг). Переключатели /D:арг и /E:арг позволяют разрешать или запрещать определенные функции во время трансляции и, таким образом, влиять на форму и содержание двоичного объектного файла. Аргументы переключателей приведены в табл. 4.

Используя данные переключатели, можно запретить действие директив исходной программы .ENABLE и .DSABL с определенными аргументами:

— переключатель /E:арг запрещает действие директивы .DSABLE арг;
— переключатель /D:арг запрещает действие директивы .ENABLE арг.

Аргументы переключателя и аргументы директивы в этом случае должны совпадать. Описание директив .ENABLE и .DSABLE см. в [2].

Пример:

*,LP:=SRCPRG.MAC/E:CDR

По данной командной строке транслируется файл SRCPRG.MAC, листинг выводится на LP:. Вся исходная информация за 72 колонкой рассматривается как комментарий.

Таблица 4

АРГУМЕНТЫ ПЕРЕКЛЮЧАТЕЛЕЙ /E:арг и D:арг

Аргумент	Действие по умолчанию	Назначение
1	2	3
ABS	Запрещает	Вывод в абсолютном двоичном формате
AMA	Запрещает	Абсолютные адреса (код метода адресации 67) транслировать как относительные (код метода адресации 37)
CDR	Запрещает	Рассматривать информацию за 72 колонкой как комментарий
CRF	Разрешает	Вывод листинга таблицы перекрестных ссылок
FPT	Запрещает	Усечение чисел с плавающей запятой без округления (/E:FPT); с округлением (/D:FPT)
GBL	Разрешает	Неопределенные имена транслировать как глобальные
LC	Разрешает	Прием входной информации, содержащей буквы русского алфавита
LCM	Запрещает	Зависимость условных директив .IF IDM и IF DIF от верхнего и нижнего регистров
LSB	Запрещает	Временно прервать блок локальных меток новой программной секцией, а затем продолжить прерванный блок локальных имен
MCL	Запрещает	Поиск макроопределения во всех макробиблиотеках, если в исходном файле встретился код неопределенной операции
PNC	Разрешает	Включение двоичных кодов в объектный модуль
REG	Разрешает	Переопределение регистров, назначенных по умолчанию

2.3.3. Переключатель определения файла макробиблиотеки (/M). Переключатель /M в командной строке следует за спецификацией файла, к которому он применяется, и указывает, что файл, за которым /M следует, является файлом макробиблиотеки. Переключатель /M не имеет аргументов.

Если командная строка не содержит спецификацию системной макробиблиотеки SYSMAC.SML система автоматически включает ее в командную строку первым входным файлом.

Если в исходном файле встречается макрокоманда, АССЕМБЛЕР просматривает все макробиблиотеки в порядке их появления в командной строке. Если макроопределение для макрокоманды с определенным именем содержится в двух или более библиотеках, макроопределение для данной макрокоманды выбирается из библиотеки, которая в строке команды была указана самой правой.

Если макробиблиотека пользователя содержит макроопределение с именем, совпадающим со стандартным именем макроопределения системной макробиблиотеки, то преимущество имеет макробиблиотека пользователя.

Пример:

*выходспф=ALIB.MLB/M,BLIB.MLB/M,XIZ

Предположим, исходный файл XIZ.MAC содержит макрокоманды .MCALL .BIG, а файлы макробиблиотек ALIB и BLIB содержат различные макроопределения макрокоманды .BIG. Система будет включать в файл расширения макрокоманды .BIG из библиотеки BLIB.

2.3.4. Переключатель управления печатью таблицы перекрестных ссылок (/C:арг).

Переключатель /C:арг предназначен для управления печатью таблицы перекрестных ссылок (CREF), т. е. переключатель /C:арг определяет содержание данной таблицы. Аргументы переключателя приведены в табл. 5. Обычно /C:арг следует в командной строке за спецификацией файла листинга, но также может быть указан в любом месте командной строки.

Таблица перекрестных ссылок (CREF) содержит все или часть имен исходной программы.

Если командная строка не содержит спецификацию файла таблицы перекрестных ссылок, система создает временный файл CREF.TMP на DK:. После того, как получена таблица перекрестных ссылок, временный файл стирается автоматически. Если необходимо вывести файл CREF на другое устройство, следует указать в командной строке уст:псспф

(уст — имя устройства, куда будет выводиться файл CREF; псспф — спецификация файла CREF). Используя команду .ASSIGN уст:CF, можно направить вывод файла CREF.TMP на нужное устройство, не указывая каждый раз при трансляции его в командной строке. Если же псспф в командной строке все же указывается, то таблица перекрестных ссылок выводится на указанное устройство файлом, имя которого задано в псспф.

Если листинг надо вывести на MT:, по команде LOAD следует загрузить драйвер MT: в память, а затем выполнять трансляцию.

Пример:

*,LP:,RK1:TEMP.TMP=SOURCE/C

По данной командной строке листинг трансляции выводится на LP:, таблица перекрестных ссылок файлом TEMP.TMP на RK1:.

Таблица 5

АРГУМЕНТЫ ПЕРЕКЛЮЧАТЕЛЯ/С

Аргумент	Раздел таблицы перекрестных ссылок
1	2
C	Имена программных секций
E	Ошибки (сгруппированные по типам ошибок)
M	Имена макрокоманд
P	Постоянные имена (включая команды и директивы)
R	Имена регистров
S	Имена, определенные пользователем

ПРИМЕЧАНИЕ. Переключатель /С без аргументов эквивалентен /C:S:M:E. Если необходимо в таблицу перекрестных ссылок включить определенные разделы, то с переключателем /С указывают соответствующие аргументы. Таблица перекрестных ссылок не создается, если не указан переключатель /С, даже если в командной строке указана спецификация файла CREF.

3. ОБРАЩЕНИЕ К ПРОГРАММЕ

Для вызова АССЕМБЛЕРа MACRO с системного устройства следует подать с терминала команду:

R MACRO <ВК>

после того, как монитор напечатает на терминале точку.

После вызова АССЕМБЛЕР печатает звездочку и ожидает ввода командной строки. Если в это время нажать клавишу <ВК>, то АССЕМБЛЕР печатает номер своей версии.

Для выхода из MACRO и передачи управления монитору следует подать команду СУ/С, если MACRO ожидает ввода с терминала, или дважды СУ/С, если MACRO выполняет операцию. Для повторного пуска MACRO следует подать команду REENTER.

После выполнения операции MACRO печатает звездочку и ожидает ввода командной строки.

ПРИМЕЧАНИЕ. Вызвать АССЕМБЛЕР MACRO можно также по команде монитора MACRO (см. [1]).

4. ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

Программа на исходном языке может состоять из одного или нескольких модулей. Каждый модуль транслируется независимо. Результатом трансляции является объектный модуль, листинг и таблица перекрестных ссылок. Объектные модули обрабатываются программой редактор связей для получения загрузочного модуля.

4.1. Исходный модуль. Исходный модуль представляет собой последовательность операторов на языке АССЕМБЛЕР. Описание форматов операторов приведено в документе [2].

4.2. Объектный модуль. Объектный модуль, полученный в результате трансляции, выдается в формате, допустимом для работы в системе ФОДОС-2. Объектный модуль состоит из отформатированных двоичных блоков.

Формат двоичного блока:

Признак начала блока (младший байт — 1)
Признак начала блока (старший байт — 0)
Количество байтов в блоке (младший байт)
Количество байтов в блоке (старший байт)
Блок информации
Байт контрольной суммы

Часть двоичного блока — «Блок информации» содержит фактически информацию объектного модуля. Система ФОДОС-2 использует 8 типов блоков информации. Табл. 6 содержит описание 8 типов блоков информации.

Таблица 6

Код типа блока	Тип блока
1	2
1	Словарь глобальных имен (GSD)
2	Конец словаря глобальных имен (ENDGSD)
3	Двоичный текст программы (TXT)
4	Словарь перемещения (RLD)
5	Словарь внутренних имен (ISD)
6	Конец объектного модуля (ENDMOD)
7	Заголовок библиотеки
10	Конец библиотечного файла

В дальнейшем двоичный блок, содержащий словарь глобальных имен, будем называть блок GSD, также и все типы блоков (блок ENDGSD, блок TXT, блок RLD, блок ISD, блок ENDMOD, блок заголовка библиотеки, блок конца библиотеки).

Формат объектного модуля:

Блок GSD
Блок RLD
Блок GSD
Блок TXT
Блок TXT
Блок RLD
. . .
Блок GSD
Блок ENDGSD
Блок ISD
Блок ISD
Блок TXT
Блок TXT
Блок TXT
Блок RLD
Блок ENDMOD

Объектный модуль должен начинаться с блока GSD и заканчиваться блоком ENDMOD. Дополнительные блоки GSD могут встречаться в любом месте в файле, но перед блоком ENDGSD. Блок ENDGSD должен быть до блока ENDMOD. По крайней мере, один блок RLD должен предшествовать блоку TXT. Дополнительные блоки TXT и RLD могут встречаться в любом месте файла. Блоки ISD также могут встречаться в любом месте файла между блоками GSD и блоками ENDMOD.

Первый байт блока информации — код типа блока (см. табл. 6).

4.2.1. Блок словаря глобальных имен (GSD)

Блок GSD содержит информацию, необходимую для присвоения адресов именам и распределения памяти задачи. Каждому имени в блоке информации соответствует запись из четырех слов, содержащая имя в коде RIDIX-50 (2 слова), код типа записи и значение имени.

Формат блока информации GSD:

0	Код типа блока информации = 1
Имя в кодах RADIX-50 2 слова	
Код типа записи	Флаг
Значение имени	
Имя в кодах RADIX-50 2 слова	
Код типа записи	Флаг
Значение имени	
. . .	
Имя в кодах RADIX-50 2 слова	
Код типа записи	Флаг
Значение имени	

Блок информации блока GSD построен таким образом, что сначала следует информация об абсолютной секции и обо всех ее глóбальных именах; затем информация об относительной секции (если она есть в программе) и перечисляются все глобальные имена данной секции. Если в программе несколько программных секций, то информация об этих секциях в блоке GSD следует в том порядке, в каком программные секции встречаются в программе. Первый блок GSD содержит информацию об имени модуля.

Типы записей и соответствующие им коды приведены в табл. 7.

Таблица 7

Код типа записи	Значение
1	2
0	Имя модуля
1	Имя секции управления (CSECT)
2	Внутреннее символическое имя
3	Адрес смещения
4	Глобальное символическое имя
5	Имя программной секции
6	Идентификация версии программы (IDENT)
7	Описание массива отображения (VSECT)

Ниже описаны все типы записей для блоков GSD.

4.2.1.1. Имя модуля (код типа записи — 0).

Запись имени модуля объявляет имя объектного модуля. В каждом объектном модуле может встречаться только одно объявление каждого модуля.

Формат записи имени модуля:

Имя модуля (в коде RADIX-50)		
0	!	0
0		

4.2.1.2. Имя секции управления (код типа записи — 1).

Запись имени секции управления объявляет имя секции управления. Редактор связей редактирует секцию управле-

ния .ASECT и .CSECT (с именем или без него) в относительную программную секцию с признаками, указанными ниже.

Формат оператора .PSECT для неименованной относительной секции:

.PSECT, RW,I, LCL, REL, CON

Формат оператора .PSECT для именованной относительной секции:

.PSECT имя, RW, I, GBL, REL, OUR

Формат оператора .PSECT для абсолютной секции:

.PSECT .ABS., RW, I, GBL, ABS, OUR

Формат записи имени секции управления:

Имя секции управления (в коде RADIX-50)	
1	Не используется
Максимальная длина	

4.2.1.3. Внутреннее имя (код типа записи — 2).

Запись внутреннего символического имени объявляет имя, внутреннее по отношению к модулю. Т. к. редактор связей не создает таблицу внутренних имен, то при редактировании записи внутренних имен игнорируются редактором связей.

Формат записи внутреннего имени:

Внутреннее имя (в коде RADIX-50)	
2	0
Не определено	

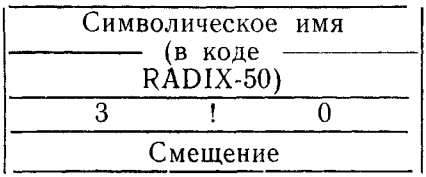
4.2.1.4. Адрес смещения (код типа записи — 3).

Запись адреса смещения объявляет адрес смещения модуля относительно программной секции. Первые два слова записи определяют имя относительной секции. Четвертое слово указывает относительное смещение от начала относительной программной секции.

Редактор связей передает четный адрес смещения, который в программе встречается первым (с директивой .END), системе как пусковой адрес программы. Если в программе не указан адрес смещения (в этом случае смещение — 000001), или если указан нечетный адрес смещения — программа при

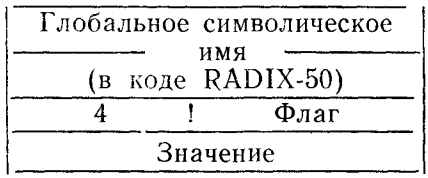
загрузке не запускается (запуск может быть осуществлен по команде монитора START).

Формат записи адреса смещения:



4.2.1.5. Глобальное символическое имя (код типа записи — 4). Запись глобального символического имени объявляет или об обращении к глобальному имени, или об определении глобального имени (запись определения глобального имени). Все записи определения глобальных имен должны следовать в блоке информации блока GSD за записью имени программной секции, но перед записью имени следующей программной секции. Записи обращения к глобальным именам могут быть в любом месте внутри блока информации блока GSD.

Формат записи глобального символического имени:



В записи глобального имени: первые два слова — код RADIX-50 глобального имени; четвертое слово — значение имени относительно программной секции, в которой имя определено.

Байт флага третьего слова определяет обращение к глобальному имени, или указывает, что определение глобального имени, абсолютное или относительное, встретилось в данной программной секции.

4.2.1.6. Имя программной секции (код типа записи — 5). Запись имени программной секции объявляет имя программной секции и ее максимальную длину в модуле. Байт флага используется для объявления определенных свойств программной секции (размер секции, в оверлейном или кор-

невом сегменте расположена секция, вид доступа к программной секции).

Формат оператора .PSECT для программной секции:
.PSECT , RW, I, LCL, REL, CON

Формат записи имени программной секции:

Имя программной секции (в кодах RADIX-50)		
5	!	Флаг
Максимальная длина		

ПРИМЕЧАНИЕ. Для всех абсолютных секций 4-е слово содержит 0.

4.2.1.7. Идентификация версии программы (код типа записи — 6). Запись идентификации версии программы объявляет версию модуля. Если исходная программа содержит директиву .IDENT, то блок информации блока GSD будет содержать запись идентификации версии программы. Первые два слова записи содержат идентификацию версии. Четвертое слово и байт флага третьего слова редактором связей не используется.

Формат записи идентификации версии программы:

Идентификация версии программы (в кодах RADIX-50)		
6	!	0
0		

4.2.1.8. Имя отображенного массива (код типа записи — 7).

Запись имени отображенного массива объявляет имя отображенного массива, т. е. распределяет память внутри области задания, для отображения массива. Байт флага записи имени отображенного массива резервируется и предполагается, что его содержимое = 0.

Четвертое слово, длина, содержит число 32-х словных блоков. Если длина равна 0, сегмент — корневой. Не должно быть глобальных имен в секции, адрес базы которой = 0.

Формат оператора .PSECT для программной секции (VSECT):
.PSECT .VIR., RW, D, GBL, REL, CON

Формат записи имени отображенного массива:

Имя отображенного массива (в кодах RADIX-50)	
7 !	Резервируется
Длина	

4.2.2. Блок конца словаря глобальных имен (ENDGSD). Блок ENDGSD указывает, что в данном объектном модуле больше не содержится блоков GSD. В каждом объектном модуле должен быть только один блок ENDGSD. Длина блока информации блока ENDGSD — одно слово.

Формат блока ENDGSD:

! 0 ! Код блока ENDGSD=2 !

4.2.3. Блок двоичного текста программы (TXT). Блок TXT содержит текст программы, который должен быть записан непосредственно в загрузочный модуль.

Формат блока TXT:

0	Код блока TXT=3
Адрес загрузки	
Текст	Текст
Текст	Текст
Текст	Текст
. . .	
Текст	Текст
Текст	Текст
Текст	Текст

Блок TXT содержит адрес загрузки, слова (байты) текстовой информации. Конечное значение некоторых слов (байт) может быть еще не определено, их значения изменяются в соответствии с информацией блока RLD. Если значения некоторых слов (байт) блока TXT не определены, то после блока TXT должен следовать блок RLD. Если же все значения блока TXT определены, блок RLD не требуется после блока TXT. Однако, по крайней мере, один блок RLD должен предшествовать первому блоку TXT.

4.2.4. Блок словаря перемещения (RLD). Блок словаря перемещения содержит информацию, необходимую редактору связей для распределения и связывания блоков TXT.

Формат блока RLD:

0	Код блока RLD=4
Байт смещения	Командный байт
Инф	Инф
Инф	Инф
. . .	
Командный байт	Инф
Инф	Байт смещения
Инф	Инф
Инф	Инф
Байт смещения	Командный байт
Инф	Инф
Инф	Инф

Блок RLD содержит командный байт (указывающий код типа команды, т. е. правило, по которому будут вычисляться

значения неопределенных слов (байтов) предшествующего блока TXT), байт смещения и информацию, которая требуется для соответствующего типа команд.

4.2.5. Блок словаря внутренних имен (ISD).

Блок словаря внутренних имен объявляет определения внутренних имен. Редактор связей системы ФОДОС-2 не поддерживает этот блок. Поэтому ниже приведен не детальный формат блока ISD. Редактор связей игнорирует блок ISD.

Формат блока ISD:

0	Код блока ISD=5
Не указано	

4.2.6. Блок конца объектного модуля (ENDMOD).

Блок конца объектного модуля объявляет конец объектного модуля. Блок ENDMOD должен быть последним в каждом объектном модуле.

Формат блока ENDMOD:

0	Код блока ENDMOD=6

Длина блока ENDMOD — одно слово.

4.3. Листинг. Листинг представляет собой результат трансляции программы, который выводится на терминал или на периферийное устройство и содержит программу в исходном и объектном виде, сообщения об ошибках и таблицу имен.

Листинг выдается по страницам. На рис. 1 каждое поле листинга помечено цифрой.

1 поле оглавления

первая строка — информационная. Она содержит:

— наименование объектного модуля;
— наименование АССЕМБЛЕРа и его версию;
— дату (число, месяц, год);
— время дня (час, минута, секунда).

Последующие строки оглавления содержат подзаголовки с указанием номера страницы и номера строки соответствующей директивы .SBTTL.

2 поле заголовка страницы содержит информационную строку с указанием номера страницы листинга и подзаголовок, определенный директивой .SBTTL.

3 поле ошибки заполняется в случае обнаружения ошибки, содержит не более четырех сообщений на строке (см. табл. 8).

4 поле номера строки содержит порядковый номер строки исходной программы.

5 поле счетчика адреса содержит восьмеричный адрес оператора.

6 поле команды содержит объектные коды в восьмеричном виде.

7 поле модификации содержит признак модифицируемых данных:

G — глобальное имя; C — перемещаемое имя;

' (апостроф) — операнд.

8 поле оператора содержит операторы исходной программы.

9 поле таблицы имен содержит имена пользователя в алфавитном порядке и список программных секций в порядке появления их в программе.

Для имен пользователя указывается:

1) имя пользователя;

2) значение имени (******, если имя не определено);

3) признак имени:

пробел — абсолютное

R — относительное;

X — внешнее;

G — глобальное.

Для программных секций указывается:

1) имя секции;

2) длина секции;

3) номер секции:

00 — абсолютной;

01 — неименованной перемещаемой;

02 и т. д. — именованной.

10 поле командной строки повторяет введенную командную строку АССЕМБЛЕРа.

11 поле строки ошибок содержит общее число ошибок, обнаруженных в исходной программе.

Формат листинга можно изменить, задав в командной строке переключатель управления листингом /L:арг или /N:арг, см. п. 2.3.1.

4.4. Таблица перекрестных ссылок.

Таблица перекрестных ссылок печатается за листингом. Полная распечатка таблицы перекрестных ссылок состоит из шести полей (см. рис. 2). Каждое поле начинается с новой страницы.

1 Имена пользователя
2 Имена регистров
3 Имена макрокоманд
4 Постоянные имена
5 Имена программных секций
6 Ошибки

Ссылки печатаются в виде P — L, где P — номер страницы, в которой появляется имя или код ошибки, а L — номер строки. Знак «#» вслед за ссылкой указывает на определе-

```
                Фрагмент листинга, полученного
                с использованием переключателя  /L:TTM

1  TAIP  MACRO  B03.003-JAN-85   00:08:58
   TABLE  OF  CONTENTS

-------------------------------------------------------------------------------

       1-   2       KONTROL PRIMER
2  TAIP     MACRO B03.00  00:08:58 PAGE 1
-------------------------------------------------------------------------------
3     !     4!  5   !   6  !7!          8
      !     1!      !      ! !         .TITLE  TAIP
      !     2!      !      ! !         .SBTTL  KONTROL PRIMER
      !     3!      !      ! !         .GLOBL  SUBR1,SUBR2
      !     4!000000!      ! !         .CSECT  PROG
      !     5!      !      ! !         .MCALL  .TTYIN
      !     6!000000!012702! !START:   MOV     #BUFFER,R2
      !      !      !000026!'!
      !     7!000004!110022! !         MOVB    R0,(R2)+
4     !     8!000006!120027! !        CMPB    R0,#LF
      !      !      !000000! !
      !     9!000012!105022! !         CLRB    (R2)+
      !    10!000014!004767! !         JSR     PC,SUBR1
      !      !      !000000!G!
      !    11!000020!103767! !        BCS     START
      !    12!000022!      ! !         .TTYIN
      !    13!000026!      ! !BUFFER:  .BLKB   72
      !    14!      !000000!'!         .END    START
-------------------------------------------------------------------------------

7  TAIP    MACRO B03.00 3-JAN-85  00:08:58 PAGE 1-1
   SYMBOL TABLE
   BUFFER 000026R   002 START   000000R   002 SUBR2 = ****** G
   LF     = ******        SUBR1 = ****** G
   . ABS.  000000      000
           000000      001
   PROG    000120      002

-------------------------------------------------------------------------------
11 ERRORS DETECTED:  1
-------------------------------------------------------------------------------
9  VIRTUAL MEMORY USED:  367 WORDS  ( 2 PAGES)
   DYNAMIC MEMORY AVAILABLE FOR  68 PAGES
-------------------------------------------------------------------------------
10 ,DK0:PR.LST/L:TTM=PROG1
-------------------------------------------------------------------------------
```

Рис. 1

ние имени. Знак «*» соответствует имени, изменяющему свое значение.

На рис. 2 приведен пример таблицы перекрестных ссылок:

```
1  TAIR    MACRO B03.00  00:12:27 PAGE S-1
   CROSS REFERENCE TABLE (CREF B03.00 )
   BUFFER    1-6       1-13#
   LF        1-8
   START     1-6#      1-11      1-14
   SUBR1     1-3       1-10
   SUBR2     1-3
2  TAIP    MACRO B03.00  00:12:27 PAGE R-1
   CROSS REFERENCE TABLE (CREF B03.00 )
   PC        1-10*
   R0        1-7       1-8
   R2        1-6*      1-7*      1-9*
3  TAIP    MACRO B02.00  00:12:2 PAGE M-1
   CROSS REFERENCE TABLE (CREF B03.00 )
   .TTYIN    1-5*      1-12
4  TAIP    MACRO B03.00  00:12:27 PAGE P-1
   CROSS REFERENCE TABLE (CREF B03.00 )
   .BLKB     1-13
   .CSECT    1-4
   .END      1-14
   .GLOBL    1-3
   .IF       1-12
   .MCALL    1-5
   .SBTTL    1-2
   .TITLE    1-1
   BCS       1-11      1-12
   CLRB      1-9
   CMPB      1-8
   EMT       1-12
   JSR       1-10
   MOV       1-6
   MOVB      1-7
5  TAIP    MACRO B03.00  00:12:27 PAGE C-1
   CROSS REFERENCE TABLE (CREF B03.00 )
             0-0
   .ABS.     0-0
   PROG      1-4
6  TAIP    MACRO B03.00  00:12:27 PAGE E-1
   CROSS REFERENCE TABLE (CREF B03.00 )
   U         1-8*
```

Рис. 2

5. СООБЩЕНИЯ

5.1. Сообщения программисту.

Ошибки, обнаруженные в исходной программе, отмечаются в листинге соответствующими сообщениями. Сообщения печатаются в поле ошибок (см. п. 4.3). Для исправления

ошибок необходимо внести изменения в исходную программу и повторно ее протранслировать.

Таблица 8

СООБЩЕНИЯ ПРОГРАММИСТУ

Сообщение	Причина
1	2
A	Ошибка адресации или перемещения: — превышена допустимая величина смещения в команде условного ветвления (т е от —128 (десятичное) до 127 (десятичное)) — оператор неправильно изменяет счетчик текущего адреса (например, оператор превысил счетчик текущего адреса за границы директивы PSECT) — оператор содержит недопустимое выражение, определяющее адрес (например, абсолютное выражение содержит глобальное имя, перемещаемый терм или составной перемещаемый терм) Директивы АССЕМБЛЕРа BLKB, BLKW и REPT должны содержать абсолютное значение или выражение — использование нескольких выражений, не отделенных друг от друга запятыми — ошибка определения глобального имени Если исходная программа содержит директиву ENABL GBL, АССЕМБЛЕР просматривает в конце первого прохода таблицу имен и предполагает все неопределенные имена глобальными Если какое нибудь из этих имен определяется при втором проходе, происходит общая ошибка адресации — недопустимые ссылки вперед 1) оператор прямого присваивания имя=выражение содержит ссылку вперед 2) выражение, определяющее счетчик адреса, содержит ссылку вперед — директива АССЕМБЛЕРа в исходной программе содержит недопустимый аргумент, недопустимый ограничитель или недопустимую конструкцию аргументов
B	Значение счетчика адреса нечетно, во время трансляции значение счетчика адреса увеличивается на 1
D	Обращение к многократно определенному имени
E	Отсутствует директива END в конце исходной программы. Система завершает текущий проход трансляции
I	Недопустимый символ Недопустимый символ в листинге заменяется знаком вопроса(?), символ игнорируется, трансляция выполняется
L	Исходная строка содержит более 132 символов (например, при замене формальных параметров фактическими в макрорасширениях)

1	2
M	Многократное определение метки. Метка эквивалентна по первым шести символам ранее встреченной метке
N	Константа содержит цифру, которая не входит в текущую систему счисления программы, константа рассматривается как десятичная
O	Ошибка в поле операции: — директива вне контекста — превышен допустимый уровень вложения для директив условной трансляции — не найдено макроопределение, заданное директивой .MCALL
P	— значение метки меняется от прохода к проходу — в блоке локальных имен появляется многократное определение локального имени
Q	Синтаксическая ошибка в операторе: — пропущен аргумент — указан лишний аргумент — не закончен просмотр оператора
R	Ошибка регистра: — недопустимое обращение к регистру — попытка переопределить стандартное имя регистра без использования директивы .DSABL REG
T	Ошибка усечения: — сформированное число занимает более 16 разрядов — значение выражения содержит более 8 разрядов для директивы .BYTE или команды прерывания (EMT или TRAP)
U	Неопределенное имя. Неопределенному имени присваивается нулевое значение
Z	Выполнение команды различно в ЭВМ «ЭЛЕКТРОНИКА-60», «ЭЛЕКТРОНИКА-100-25», «ЭЛЕКТРОНИКА-79» (Например, двухадресные команды, имеющие адресацию RN, (RN)+; RN, —(RN); команды JMP и JSR с автоинкрементным методом адресации)

5.2. Сообщения оператору.

Сообщения, приведенные ниже, указывают на преждевременное прекращение трансляции.

?MACRO—F—DEVICE FULL DEV

Причина. На томе, используемом для вывода, недостаточно свободного места для размещения выходного файла.

Действие. Освободить место на томе или использовать для вывода другой том.

?MACRO—F—FILE NOT FOUND
DEV:FILNAM.TYP

Причина. Файл, указанный в командной строке, не найден.

Действие. Проверить, существует ли файл с указанным именем. Ввести правильную командную строку.

?MACRO—F—.INCLUDE DIRECTIVE FILE ERROR

Причина. Файл, указанный в директиве .INCLUDE, не существует, или в директиве указана недопустимая спецификация файла. В командной строке указано недопустимое имя устройства. Уровень вложения исходных файлов по директиве .INCLUDE превышает 5.

Действие. Проверить введенную командную строку и ввести допустимую для директивы .INCLUDE спецификацию файла. Вновь ввести командную строку, указав допустимое имя устройства. Убедиться, что уровень вложения исходных файлов не превышает 5.

?MACRO—F—INSUFFICIENT MEMORY

Причина. Конфигурация вычислительной системы имеет объем памяти менее 32К байт, необходимый для выполнения программы.

Действие. Увеличить объем свободной памяти (разгрузить драйверы неиспользуемых в данный момент устройств; завершить выполнение основного или системного задания и удалить его; использовать монитор одного задания SJ; разрешить свопинг по команде SET USR SWAP; уменьшить размеры программы, уменьшив максимальное число каналов, открытых одновременно, или разбив программу на небольшие модули для увеличения оверлейного эффекта, или использовав алгоритм, требующий минимальный объем памяти, или записав данные на периферийные устройства).

?MACRO—F—INVALID COMMAND

Причина. Командная строка содержит синтаксическую ошибку или более шести спецификаций файлов.

Действие. Ввести правильную командную строку.

?MACRO—F—INVALID DEVICE DEV:

Причина. Указанное устройство не обслуживается системой.

Действие. Установить в системе обслуживание требуемого устройства либо использовать другое устройство.

?MACRO—F—INVALID MACRO LIBRARY

Причина. Файл макробиблиотеки поврежден или создан не библиотекарем системы ФОДОС-2.

Действие. Использовать программу LIBR для создания новой копии SYSMAC.SML.

?MACRO—F—INVALID OPTION: /X

Причина. Указан недопустимый переключатель /X.

Действие. Проверить переключатель, указанный в командной строке, и ввести правильную командную строку.

?MACRO—F—I/O ERROR ON DEV:FILNAM.TYP

Причина. Ошибка при считывании или записи указанного файла.

Действие. Проверить исправность и готовность оборудования.

?MACRO—F—I/O ERROR ON WORKFILE

Причина. Ошибка при считывании или записи рабочего файла WRK.TMP. Эта ошибка может произойти при недостаточной области на диске для размещения рабочего файла.

Действие. Проверить исправность и готовность оборудования и использовать команду SQUEEZE или переключатель /S программы DUP для сжатия тома и размещения рабочего файла.

?MACRO—F—.LIBRARY DIRECTIVE FILE ERROR

Причина. Файл, указанный в директиве .LIBRARY, не существует или в директиве указана недопустимая спецификация файла. Спецификация файла, указанная в директиве .LIBRARY, содержит имя устройства последовательного доступа. Конфигурация вычислительной системы не содержит устройство, указанное в командной строке. Уровень вложения файлов по директиве .LIBRARY превысил допустимый.

Действие. Проверить введенную командную строку и ввести допустимую для директивы .LIBRARY спецификацию файла. Убедиться, что спецификация файла, указанная в директиве .LIBRARY, содержит имя устройства произвольного доступа. Ввести новую командную строку, содержащую имя устройства конфигурации данной вычислительной системы. Проверить, чтобы уровень вложения файлов по директиве .LIBRARY не превысил допустимое значение.

?MACRO—F—PROTECTED FILE ALREADY EXISTS DEV:FILNAM.TYP

Причина. Попытка создать файл с таким же именем, как у имеющегося защищенного файла.

Действие. Отменить защиту существующего файла по команде монитора UNPROTECT или по переключателю /Z программы PIP, или указать другое имя для создаваемого файла.

?MACRO—F—STORAGE LIMIT EXCEEDED (64K)

Причина. Программа требует более 128K байт памяти для хранения таблицы виртуальных имен, в то время как система отводит данной таблице не более 128K байт.

Действие. Проверить причины, вызывающие переполнение таблицы виртуальных имен. Разделить исходную программу на независимые модули, и транслировать каждый модуль отдельно.

?MACRO—W—I/O ERROR ON CREF FILE:CREF ABORTED

Причина. На выходном томе недостаточно свободного места для выполнения операции, или во время записи на том рабочего файла CREF произошла ошибка ввода — вывода. Вывод файла CREF прерывается, но трансляция продолжается.

Действие. Увеличить на томе объем свободной памяти (удалить с тома ненужные файлы; использовать переключатель /ALLOCATE для резервирования нужного количества блоков для выходного файла; сжать том по команде монитора SQUEEZE или по переключателю /S программы DUP; переписать часть сегментов справочника на другой том). Создать несколько логических дисков на томе, используя команды MOUNT и DISMOUNT. Если переполнение тома все еще сохраняется после принятых мер, использовать том большей емкости.

ПЕРЕЧЕНЬ ССЫЛОЧНЫХ ДОКУМЕНТОВ

1. Операционная система ФОДОС-2
   Командный язык системы
2. Операционная система ФОДОС-2
   АССЕМБЛЕР. Описание языка

РЕДАКТОР СВЯЗЕЙ

РУКОВОДСТВО ОПЕРАТОРА

1. ОПРЕДЕЛЕНИЯ

Ниже даются определения некоторых терминов, часто используемых в настоящем документе.

Программная секция — именованная секция, которая состоит из смежных блоков команд или данных, рассматривается как целое и может быть перемещена отдельно, что не нарушит логику программы. Может также называться П-секцией.

Объектный модуль — основные выходные данные АССЕМБЛЕРа или компилятора; могут быть скомпонованы с другими модулями и загружены в память в виде готовой к выполнению программы. Объектный модуль состоит из перемещаемого машинного кода, информации о перемещении и соответствующей таблицы глобальных имен, определяющей использование символов в программе.

Загрузочный модуль — программа в готовом для загрузки и выполнения формате.

Библиотечный файл — файл, содержащий один или несколько перемещаемых объектных модулей — стандартных подпрограмм — которые могут быть включены в другие программы.

Библиотечный модуль — модуль из библиотечного файла.

Корневой сегмент — сегмент оверлейной структуры, который после загрузки остается резидентным в памяти во время выполнения программы.

Оверлейный сегмент — секция команд или данных, рассматриваемая как целое, которая может накладываться на расположенные в памяти команды или данные и на которую может накладываться другой оверлейный сегмент при вызове из корневого или другого оверлейного сегмента. Далее в тексте — оверлей.

Глобальный символ — глобальное значение или глобальная метка.

Нижняя память — физическая память от 0 до 28К слов.

Расширенная память — физическая память свыше 28К слов.

Составное перемещение — перемещение, в котором любые двоичные или унарные операции АССЕМБЛЕРа разрешены с любым типом аргумента (неразрешенные глобальные имена, перемещаемые с любой базой П-секции, абсолютные или составные перемещаемые подвыражения).

2. НАЗНАЧЕНИЕ ПРОГРАММЫ И УСЛОВИЯ ВЫПОЛНЕНИЯ ПРОГРАММЫ

Основной задачей редактора связей является формирование готовых к выполнению программ. Входной информацией для редактора связей являются объектные и библиотечные модули. Редактор связей обрабатывает эти модули и формирует загрузочный модуль. При формировании загрузочного модуля редактор связей **выполняет следующие функции:**

1) присваивает абсолютные адреса;

2) определяет внешние связи между модулями;

3) создает блок управления, используемый при загрузке программ;

4) если указано, формирует загрузочный модуль оверлейной структуры;

5) включает в загрузочный модуль требуемые модули из системной библиотеки и библиотек пользователя;

6) если указано, создает карту загрузки;

7) если указано, создает файл определений имен.

Редактор связей формирует загрузочный модуль за два прохода. Во время первого прохода создается таблица имен, в которую входят все имена программных секций и глобальные имена входных модулей, и обрабатываются модули кор-

невого сегмента (часть программы, постоянно находящаяся в памяти). Во время второго прохода редактирование завершается и формируется загрузочный модуль. Загрузочный модуль может быть получен в одном из трех форматов: в формате отображения памяти (SAV) для заданий, выполняемых под управлением монитора одного задания, или для фоновых заданий, выполняемых под управлением монитора основного — фонового задания; в перемещаемом формате отображения памяти (REL) для основных заданий, выполняемых под управлением монитора основного — фонового задания или расширенной памяти; в абсолютном двоичном формате (LDA) для использования в перфоленточной операционной системе.

Редактор связей может работать в системах с минимальным объемом оперативной памяти (16К слов). Дополнительная память используется для повышения скорости редактирования и увеличения размера таблицы имен, формируемой редактором связей.

3. ОБЩИЕ ПОНЯТИЯ

Когда редактор связей обрабатывает оттранслированные или скомпилированные объектные модули, он создает загрузочный модуль, в котором присвоены все абсолютные адреса и отведена память под программные секции.

3.1. Абсолютная секция. Абсолютная секция часто носит название ASECT, потому что объявляется директивой АССЕМБЛЕРа .ASECT. В карте загрузки абсолютная секция всегда идет первой с именем .ABS. Абсолютная секция обычно заканчивается адресом 1000 (восьмеричное) и содержит следующую информацию:

— область связи с системой;
— векторы прерываний;
— стек пользователя.

Область связи с системой располагается в ячейках 0—377 и содержит данные, которые редактор связей использует для передачи управляющих параметров программы, и карту использования памяти (см. п. 5.2).

Стек — это область памяти, которую программа может использовать для временного хранения данных и связи с подпрограммами. Обращение к стеку осуществляется через регистр общего назначения R6 — указатель стека (SP).

3.2. Программные секции. Программные секции следуют за абсолютной секцией. Набор признаков каждой секции контролирует распределение памяти и размещение секции в загрузочном модуле.

При обработке программной секции как единицы памяти программы редактор связей использует следующие параметры:

— имя секции;

— набор признаков, определяющих тип, режим доступа, распределение памяти, размещение секции в загрузочном модуле (см. табл. 1);

— размер, определяющий необходимый секции объем памяти.

Создать П-секции можно с помощью оператора COMMON в ФОРТРАНе или директивы .PSECT или .CSECT в MACRO. Директиву .PSECT или .CSECT можно использовать для указания признаков секции.

Следует отметить, что признаки, следующие за именем П-секции в карте загрузки, не являются частью имени; только имя отличает одну П-секцию от другой. Поэтому, если требуется скомпоновать вместе П-секции с одинаковыми именами, то и перечни признаков у них должны быть одинаковыми. Если редактору связей попадаются П-секции с одинаковыми именами, но разными признаками, то печатается предупреждение; в этом случае используются признаки из той П-секции, которая встретилась первой.

Таблица 1

Признак	Значение	Назначение
1	2	3
Доступ	RW	Разрешает чтение и запись
	RO	Разрешает только чтение
Тип	I	Программная секция содержит команды и данные
	D	Программная секция содержит данные
Размещение	GBL	Имя программной секции распознается за границами оверлейного сегмента. Редактор связей распределяет память для программной секции в корневом сегменте, если к программной секции обращаются из нескольких оверлейных сегментов. Если обращение к программной секции происходит из одного сегмента, редактор связей распределяет память для программной секции в этом оверлейном сегменте
	LCL	Имя программной секции распознается внутри оверлейного сегмента
	SAV	Имя программной секции распознается за границами оверлейного сегмента. Редактор связей всегда распределяет память для программной секции в корневом сегменте

1	2	3
Перемещение	REL	Базовый адрес программной секции перемещается относительно базового адреса программы
	ABS	Базовый адрес программной секции равен 0
Распределение	CON	Распределения памяти, относящиеся к программной секции, располагаются одно за другим. Размер выделяемой области равен сумме отдельных распределений
	OVR	Распределения памяти, относящиеся к программной секции, перекрываются. Размер выделяемой области памяти равен размеру наибольшего распределения

ПРИМЕЧАНИЕ. В операционной системе ФОДОС признак доступа не используется.

Признаки размещения имеют смысл только для программ оверлейной структуры. Если признак размещения программной секции имеет значение GBL, то такая секция (глобальная секция) компонуется из модулей всей программы. Если глобальная секция размещена в модулях, входящих в разные сегменты, то редактор связей помещает ее в корневой сегмент. Если признак размещения программной секции имеет значение LCL (локальная секция), то для каждого сегмента эта секция компонуется только из модулей, входящих в данный сегмент. Поэтому в разных сегментах могут быть разные программные секции с одним и тем же именем.

Значение признака распределения определяет адрес загрузки и объем памяти, отводимый программной секции. Если признак распределения программной секции имеет значение OVR, то адрес загрузки для всех частей этой секции, находящихся в разных модулях, будет один и тот же. Размер отводимой области памяти равен размеру наибольшей части этой секции. После редактирования в этой области загрузочного модуля будут находиться данные из последнего объектного модуля, содержащего данную секцию. Если признак распределения программной секции имеет значение CON, то все части этой секции, находящиеся в разных модулях, редактор связей размещает в загрузочном модуле последовательно друг за другом. Объем отводимой области памяти равен сумме размеров всех частей данной секции.

Любая П-секция с признаком D, содержащая обращения

к меткам, значение которых четно, должна начинаться с четного адреса. Это достигается с помощью директивы АССЕМБЛЕРа .EVEN в конце П-секции с признаком CON каждого модуля. В противном случае редактор связей может выдать сообщение:

?LINK—F—WORD RELOCATION ERROR IN FILNAM

и редактирование связей не выполняется.

Если для программной секции признаки типа и распределения имеют значения I и CON соответственно, то для всех частей этой секции, находящихся в разных модулях, редактор связей отводит память, начиная с ближайшего четного адреса. Если признаки типа и распределения имеют значения D и CON соответственно, то редактор связей отводит память, начиная с первого свободного байта. При этом может оказаться, что адрес этого байта нечетный.

АССЕМБЛЕР и редактор связей преобразуют директиву АССЕМБЛЕРа .CSECT в эквивалентную директиву .PSECT с фиксированными признаками.

Неименованная .CSECT эквивалентна неименованной .PSECT с признаками RW, I, LCL, REL и CON.

Именованная .CSECT эквивалентна именованной .PSECT с признаками RW, I, GBL, REL и OVR. Эти секции и их признаки приведены в табл. 2.

Таблица 2

Секция	Признак доступа	Признак типа	Признак размещения	Признак перемещения	Признак распределения
1	2	3	4	5	6
CSECT	RW	I	LCL	REL	CON
Именованная CSECT	RW	I	GBL	REL	OVR
ASECT (.ABS)	RW	I	GBL	ABS	OVR
Именованная COMMON	RW	D	GBL	REL	OVR
VSECT (.VIR)	RW	D	GBL	REL	CON

Имена, присвоенные П-секциям, глобальными именами не являются, поэтому обращаться к ним как к именам нельзя.

Например:

```
MOV #PNAME,R0
```

Эта команда (PNAME — имя П-секции) неверна, и если не существует глобального символа PNAME, то редактор связей выдаст сообщение о неопределенном глобальном имени.

Имя П-секции и глобальное имя могут совпадать, но обрабатывает их редактор связей по-разному.

3.2.1. Порядок программных секций.

Редактор связей распределяет память для П-секций в том порядке, в каком они идут во входных модулях. В табл. 3 показано, в каком порядке следуют П-секции в файлах оверлейной структуры (и неоверлейной структуры).

Таблица 3

Файлы неоверлейной структуры	Файлы оверлейной структуры
1	2
Абсолютная (.ABS) Неименованная Именованная (NAME)	Абсолютная (.ABS) Оверлейный драйвер (¤OHAND) Оверлейная таблица (¤OTABL) неименованная именованная (NAME)

Если именованных секций несколько, то они располагаются в том порядке, в каком идут во входных файлах. Например, компилятор ФОРТРАНа размещает П-секции в основном модуле программы, так что свопинг USR может осуществляться на область команд и данных в нижней памяти, а не на область данных, которые необходимы для функции, вызвавшей USR.

Если размер неименованной П-секции — нулевой, то она в карте загрузки не указывается.

3.3. Глобальные имена.

Глобальные имена обеспечивают связь между модулями программы. Рассмотрим на примере алгоритм, по которому редактор связей обрабатывает глобальные имена. Пусть требуется объединить три объектных модуля, сведения о которых приведены в табл. 4.

Таблица 4

Модуль	Определение глобального имени	Обращение к глобальному имени
1	2	3
M1	B1, B2	A, B, C, X
M2	A, B1	B2
M3		B1

Обрабатывая модуль M1, редактор связей находит определение глобальных имен B1, B2 и обращение к глобальным именам A, B, C и X. Поскольку для A, B, C и X нет определений, то разрешение этих глобальных имен откладывается. Обрабатывая модуль M2, редактор связей находит определение A, что определяет обращение в модуле M1, и обращение к B2, которое разрешается непосредственно.

После обработки всех модулей остаются три неопределенных глобальных имени — B, C и X. Предположим, что X разрешено в результате поиска в системной библиотеке. Глобальные имена B и C остаются неопределенными, и на терминал будет выдано соответствующее сообщение.

Глобальное имя B1 определено дважды, и на терминал будет выдано сообщение об этом. Редактор связей использует первое встреченное определение этого глобального имени. Редактор связей не выдает сообщений о многократном определении глобального имени, которое при каждом определении получает одно и то же абсолютное значение.

Более подробно об использовании глобальных имен см. в [1].

3.4. Оверлейные программы.

Возможность создания оверлейных программ предоставляет пользователю практически неограниченную виртуальную память.

Для создания оверлейной структуры программу необходимо сегментировать, т. е. разделить на части (сегменты). Сегменты хранятся во внешней памяти и по мере необходимости вызываются в оперативную память для выполнения.

Редактор связей создает программы с оверлеями в нижней памяти (физическая память от 4 до 28K слов) и расширенной памяти (физическая память свыше 28K слов) в зависимости от того, в какую область памяти (нижнюю или расширенную) вызываются сегменты.

Программы с оверлеями в расширенной памяти предполагают наличие в системе расширенной памяти и монитора XM.

3.4.1. Оверлеи в нижней памяти.

Редактор связей создает оверлейную структуру с корневым сегментом, который всегда резидентен в оперативной памяти. Корневой сегмент является необходимой частью каждой оверлейной программы. Он содержит точку входа редактируемой программы, область стека, смешанные переменные и переменные, необходимые для многих сегментов. Поэтому корневой сегмент никогда не перекрывается другими сегментами.

Оверлейные сегменты пользователь группирует в зависимости от конкретной задачи. Каждой группе сегментов ставится в соответствие отдельная область оперативной памяти — так называемая оверлейная область. Редактор связей определяет размер этой области, равный размеру наибольшего оверлейного сегмента, соответствующего этой области. Во время выполнения программы сегменты, соответствующие данной оверлейной области, по мере необходимости загружаются в эту область, т. е. в одни и те же ячейки оперативной памяти.

Для создания программ с оверлеями в нижней памяти используется переключатель /O:N, где N — номер оверлейной области (см. п. 5.2.14).

Для обеспечения выполнения программ с оверлеями как в нижней, так и в расширенной памяти, редактор связей включает в загрузочный модуль небольшую подпрограмму (оверлейный драйвер) и необходимые для ее работы таблицы. Драйвер и таблицы размещаются в загрузочном модуле, начиная с младшего адреса программы.

При создании программ с оверлеями как в нижней, так и в расширенной памяти должны соблюдаться следующие правила:

1) для создания оверлейной структуры необходимо наличие системной библиотеки SYSLIB, т. к. она содержит оверлейный драйвер;

2) оверлейные сегменты, предназначенные для одной области, должны быть логически независимыми, т. е. компоненты одного сегмента не могут обращаться к компонентам другого сегмента, предназначенного для той же области;

3) глобальная программная секция, к которой обращаются несколько сегментов, размещается в корневом сегменте редактором связей (например, блоки COMMON или .PSECT с размещением GBL);

4) объектный модуль из библиотечного файла будет автоматически размещен в оверлейном сегменте, если к нему обращается только этот сегмент. Если к библиотечному файлу обращаются несколько сегментов, редактор связей размещает его в корневом сегменте (если не указан переключатель /D, (см. п. 5.2.4). Редактор связей включает также в корневой сегмент все модули из библиотеки кратных определений и модули, указанные с помощью переключателя /I (см. п. 5.2.9);

5) все блоки COMMON, которые инициализируются опе-

ратором DATA, должны быть подобным образом инициализированы в сегменте, в который они помещены;

6) когда происходит обращение к оверлеям, весь путь возврата должен находиться в памяти. Это имеет место, если выполнены следующие правила:

— из оверлейного сегмента можно обращаться (с ожидаемым возвратом) к элементам того же сегмента, к корневому сегменту или к оверлейным сегментам с большими номерами областей;

— переходы к оверлейным сегментам (без ожидаемого возврата) можно осуществлять только в точки входа этих сегментов; однако при переходах нельзя обращаться к областям оверлеев с номерами меньшими, чем номер области, из которой было произведено последнее невозвращенное сообщение. Так, если обращение было осуществлено из третьей области, то нельзя обращаться к областям 1, 2 и 3 до тех пор, пока не будет произведен возврат в третью область;

— обращения внутри сегмента области допускаются, а обращения к другому сегменту этой области не допускаются;

7) передавать управление в оверлейные сегменты можно только в точки входа этих сегментов. Точкой входа оверлейного сегмента является глобальное имя, определенное в программной секции, для которой признак типа имеет значение I. Такие имена помечаются в карте загрузки символом '@'. Например, если ENTER — глобальный символ в оверлейном сегменте, то первая команда действительна, а вторая недопустима:

```
JMP ENTER
JMP ENTER+6
```

8) использовать точки входа оверлейных сегментов можно только для передачи управления, а не для обращения к данным. Нарушение этого правила не выявляется редактором связей и может повлечь использование программой неверных данных;

9) редактор связей не использует оверлейный драйвер при разрешении глобальных имен, определенных в программной секции, для которой признак типа имеет значение D. Это значит, что программа пользователя должна сама загружать в память соответствующие оверлейные сегменты перед обращением к имеющимся в них данным;

10) для передачи управления в оверлейный сегмент нельзя использовать имя .CSECT. Эта команда не приводит к загрузке нужного сегмента в память. Для передачи управления от одного сегмента другому следует использовать только глобальный символ;

11) оверлейные области должны задаваться в возрастающем порядке и являются только считываемыми. Когда один оверлейный сегмент в области заменяет другой, текущее состояние старого сегмента не запоминается, поэтому все таблицы, переменные и команды, которые должны оставаться неизменными, следует размещать в корневом сегменте;

12) нельзя использовать в программе канал 17 (восьмеричное), поскольку по этому каналу считываются оверлеи;

13) редактор связей не создает оверлейную таблицу входов для разрешения глобальных имен, определенных в программных секциях программ, написанных на языках ФОРТРАН и АССЕМБЛЕР, т. к. редактор связей пересылает программные секции из оверлейного сегмента в корневой.

3.4.2. Оверлеи в расширенной памяти.

Программа редактор связей может использоваться для создания программ оверлейной структуры, использующих расширенную память. Хотя для выполнения таких программ необходима конфигурация, включающая диспетчер памяти, редактирование связей можно выполнить в системе любой конфигурации. Перед обращением к настоящему разделу необходимо ознакомиться с материалами, изложенными в п. 3.4.1 (оверлеи в нижней памяти): многое из этого материала применимо также и к оверлеям в расширенной памяти.

Преобразовать программу оверлейной структуры для работы с расширенной памятью обычно можно не прибегая к модификации программы. Оверлейный драйвер расширенной памяти и клавиатурный монитор содержат все программные запросы, необходимые для осуществления доступа к расширенной памяти (подробно об ограничениях, накладываемых расширенной памятью, см. [3]). Дополнительные данные для этих запросов содержатся также в оверлейных таблицах, так что доступ к расширенной памяти программа пользователя может осуществлять автоматически, не используя в программе запросы расширенной памяти.

Для создания программ с оверлеями в расширенной памяти следует использовать переключатель /V (см. п. 5.2.21).

Более подробно об использовании расширенной памяти см. [3].

3.4.2.1. Виртуальное адресное пространство.

Пользователь создает оверлейную структуру в расширенной памяти, считая, что ему доступны все 32К слов памяти (ячейки с 0 по 177777). В действительности это не так. Большой объем физической памяти (нижней памяти) занимают монитор (с ячейки 160000 вниз) и страница ввода — вывода

(страница в-в) (ячейки 160000—177777). Абсолютная секция программы пользователя обычно располагается в ячейках с 0 по 500. Однако, в результате управления памятью можно создать такую структуру программы, которая позволяет использовать 32К слов памяти. Это пространство называется виртуальным адресным пространством. Аппаратура диспетчера памяти и монитор позволяют разместить часть 32К слов адресного пространства в расширенной памяти.

Виртуальное адресное пространство разделяется на 8 секций, называемых страницами, с номерами от 0 до 7. Каждая страница содержит 4К слов памяти. Обращение к странице осуществляется с помощью регистра активной страницы (РАС). Этот регистр содержит константу перемещения, которая управляет отображением каждой страницы в расширенную память.

Виртуальное адресное пространство показано на рис. 1.

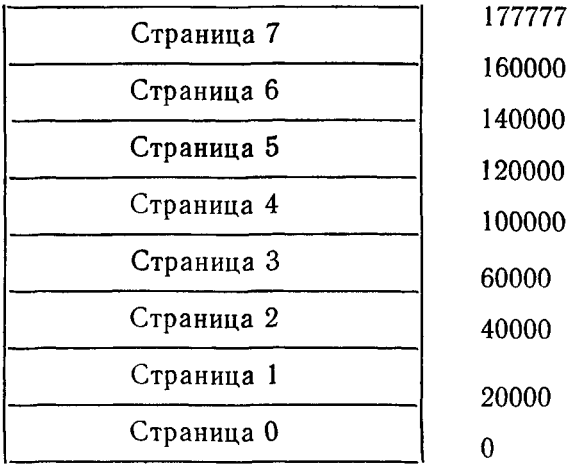


Рис. 1

Каждый оверлей, который находится в расширенной памяти, должен начинаться на границе страницы. Для этого редактор связей автоматически округляет размер каждого сегмента.

3.4.2.2. Физическое адресное пространство.

При создании загрузочного модуля с оверлеями в расширенной памяти редактор связей определяет, каким образом каждый оверлей будет отображен в расширенную память.

Программа редактор связей обрабатывает оверлеи в расширенной памяти не так, как оверлеи в нижней памяти (см. рис. 2 и рис. 3).

Физическое адресное пространство программы с оверлеями в нижней памяти показано на рис. 2.

ФИЗИЧЕСКОЕ АДРЕСНОЕ ПРОСТРАНСТВО ПРОГРАММЫ С ОВЕРЛЕЯМИ В НИЖНЕЙ ПАМЯТИ

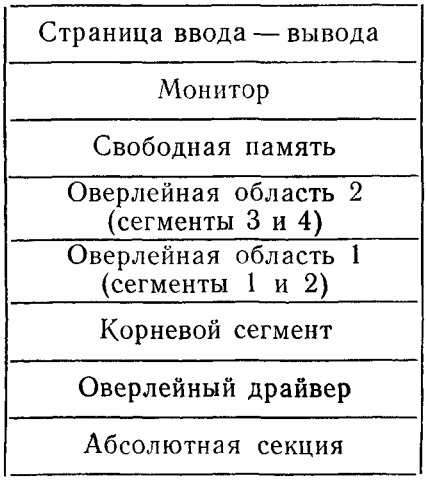


Рис. 2

Как показано на рис. 2, оверлейные сегменты совместно используют соответствующие оверлейные области. Каждый оверлей считывается в свою область со вспомогательного запоминающего устройства после вызова.

Виртуальное адресное пространство программы с оверлеями в расширенной памяти (левая диаграмма) и физическое адресное пространство этой программы (правая диаграмма) показаны на рис. 3.

Как показано на рис. 3, в виртуальном адресном пространстве расположена одна виртуальная оверлейная область, начинающаяся на границе страницы (ячейки с 20000 по 37777). Области адресного пространства, которые отображаются в расширенную память, называются виртуальными оверлейными областями. Оверлейные сегменты, отображенные в

расширенную память, занимают в расширенной памяти смежные области, называемые разделами. Таким образом, после вызова сегменты с 1 по 4 (рис. 3) находятся одновременно в расширенной памяти.

ВИРТУАЛЬНОЕ И ФИЗИЧЕСКОЕ АДРЕСНЫЕ ПРОСТРАНСТВА
С ОВЕРЛЕЯМИ В РАСШИРЕННОЙ ПАМЯТИ

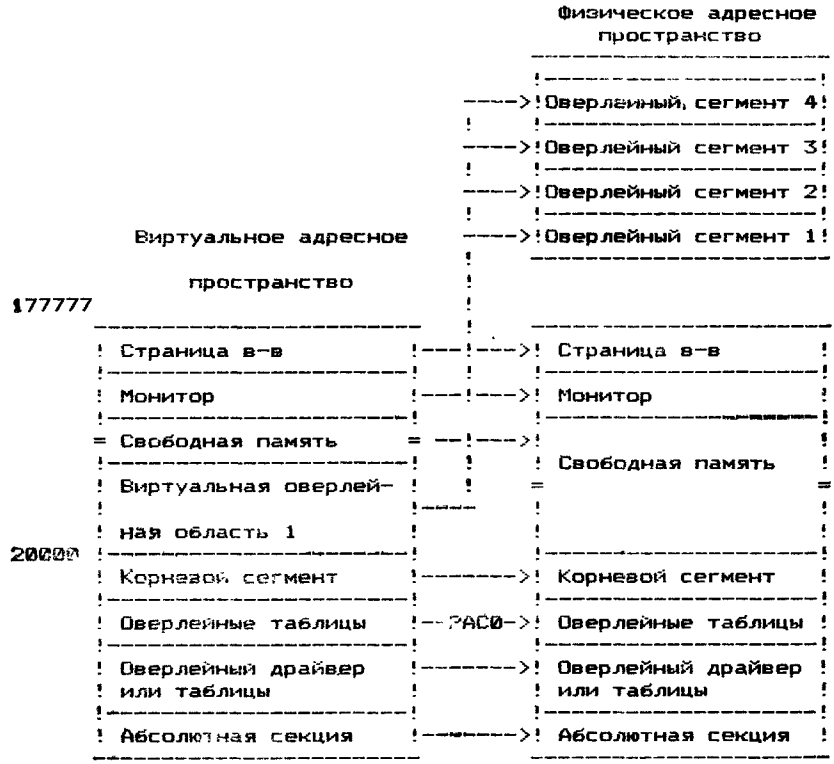


Рис.3

3.4.2.3. Виртуальные и привилегированные задания.

Объем виртуального адресного пространства, распределенного для программы, зависит от типа программы.

Фоновые, основные и системные задания разделяются на:

1) виртуальные;

2) привилегированные.

Виртуальные задания могут использовать все 32К слов виртуального адресного пространства, но они не имеют прямого доступа к странице в-в, монитору, векторам и другим заданиям.

Привилегированные задания также могут использовать 32К слов памяти, но защищенные области (монитор, страница в-в, векторы) являются частью этого адресного пространства.

Виртуальные и привилегированные задания могут отображаться в расширенную память. Поэтому можно создавать как привилегированные задания (основные, фоновые, системные), так и виртуальные задания (основные, фоновые, системные) с оверлеями в расширенной памяти.

Более подробно об использовании виртуальных и привилегированных заданий см. [3].

4. ВЫПОЛНЕНИЕ ПРОГРАММЫ

Для вызова редактора связей с системного устройства следует ввести с терминала команду R LINK <ВК> после того, как монитор выведет на терминал точку.

После вызова редактор связей выводит на терминал звездочку и ожидает ввода командной строки. Если в это время нажать клавишу <ВК>, то редактор связей выводит на терминал номер своей версии.

ПРИМЕЧАНИЕ. Операции, выполняемые редактором связей, могут быть выполнены монитором по команде LINK (см. [2]).

5. КОМАНДА ОПЕРАТОРА

5.1. Режим работы.

Режим работы редактора связей задается введением с терминала одной или нескольких командных строк. Формат первой командной строки:

[загрспф], [картспф], [опрспф]=обспф [, обспф...][/прк...]

формат последующих командных строк:

обспф [, обспф...][/прк...],

где загспф — спецификация файла загрузочного модуля (устройство, имя и тип файла);

картспф — спецификация файла карты загрузки;

опрспф — спецификация файла определений имен;

обспф — спецификация объектного или библиотечного файла;

/прк — переключатель (см. табл. 5).

Последующие командные строки используются:

1) при создании оверлейных программ;

2) если спецификации входных файлов не могут быть размещены в одной командной строке.

По умолчанию типом файла предполагается:

— .SAV для файла загрузочного модуля в формате отображения памяти;

— .REL для файла загрузочного модуля в перемещаемом формате отображения памяти;

— .LDA для файла загрузочного модуля в абсолютном двоичном формате;

— .MAP для файла карты загрузки;

— .STB для файла определений имен;

— .OBJ для входных файлов.

Указанные в командной строке устройства ввода — вывода должны быть устройствами с произвольным доступом. Исключение составляют устройства вывода для файла загрузочного модуля в формате LDA, файла карты загрузки и файла определений имен. Эти файлы могут быть выведены на любое устройство вывода системы.

Если в командной строке не указана спецификация какого-либо выходного файла, то редактор связей предполагает, что создание этого файла не требуется. Например, если в командной строке не указаны спецификации выходных файлов, то результатом работы редактора связей будут выдаваемые на терминал сообщения об обнаруженных ошибках (см. р. 7).

Для выхода из редактора связей и передачи управления монитору следует ввести команду СУ/С, если редактор связей ожидает ввода с терминала, или дважды СУ/С, если редактор связей выполняет редактирование.

Для повторного пуска редактора связей следует ввести с терминала команду REENTER.

После завершения редактирования редактор связей выводит на терминал звездочку и ожидает ввода командной строки.

5.2. Переключатели.

В табл. 5 приведены переключатели, используемые программой редактор связей.

Переключатели должны указываться в первой командной строке. Исключение составляют переключатели:

— /C, указываемый в любой строке, кроме последней;

— /O:N, указываемый в любой строке, кроме первой;

— //, указываемый в первой и последней строках.

Таблица 5

Переключатель	Назначение
1	2
/A	Расположить глобальные имена в карте загрузки в алфавитном порядке
/B:N	Использовать значение N в качестве младшего адреса редактируемой программы (не допускается использование совместно с /H и /R)
/C	Ввести еще одну командную строку (не допускается использование совместно с //)
/D	Включить библиотечный модуль одновременно в несколько оверлейных сегментов, которые обращаются к этому модулю
/E:N	Увеличить размер указываемой программной секции до N байтов
/F	Использовать для разрешения глобальных имен библиотеку SY:FORLIB.OBJ
/G	Изменить размер буфера библиотечного справочника
/H:N	Использовать значение N в качестве старшего адреса редактируемой программы (не допускается использование совместно с /B, /R, /Y и /Q)
/I	Включить в загрузочный модуль указываемые модули из библиотек, заданных в командной строке
/K:N	Занести значение N (N=2, 3 ... 28) по адресу 56 нулевого блока загрузочного модуля. Ограничивает отводимую по запросу .SETTOP память величиной NK. Не допускается использование совместно с /R
/L	Сформировать загрузочный модуль в формате LDA
/M [:N]	Использовать в качестве начального значения указателя стека значение указываемого глобального имени или значение N. Не следует использовать совместно с /R
/N	Включить в карту загрузки таблицу перекрестных ссылок глобальных имен
/O:N	Сформировать оверлей в нижней памяти; N — номер оверлейной области. Не допускается использование совместно с /L
/P	Изменить размер таблицы имен библиотечных программ
/Q	Указать абсолютные базовые адреса программных секций. Не допускается использование совместно с /H или /R
/R [:N]	Сформировать загрузочный модуль в формате REL: N — размер стека в байтах. Не допускается использование совместно с /B, /H, /K и /L
/S	Использовать максимально возможный объем памяти для таблицы имен (только в случае переполнения таблицы имен)
/T:N	Использовать в качестве точки входа редактируемой программы значение указываемого глобального имени или значение N

1	2
/U:N	Увеличить размер указываемой программной секции корневого сегмента так, чтобы размер корневого сегмента (в байтах) увеличился до величины, кратной N
/V	Позволяет использовать специальные функции запроса .SETTOP и директивы .LIMIT в режиме XM. Не допускается использование совместно с /L
/V:N[:M]	Сформировать оверлей в расширенной памяти; N — номер виртуальной оверлейной области; M — номер раздела в расширенной памяти
/W	Сформировать широкоформатную карту загрузки
/X	Не включать в загрузочный модуль карту использования памяти, если старший адрес редактируемой программы меньше 400 (восьмеричное)
/Y:N	Расположить указанную программную секцию в корневом сегменте, начиная с указанного адреса. Не допускается использование совместно с /H
/Z:N	Занести значение N в неиспользуемые программой адреса загрузочного модуля.
//	Вводить командные строки до следующего задания переключателя //. Не допускается использование совместно с /C

Если в командной строке не указан переключатель /L или /R, то редактор связей формирует загрузочный модуль в формате SAV.

5.2.1. Переключатель /A. Переключатель /A формирует карту загрузки, в которой глобальные имена по каждой программной секции записываются в карте загрузки в алфавитном порядке. Если переключатель /A не указан, глобальные имена записываются в порядке возрастания их значений.

5.2.2. Переключатель /B:N. Переключатель /B:N указывает значение младшего адреса редактируемой программы. N (младший адрес) должно быть четным, не более чем шестизначным восьмеричным числом без знака.

Если значение N не указано, редактор связей выдает сообщение: ?LINK—F—/B NO VALUE.

По умолчанию, вне зависимости от формата загрузочного модуля, редактор связей полагает значение младшего адреса равным 1000 (восьмеричное). Если размер абсолютной секции программы больше 1000, то значение младшего адреса полагается равным старшему адресу абсолютной секции плюс 2.

Не допускается задавать переключатель /B:N одновременно с переключателем /H или /R.

5.2.3. Переключатели /C и //. Переключатели /C и // ис-

пользуются, когда для задания режима работы требуется ввести несколько командных строк.

Переключатель /C указывает редактору связей на необходимость ввести еще одну командную строку. После ввода командной строки, в которой указан переключатель /C, редактор связей выводит на терминал звездочку и ожидает ввода следующей командной строки.

Пример:

```
*OUTPUT,LP:=INPUT/C
*
```

Переключатель // указывает редактору связей на необходимость вводить командные строки до тех пор, пока не будет задан еще один переключатель //.

Одновременное задание переключателей /C и // недопустимо.

Пример:

```
*LINK,LINK=LINK0/B:700//
*LINK1/O:1
*LINK2/O:1
*LINK3/O:1
*LINKM/O:1//
```

5.2.4. Переключатель /D. Переключатель /D позволяет поместить библиотечные модули одновременно в несколько оверлейных сегментов, которые обращаются к этим модулям.

Если указан переключатель /D, то после ввода последней командной строки, редактор связей выдает сообщение:

DUPLICATE SYMBOL?

В ответ необходимо ввести глобальное имя из библиотечного модуля, который следует поместить в тех сегментах, которые обращаются к этим именам, и нажать клавишу <ВК>. Если введено имя, редактор связей вновь выдает: DUPLICATE SYMBOL? и ожидает ответа. Нажатие клавиши <ВК> означает конец списка глобальных имен.

Если указано глобальное имя не из библиотечного модуля, глобальное имя не дублируется, и редактор связей выдает сообщение: ?LINK—W—DUPLICATE SYMBOL 'SYMBOL' DEFINED IN DEV:FILNAM.TYP.

Когда не указан переключатель /D, и к глобальному имени, определенному в библиотечном модуле, обращаются из другого сегмента, редактор связей помещает библиотечный модуль в корневой сегмент.

Если к глобальному имени, определенному в библиотечном модуле, обращаются из корневого сегмента, редактор связей помещает библиотечный модуль в корневой сегмент.

В этом случае глобальное имя не дублируется, и редактор связей выдает сообщение:

?LINK-W-DUPLICATE SYMBOL 'SYMBOL' IS FORCED TO THE ROOT

Если к дублируемому библиотечному модулю не обращаются из корневого сегмента, все глобальные секции из этого модуля, к которым обращаются из нескольких сегментов, всегда помещаются в корневой сегмент. Если локальные секции из этого библиотечного модуля нет необходимости связывать друг с другом, следует определить глобальную секцию с помощью признака CON. После этого редактор связей размещает отдельную копию глобальной секции в корневом сегменте для каждой копии локальной секции дублированного библиотечного модуля, размещенного в оверлеях. Несмотря на то, что глобальные секции резидентны в корневом сегменте, в то время, как локальные секции резидентны в сверлеях, каждая копия библиотечного модуля остается тождественной ему, как отдельная копия модуля.

Если дублированный библиотечный модуль содержит секцию глобальных данных, к которой будут обращаться локальные секции в пределах модуля, но глобальная секция не обращается к локальной секции в пределах модуля, копию глобальной секции следует переслать в корневой сегмент. Для этого следует определить секцию с помощью уникального имени и признаков GBL и OVR. После размещения глобальной секции в корневом сегменте локальная секция из дублированного библиотечного модуля, которая резидентна в оверлее, может обращаться к глобальной секции в корневом сегменте.

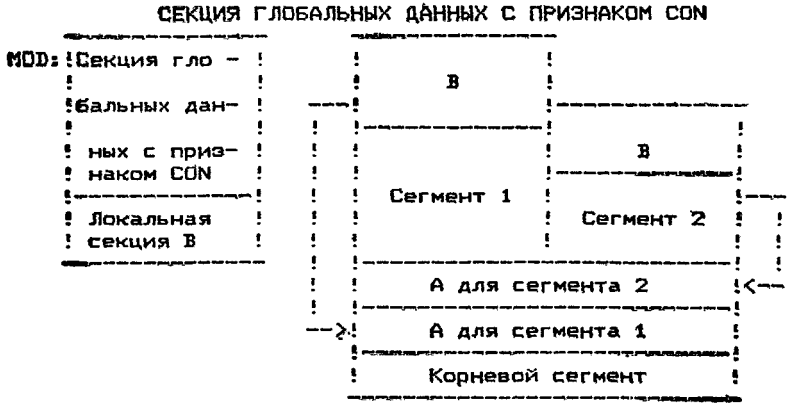


Рис. 4

На рис. 4 показан дублированный библиотечный модуль MOD, из которого секция глобальных данных A помещена в корневой сегмент с признаком CON. Каждой локальной секции B доступна информация из копии глобальной секции, размещенной в корневом сегменте. Однако, локальные секции не могут обмениваться данными, поскольку их обращения ограничены для разных ячеек.

На рис. 5 показан тот же дублированный библиотечный модуль MOD, из которого глобальная секция A помещена в корневой сегмент с признаком OVR. В этом случае локальные секции B могут обращаться к одним и тем же ячейкам в глобальной секции A для обмена информацией.

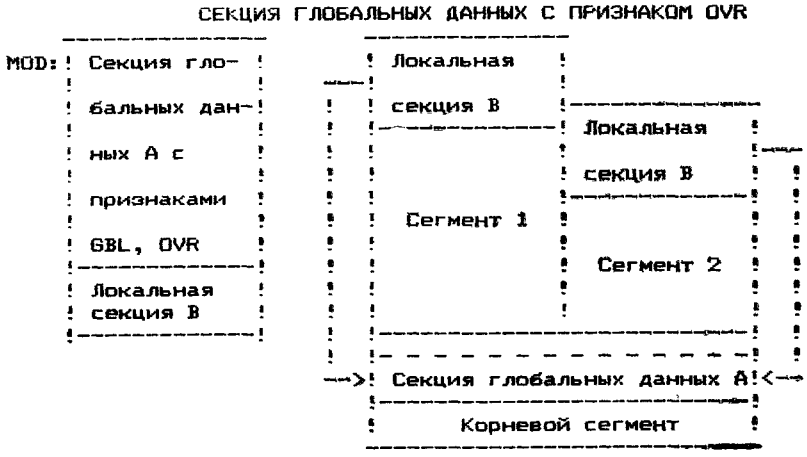


Рис.5

5.2.5. Переключатель /E:N. Переключатель /E:N позволяет увеличить размер одной из программных секций корневого сегмента редактируемой программы до величины N байтов. N должно быть четным, не более чем шестизначным восьмеричным числом без знака.

Если указан переключатель /E:N, то после ввода последней командной строки редактор связей выводит на терминал:
EXTEND SECTION?
В ответ необходимо ввести имя программной секции, размер которой требуется увеличить. Если значение N меньше фактического размера указанной программной секции, то переключатель /E:N игнорируется, редактирование продолжается.

Оператор может увеличить размер только одной программной секции.

Пример:

```
*X,TT:=LK001/E:100
EXTEND SECTION?CODE
```

В приведенном выше примере размер программной секции CODE увеличен до 100 байтов.

5.2.6. Переключатель /F. Переключатель /F включает в редактируемую программу модули из библиотеки FORLIB.OBJ для разрешения глобальных имен. Файл FORLIB.OBJ должен находиться на системном устройстве SY:.

Пример:

```
*FILE,LP:=AB/F
```

По этой команде происходит редактирование объектного файла AB.OBJ и модуля из библиотеки фортрана FORLIB.OBJ, в результате которого формируется загрузочный модуль FILE.SAV.

Переключатель /F используется только для совместимости с другими версиями ФОДОС.

5.2.7. Переключатель /G. Переключатель /G позволяет изменить размер внутреннего буфера таблицы точек входа редактора связей для модулей из библиотеки кратных определений.

Если размер внутреннего буфера таблицы точек входа слишком мал, редактор связей выдает сообщение:

```
?LINK—F—LIBRARY EPT TOO BIG,
INCREASE BUFFER WITH /G
```

Переключатель /G следует использовать только в случае переполнения таблицы имен, поскольку он замедляет процесс редактирования.

5.2.8. Переключатель /H:N. Переключатель /H:N задает старший адрес редактируемой программы. N (старший адрес) должно быть четным, не более чем шестизначным восьмеричным числом без знака.

Если значение N не указано, редактор связей выдает сообщение: ?LINK—F—/H NO VALUE.

Если указано нечетное значение N, редактор связей выдает сообщение: ?LINK—F—/H ODD VALUE.

Если задан переключатель /H:N, то младший адрес редактируемой программы определяется в соответствии с размером программы.

Если значение N меньше допустимого, то редактор связей выдает сообщение: ?LINK—F—/H VALUE TOO LOW.

Недопустимо задание переключателя /H:N одновременно с переключателем /R, /Y или /B.

5.2.9. Переключатель /I. Переключатель /I позволяет включить в редактируемую программу модули из указанных в командных строках библиотек, т. е. включить в программу библиотечные модули, необходимые для разрешения глобальных имен. Эти модули размещаются в корневом сегменте.

Если задан переключатель /I, то после ввода последней командной строки редактор связей выводит на терминал:

LIBRARY SEARCH?

В ответ необходимо ввести глобальное имя из библиотечного модуля, включаемого в программу, или нажать клавишу <ВК>. Если введено имя, редактор связей выводит на терминал:

LIBRARY SEARCH?

и ожидает ответа. Нажатие клавиши <ВК> означает конец списка глобальных имен.

Пример:

```
*SCCA=SCCA/I
LIBRARY SEARCH? ¤SHORT
LIBRARY SEARCH?
```

В данном примере глобальное имя ¤SHORT включается в загрузочный модуль SCCA.

5.2.10. Переключатель /K:N. Переключатель /K:N позволяет записать значение N (N — число блоков памяти размером 1K, требуемых программе) по адресу 56 нулевого блока загрузочного модуля. N должно быть натуральным числом от 2 до 28.

Переключатель /K позволяет ограничить количество памяти, распределенной с помощью запроса .SETTOP (см. [3]), до NK слов.

Не допускается задавать переключатель /K:N одновременно с переключателем /R.

5.2.11. Переключатель /L. Переключатель /L формирует загрузочный модуль в формате LDA. Такой загрузочный модуль предназначен для работы в перфоленточной операционной системе. Недопустимо задание переключателя /L одновременно с переключателями /O, /R или /V. В следующем примере показано создание загрузочного модуля в абсолютном формате:

```
*OUT,LP:=IN1,IN2/L
```

5.2.12. Переключатель /M[:N]. Переключатель /M[:N] устанавливает начальное значение указателя стека по адресу 42. Если /M:N используется совместно с /R:N, то редактор

связей игнорирует значение, указанное в /R:N. N (начальное значение указателя стека) должно быть четным, не более чем шестизначным восьмеричным числом без знака. Если N не указано, то после ввода последней командной строки, редактор связей выводит на терминал: STACK SYMBOL?
В ответ необходимо ввести глобальное имя (число указывать не следует), значение которого будет начальным значением указателя стека. Это глобальное имя должно быть определено в корневом сегменте редактируемой программы.

Если указано несуществующее имя, на терминал выводится сообщение об ошибке, и стековый адрес устанавливается равным 1000 (для файлов типа .SAV) или значению младшего адреса редактируемой программы, если использован переключатель /B. Если размер абсолютной программной секции больше 1000 байтов, то область стека по умолчанию начинается после наибольшего адреса ASECT. Следует отметить, однако, что прямое присвоение (с помощью ASECT) адреса стека в программе предпочтительнее присвоения с помощью переключателя /M. Для этого используются операторы в программе MACRO:

```
.ASECT
.=42
.WORD INITSP    ;задается начальное значение указателя
                ;стека
.PSECT          ;возврат в предыдущую программную сек-
                ;цию
```

В следующем примере адрес стека указывается с помощью переключателя /M:

```
*OUTPUT=INPUT/M
STACK SYMBOL? BEG
```

Не допускается задавать переключатель /M:N одновременно с переключателем /R.

5.2.13. Переключатель /N. Переключатель /N включает в карту загрузки таблицу перекрестных ссылок глобальных имен (глобальные имена перечислены в алфавитном порядке). За каждым глобальным именем стоит имя модуля (имена модулей также перечислены в алфавитном порядке). Знак '#', следующий за именем модуля, указывает, что глобальное имя определено в этом модуле, знак '+' указывает, что модуль из библиотеки.

5.2.14. Переключатель /O:N. Переключатель /O:N используется для создания загрузочных модулей оверлейной структуры в нижней памяти (см. п. 3.4.1). N (номер оверлейной области) должно быть не более, чем шестизначным восьме-

ричным числом без знака. Оверлейные области необходимо указывать в командных строках в порядке возрастания их номеров, причем в одной командной строке может быть задан только один переключатель /O:N.

Все модули, указанные до следующего переключателя /O, будут находиться в памяти одновременно. Если в очередном переключателе /O:N указать уже использованный номер оверлейной области, то указанные в этом переключателе модули займут те же ячейки памяти, но уже в другое время. В следующем примере модули R и S занимают ту же область памяти, что и T, но в разное время:

```
*MAIN,LP:=ROOT/C
*R,S/O:1/C
*T/O:1
```

В следующем примере установлены две оверлейные области:

```
*OUTPUT,LP:=INPUT//
*OBJA/O:1
*OBJB/O:1
*OBJC/O:2
*OBJD/O:2//
```

В приведенном ниже примере оверлейные области указаны в порядке возрастания их номеров:

```
*A=A/C
*B/O:1/C
*C/O:1/C
*D/O:1/C
*G/O:2
```

В следующем примере оверлейные области указаны не в возрастающем порядке, поэтому на терминал выдается сообщение об ошибке:

```
*X=LIBR0//
*LIBR1/O:1
*LIBR2/O:0
?LINK—W—/O OR /V OPTION ERROR, RE—ENTER LINE
*
```

5.2.15. Переключатель /P:N позволяет изменить размер таблицы имен библиотечных программ, являющейся составной частью общей таблицы имен. N представляет собой количество библиотечных программ (имен), на которое должна быть рассчитана таблица. По умолчанию для таблицы имен отводится область памяти на 170. имен, что эквивалентно заданию переключателя /P:170. или /P:252 (восьмеричное).

При переполнении таблицы имен необходимо указать значение N меньше 170. Это уменьшит размер области, используемой таблицей имен библиотечных программ, и увеличит размер области, используемой общей таблицей имен. Если указанное значение N слишком мало, редактор связей выдает сообщение:

?LINK-F-LIBRARY LIST OVERFLOW, INCREASE SIZE WITN /P

Это означает, что необходимо увеличить размер области, используемой таблицей имен библиотечных программ.

5.2.16. Переключатель /Q. Переключатель /Q позволяет указывать абсолютные базовые адреса программных секций. Таких программных секций может быть не более 8. Этот переключатель особенно удобен, если программные секции в абсолютном формате предполагается разместить в ПЗУ.

Если указан переключатель /Q, то после ввода последней командной строки редактор связей выводит на терминал:

LOAD SECTION: ADDRESS?

В ответ необходимо ввести имя и адрес загрузки программной секции и нажать клавишу <ВК>. Если введено имя и адрес загрузки программной секции, редактор связей выводит на терминал: LOAD SECTION: ADDRESS?

Нажатие клавиши <ВК> означает конец списка программных секций.

При использовании переключателя /Q должны соблюдаться следующие правила:

1) имя программной секции не должно превышать 6-ти символов;

2) адрес загрузки программной секции должен быть четным восьмеричным числом. Если адрес загрузки не указан, редактор связей выдает сообщение:

?LINK—W—NO LOAD ADDRESS

Если указан нечетный адрес, редактор связей выдает сообщение: ?LINK—W—LOAD ADDRESS ODD.

3) недопустимо задание переключателя /Q с переключателем /H или /R.

Пример:

```
*FILE,TT:=FILE,FILE 1/Q/L
LOAD SECTION: ADDRESS? PSECT1:1000
LOAD SECTION: ADDRESS? PSECT2:2500
LOAD SECTION: ADDRESS? PSECT3:4000
LOAD SECTION: ADDRESS? <BK>
*
```

5.2.17. Переключатель /R[:N]. Переключатель /R[:N] формирует загрузочный модуль в формате .REL для исполь-

зования в качестве основного задания в режимах FB и XM. Загрузочный модуль типа .REL не используется с монитором одного задания. N (размер стека в байтах) должно быть четным восьмеричным числом без знака. По умолчанию — 128.

Пример:

*FILEO,LP:=FILEI,NEXT/R:200

По этой команде происходит редактирование объектных файлов FILEI.OBJ и NEXT.OBJ, в результате которого вырабатывается загрузочный модуль FILEO.REL.

Недопустимо задание переключателя /R[:N] одновременно с переключателем /B, /H, /L или /K.

5.2.18. Переключатель /S. Переключатель /S указывает редактору связей на необходимость предоставить максимально возможную область памяти для таблицы имен (за счет буферов ввода — вывода). Использовать переключатель /S следует только в случае переполнения таблицы имен, поскольку он замедляет процесс редактирования. Если задан переключатель /S, не следует указывать в командной строке файл определений имен или карты загрузки.

5.2.19. Переключатель /T[:N]. Переключатель /T:N указывает точку входа (адрес, с которого начинается выполнение редактируемой программы. N (точка входа) должно быть четным, не более чем шестизначным восьмеричным числом без знака. Если N не указано, то после ввода последней командной строки редактор связей выдает сообщение:

TRANSFER SYMBOL?

В ответ необходимо ввести глобальное имя, значение которого будет точкой входа программы. Число вводить нельзя.

Если указано несуществующее имя, на терминал выводится сообщение об ошибке и адрес точки входа устанавливается равным 1. Это вызывает прерывание программы.

Если указан нечетный адрес, программа не запускается после загрузки, и управление передается монитору.

Загрузка программы осуществляется с помощью команды R, RUN или FRUN. Прямое присвоение (.ASECT) адреса перехода в программе имеет приоритет над присвоением с помощью /T, которое, в свою очередь, имеет приоритет над присвоением в директиве .END. Для указания точки входа в программу на языке АССЕМБЛЕРа используется одна из следующих конструкций:

```
.ASECT
.=40
.WORD START1  ; точка входа в редактируемую
              ; программу
```

```
        .PSECT          ;возврат в предыдущую програм-
                        ;мную секцию
START1: .
        .
        .
или
START2: .               ; адрес перезапуска
        .
        .
        .END START2
```

В следующем примере объединяются файлы LIBRO.OBJ и ODT.OBJ. Выполнение начинается с адреса ODT (O.ODT — адрес точки входа в программу):

```
*LBRODT,LBRODT=LIBRO,ODT/T/W//
*LIBR1/O:1
*LIBR2/O:1
*LIBR3/O:1
*LIBR4/O:1
*LIBR5/O:1
*LIBR6/O:1
*LBREM/O:1//
TRANSFER SYMBOL? O.ODT
*
```

5.2.20. Переключатель /U:N. Переключатель /U:N позволяет увеличить размер одной из программных секций корневого сегмента так, что размер корневого сегмента (в байтах) увеличивается до величины, кратной N. N должно быть восьмеричным числом степени 2.

Если указан переключатель /U:N, то после ввода последней командной строки редактор связей выводит на терминал:

ROUND SECTION?

В ответ необходимо ввести имя программной секции, размер которой требуется увеличить.

Пример:

```
*LK007,TT:=LK007/U:200
ROUND SECTION? CHAR
```

В данном примере увеличивается секция CHAR.

Если указанная программная секция не найдена, редактор связей выдает сообщение: ?LINK—W—ROUND SECTION NOT FOUND.

В этом случае переключатель /U:N игнорируется, редактирование продолжается.

5.2.21. Переключатель /V. Переключатель /V:N[:M] используется для создания загрузочных модулей оверлейной

структуры в расширенной памяти (см. п. 3.4.2). N — номер виртуальной оверлейной области (область виртуального адресного пространства), M — номер раздела (область физического адресного пространства). Виртуальные оверлейные области необходимо указывать в командных строках в порядке возрастания их номеров, причем в одной командной строке может быть указан только один переключатель /V:N[:M].

ПРИМЕЧАНИЕ. Если переключатель /V указан в первой командной строке без аргументов, виртуальные или привилегированные задания (основные, фоновые) будут отображены в рабочую область расширенной памяти с помощью программного запроса .SETTOP. Более подробно об использовании программного запроса .SETTOP см. [3].

Пользователь может комбинировать в одной программе оверлейные сегменты в нижней памяти и оверлейные сегменты в расширенной памяти. Оверлейные области и виртуальные оверлейные области указываются в командных строках в возрастающем порядке, причем оверлейные сегменты в нижней памяти указываются перед оверлейными сегментами в расширенной памяти.

Рассмотрим на следующем примере использование переключателей /V:N[:M] и /O:N:

```
.R LINK
*PROG=PROG//
*SEG1/O:1
*SEG2/O:1
*SEG3/V:2
*SEG4/V:2
*SEG5/V:2:1
*SEG6/V:2:1
*SEG7/V:2:1
*SEG8/V:2:2
*SEG9/V:2:2
*SEG10/V:3//
```

В этом примере сегменты SEG1 и SEG2 делят одни и те же ячейки в нижней памяти. Сегменты SEG3 и SEG4, соответствующие виртуальной оверлейной области 2, используют собственные разделы в расширенной памяти, сегменты SEG5, SEG6 и SEG7 делят одни и те же ячейки в расширенной памяти, SEG8 и SEG9 также используют один раздел. SEG10, соответствующий виртуальной оверлейной области 3, использует в расширенной памяти собственный раздел. Распределение памяти для этого примера показано на рис. 6.

ВИРТУАЛЬНОЕ И ФИЗИЧЕСКОЕ АДРЕСНЫЕ ПРОСТРАНСТВА ПРОГРАММЫ С ОВЕРЛЕЯМИ В НИЖНЕЙ И РАСШИРЕННОЙ ПАМЯТИ

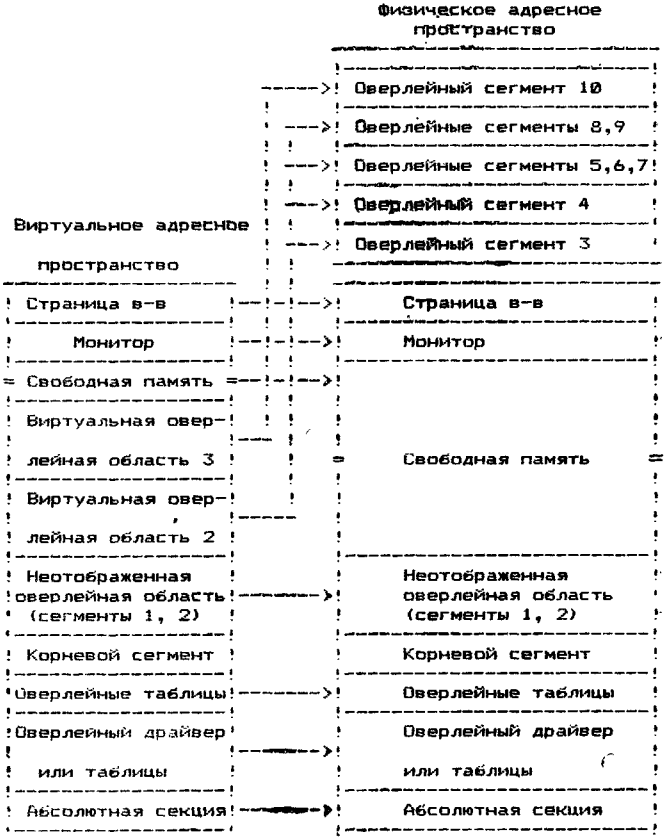


Рис.6

5.2.22. Переключатель /W. Переключатель /W формирует карту загрузки, в которой глобальные имена и их значения записываются в шесть колонок (с шириной строки 132 символа) вместо трех.

5.2.23. Переключатель /X. Переключатель /X запрещает редактору связей включать в загрузочный модуль карту использования памяти, если старший адрес редактируемой программы меньше 400 (восьмеричное). Карта использования памяти находится в ячейках с 360 по 377 нулевого блока за-

грузочного модуля. Каждый разряд представляет один 256-словный блок памяти. Разряд 7 байта 360 соответствует ячейкам с 0 по 777; разряд 6 — ячейкам с 1000 по 1777 и т. д. Эта информация используется командами R, RUN или FRUN при загрузке программы.

5.2.24. Переключатель /Y:N. Переключатель /Y:N позволяет изменить значение адреса загрузки одной из программных секций корневого сегмента редактируемой программы. Значение этого адреса увеличивается до величины, кратной N. N должно быть восьмеричным числом, представляющим целую степень числа 2. Если указан переключатель /Y:N, то после ввода последней командной строки редактор связей выводит на терминал:

BOUNDARY SECTION?

В ответ необходимо ввести имя программной секции, адрес загрузки которой требуется изменить.

Если указанная программная секция не найдена, редактор связей выдает сообщение: ?LINK—W—BOUNDARY SECTION NOT FOUND

Переключатель /Y:N игнорируется, редактирование продолжается.

Недопустимо использование переключателя /Y:N с переключателем /H.

Пример:

```
*FMONSJ.SYS=BISJ,RMSJ,KMSJ,TBSJ/Y:200
BOUNDARY SECTION? OVLYO
```

В данном примере изменяется значение адреса загрузки программной секции OVLYO.

5.2.25. Переключатель /Z:N. Переключатель /Z:N указывает редактору связей на необходимость записать значение N в неиспользуемые программой адреса загрузочного модуля. По умолчанию предполагается переключатель /Z:0.

Этот переключатель используется для устранения случайных результатов, когда программа обращается к неинициализированной памяти.

6. ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

6.1. Объектные модули.

Входными данными для редактора связей являются объектные модули, создаваемые трансляторами ФОДОС-2. Структура объектного модуля описана в [4].

6.2. Загрузочный модуль. Загрузочный модуль может быть сформирован редактором связей в одном из трех форматов:

1) в формате отображения памяти (SAV);
2) в перемещаемом формате отображения памяти (REL);
3) в абсолютном двоичном формате (LDA).

Структура загрузочных модулей в формате SAV и REL показана на рис. 7.

СТРУКТУРА ЗАГРУЗОЧНОГО МОДУЛЯ В ФОРМАТЕ SAV

Корневой сегмент	Оверлейные сегменты (необязательно)

Рис. 7

На рис. 8 приведена структура загрузочного модуля в формате REL:

Корневой сегмент	Оверлейные сегменты (необязательно)	Информация о перемещениях

Рис. 8

Загрузочный модуль в формате LDA используется для совместимости с перфоленточной операционной системой.

Загрузочный модуль в формате SAV содержит программу в том виде, в котором она будет находиться в памяти (нулевой блок соответствует адресам памяти 0—776, первый блок — адресам 1000—1776 и т. д.). В случае оверлейной программы оверлеи загружаются в память с адресов, записанных редактором связей в таблице оверлейного драйвера.

Загрузочный модуль в формате REL содержит программу в перемещаемом виде (нулевой блок соответствует адресам памяти 0—776, остальные блоки загружаются в память с адреса, определяемого монитором). Перемещаемость программы обеспечивается содержащейся в загрузочном модуле информацией о перемещении. Информация о перемещении состоит из списка адресов, содержимое которых требуется изменить при загрузке. Эти адреса рассматриваются относительно начала перемещаемой части программы пользователя (адрес 1000 в загрузочном модуле). Информация о перемещении записывается в следующем виде (рис. 9):

В разрядах 0—14 записывается деленное на 2 значение адреса ячейки слова, которое требуется изменить. Содержимое разряда 15 определяет тип изменения. Изменение выполняется следующим образом: константа, равная адресу загрузки программы, складывается или вычитается из слова по

указанному адресу в зависимости от содержимого 15-го разряда (0 — сложение, 1 — вычитание). Код 177776 является признаком окончания информации о перемещении. Для оверлейных программ информация о перемещении каждого сегмента заканчивается кодом 177777.

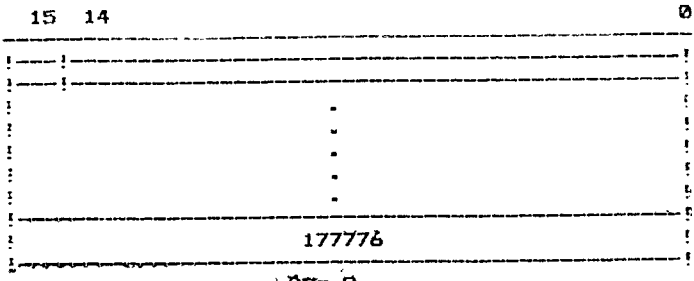


Рис. 9

В нулевом блоке загрузочного модуля редактор связей размещает информацию о программе, используемую при загрузке (табл. 6).

Таблица 6

Адрес	Назначение
1	2
40	Стартовый адрес программы
42	Начальное значение указателя стека
44	Слово состояния программы
46	Адрес загрузки программы обслуживания пользователя
50	Старший адрес программы
360—376	Карта использования памяти

Редактор связей записывает по адресам 40, 42 и 50 значения, вычисляемые во время редактирования, или значения, указанные в командной строке. По адресу 44 редактор связей записывает 1000, если программа оверлейная. По адресу 46 редактор связей записывает 0. Это указывает на то, что адрес загрузки USR определяется монитором.

Карта использования памяти (адреса 360—376) показывает, какие блоки загрузочного модуля должны загружаться в оперативную память при вызове программы.

Для загрузочных модулей в формате REL редактор связей определяет дополнительную информацию (табл. 7).

Таблица 7

Адрес	Назначение
1	2
14, 16	Вектор прерывания по команде BPT
20, 22	Вектор прерывания по команде IOT
34	Вектор прерывания по команде TRAP
52	Размер корневого сегмента в байтах
54	Размер стека в байтах (по умолчанию 128)
56	Размер оверлейной области в байтах
60	Признак формата (REL в коде RADIX-50)
62	Номер блока файла загрузочного модуля с которого начинается информация о перемещениях

Для программ с оверлеями в расширенной памяти редактор связей определяет дополнительную информацию (табл. 8).

Таблица 8

Адрес	Назначение
1	2
0	Идентификация программы (см. [3]), которая была создана с помощью переключателя /V:N[:M]
2	Старший адрес виртуальной памяти, используемый программой
64	Стартовый адрес оверлейной таблицы (используемый также программами с оверлеями в нижней памяти)
66	Начало информации о виртуальном оверлейном сегменте в оверлейной таблице драйверов

Любое слово нулевого блока загрузочного модуля может быть установлено в абсолютной секции программы. Это дает возможность пользователю самому устанавливать слова с адресами, приведенными в табл. 6, табл. 7 и табл. 8.

6.3. Карта загрузки. Карта загрузки показывает распределение памяти для загрузочного модуля.

В первой строке карты загрузки записывается номер версии редактора связей, дата и время выполнения редактирования. Во второй строке — имя файла загрузочного модуля, наименование и версия первого объектного модуля.

Затем идет информация о распределении памяти для программных секций. Для каждой программной секции эта информация включает в себя: имя, адрес загрузки, размер в байтах, значения признаков, список глобальных имен, определенных в секции, и их значения. В последней строке карты загрузки записывается точка входа программы и размер загрузочного модуля в байтах (в восьмеричной системе) и словах (в десятичной системе). Для оверлейных программ информация о распределении памяти для сегментов записывается в карте загрузки в порядке возрастания номеров сегментов независимо от того, где расположены оверлейные сегменты, в нижней или расширенной памяти.

Если в командной строке указан переключатель /N (см. п. 5.2.13), в карту загрузки будет включена таблица перекрестных ссылок глобальных имен (пример 2).

Адреса в карте загрузки, полученной для загрузочного модуля в формате REL, не являются абсолютными. Для вычисления абсолютного адреса необходимо адрес, указанный в карте загрузки, сложить с адресом загрузки и вычесть 1000 (1000 — это базовый адрес программы, предполагаемый редактором связей).

Пример 1.

Ниже показана карта загрузки фоновой программы:

```
1  PUDOS  LINK 803.00 LOAD MAP FRIDAY 11-JAN-85 11:25 PAGE 1
2  TEST       .SAV     TITLE: TEST    IDENT:
3
4  SECTION ADDR  SIZE GLOBAL VALUE GLOBAL VALUE GLOBAL VALUE
5
6  .ABS.   000000 001000 = 256. WORDS  (RW, I, GBL, ABS, OVR)
7          001000 000200 = 64.  WORDS  (RW, I, LCL, REL, CON)
8  TEST    001200 000174 = 62.  WORDS  (RW, I, LCL, REL, CON)
9                              START 001200  EXIT 001240
10
11 TRANSFER ADDRESS = 001200, HIGH LIMIT = 001372=381. WORDS
```

Подробное описание этой карты загрузки приведено в табл. 9.

Таблица 9

Номер строки	Содержание
1	2
1	Номер версии редактора связей, дата и время выполнения редактирования

1	2
2	Имя файла загрузочного модуля — TEST.SAV, наименование TEST (по умолчанию .MAIN) и номер версии (по умолчанию не указывается) первого объектного модуля
4	Информация о программной секции Заголовок описания программной секции 'SECTION' обозначает имя; 'ADDR' — стартовый адрес, 'SIZE' — длину и признаки (см табл. 1); 'GLOBAL' и 'VALUE' обозначают глобальные имена и соответствующие им восьмеричные значения
6	Абсолютная программная секция — .ABS. Эта строка включает стартовый адрес, длину и признаки (см. табл. 1) абсолютной программной секции
7	Неименованная программная секция. Эта секция следует в карте загрузки за абсолютной программной секцией. Для оверлейных программ неименованная программная секция следует за оверлейной таблицей (см. пример 2)
8—9	Программная секция TEST. В девятой строке указаны глобальные имена START и EXIT и их значения
11	Точка входа, старший адрес и количество слов редактируемой программы

Пример 2.

Ниже показана карта загрузки для загрузочного модуля PROG.SAV, оверлейная структура которого определена следующим образом:

```
*PROG,PROG=MOD//
*MOD1/O:1
*MOD2/O:1
*MOD3/V:2
*MOD4/V:3//
```

```
1 FODOS LINK 003.00 LOAD MAP THURSDAY 03-JAN-85 14:15 PAGE 1
2 PROG    .SAV    TITLE: .MAIN.  IDENT:
3
4 SECTION ADDR  SIZE  GLOBAL VALUE GLOBAL VALUE GLOBAL VALUE
5
6  .ABS.  000000 001000 = 256. WORDS  (RW, I, GBL, ABS, OVR)
7  $OHAND 001000 000252 = 85.  WORDS  (RW, I, GBL, REL, CON)
8               $OVRHV 001000  $OVRH   001004 VRREAD  001034
9               VRDONE 001046  $VDF5   001234 $VDF4   001236
10                 $VDF1  001246  $VDF2   001250
11  $OTABL 001252 000114 = 38. WORDS  (RW, D, GBL, REL, OVR)
12       001366 000410 = 132.  WORDS  (RW, I, LCL, REL, CON)
13  MAIN   001776 000070 = 28. WORDS  (RW, I, LCL, REL, CON)
14                   START 001776 RET1  002010 RET2  002014
15                      LIMIT 002024
16  LML4   002066 000026 = 11. WORDS  (RW, I, GBL, REL, CON)
17                     MSGL  002066
18  LML5   002114 000026 = 11. WORDS  (RW, I, GBL, REL, CON)
19                     MSGL2  002114
20  SEGMENT SIZE = 002142 = 561. WORDS
21
```

```
 22 OVERLAY REGION  000001  SEGMENT 000001
 23  LML2   002144  000032 = 13. WORDS (RW, I, LCL, REL, CON)
 24                      START1@ 002144
 25  SEGMENT SIZE = 000032 = 13.  WORDS
 26
 27 OVERLAY REGION  000001  SEGMENT 000002
 28  LML3   002144  000036 =15.  WORDS (RW, I, LCL, REL, CON)
 29                      START2@ 002144
 30  SEGMENT SIZE = 000036 = 15.  WORDS
 31
 32  ------------------------------------------------------------
 33
 34  VIRTUAL OVERLAY REGION  000002
 35  ---------------------------
 36
 37 PARTITION        000001 SEGMENT 000003
 38  LML7  020002   000034 = 14. WORDS (RW, I, LCL, REL, CON)
 39                      START3 020002
 40  LML6  020036   000042 = 17. WORDS (RW, I, GBL, REL, CON)
 41                      MSGL3  020036   RET4   020050
 42  SEGMENT SIZE = 000076 = 31.  WORDS
 43
 44  VIRTUAL OVERLAY REGION  000003
 45  ------------------------
 46
 47 PARTITION        000002 SEGMENT 000004
 48  LML9  040002   000076 = 31. WORDS (RW, I, GBL, REL, CON)
 49                      MSGL9@ 040002
 50 SEGMENT SIZE = 000076 = 31.  WORDS
 51
 52
 53  TRANSFER ADDRES = 001776, HIGH LIMIT =002200 =576. WORDS
 54
 55

 56 VIRTUALHIGHLIMIT=040076=8223.WORDS,NEXTFREEADDRESS=060000
 57
 58
 59 EXTENDED MEMORY REQUIRED = 000200 = 64. WORDS
 60 FODOS LINK 803.00GLOBALSYMBOLCROSS REFERENCE TABLE PAGE 1
 61
 62
 63  $OVDF1 VHANDL+
 64  $OVDF2 VHANDL+
 65  $OVDF3 VHANDL+
 66  $OVDF4 VHANDL+
 67  $OVDF5 VHANDL+
 68  $OVRH  VHANDL#+
 69  $OVRHV VHANDL#+
 70  $VDF1  VHANDL#+
 71  $VDF2  VHANDL#+
 72  $VDF4  VHANDL#+
 73  $VDF5  VHANDL#+
 74  LIMIT  .MAIN.#
 75  MSGL   .MAIN.#
 76  MSGL2  .MAIN.#
 77  MSGL3  .MAIN.#
 78  MSGL9  .MAIN.     .MAIN.#
 79  RET1   .MAIN.#
 80  RET2   .MAIN.#
 81  RET4   .MAIN.#
 82  START  .MAIN.#
 83  START1 .MAIN.     .MAIN.#
 84  START2 .MAIN.     .MAIN.#
 85  START3 .MAIN.#
 86  $$DONE VHAND#+
154 87  $$READ VHAND#+
```

Подробное описание этой карты загрузки приведено в табл. 10.

Таблица 10

Номер строки	Содержание
1	2
1—6	См. пример 1
7—10	Программная секция ¤OHAND содержит оверлейный драйвер для программ с оверлеями в нижней и расширенной памяти
11	Программная секция ¤OTABL содержит таблицы данных, используемые оверлейным драйвером
12	Неименованная программная секция. В карте загрузки оверлейных программ неименованная программная секция располагается после оверлейных таблиц
20	Указывает размер корневого сегмента
22	Информация о сегменте 1 оверлейной области 1
23—24	Программная секция LML2. В 24-й строке символ '@' указывает, что доступ к глобальному имени START1 осуществляется через оверлейную таблицу ¤OTABL, находящуюся в корневом сегменте
25	Указывает размер оверлейного сегмента отделяет часть
32	карты загрузки, предназначенную нижней памяти, от части, предназначенной расширенной памяти
34	Информация о виртуальной оверлейной области 2
37	Информация о разделе 1 сегмента 3
41	Программная секция LML6. Отсутствие символа '@' означает, что к программной секции LML6 обращаются только в пределах сегмента 3
42	Указывает размер оверлейного сегмента 3
44	Информация о виртуальной оверлейной области 3
47	Информация о разделе 2 сегмента 4
50	Указывает размер сегмента 4. Заметим, что сегменты 3 и 4 имеют одинаковую длину. Редактор связей автоматически округляет размер виртуального оверлейного сегмента до величины, кратной 32. Словам или 100(8) байтам. Для этого редактор связей суммирует слово, содержащее номер оверлейного сегмента, с его размером (число 000076, которое следует за 040002 в строке 48)
53	Точка входа и старший адрес (последний адрес нижней памяти, используемый корневым и неотображенными сегментами) редактируемой программы
56	Виртуальный старший адрес (указывает последний виртуальный адрес, используемый частью программы, расположенной в расширенной памяти), очередной свободный адрес (адрес очередной страницы, которая не используется программой)
59	Количество расширенной памяти, требуемое программе. Следует убедиться, что оно достаточно
60—87	Таблица перекрестных ссылок глобальных имен (см. п. 5.2.13)

6.4. Файл определений имен. Файл определений имен — это файл в объектном формате, содержащий каталог глобальных имен, определенных во входных объектных модулях. Этот файл можно использовать для разрешения глобальных имен при раздельном редактировании модулей.

6.5. Использование библиотек. Библиотеки объектных модулей могут использоваться редактором связей для поиска модулей, включаемых в редактируемую программу. В редактируемую программу включаются из библиотеки только те модули, к которым обращаются из небиблиотечных файлов. Количество указываемых библиотек не ограничено. Системная библиотека SY:SYSLIB.OBJ обрабатывается по умолчанию.

Библиотечные файлы должны указываться в командных строках, в которых указываются файлы для формирования корневого сегмента. Указываются библиотечные файлы так же, как и объектные файлы. Недопустимо указывать библиотечные файлы в одной командной строке с оверлейными сегментами.

Редактор связей обрабатывает библиотечные файлы следующим образом. Во время первого прохода редактор связей обрабатывает входные файлы в том порядке, в котором они указаны в командной строке. При этом на начальном этапе первого прохода обрабатываются только небиблиотечные файлы. На конечном этапе первого прохода обрабатываются только библиотечные файлы. В это время разрешаются неопределенные глобальные имена, к которым были обращения из небиблиотечных файлов. Файл системной библиотеки всегда обрабатывается последним.

Пример:

*TASK01,LP:=MAIN,MEASUR

В этом примере библиотечный модуль MEASUR.OBJ служит для разрешения глобальных имен, неопределенных в программе MAIN.OBJ. Для разрешения оставшихся неопределенных глобальных имен используется системная библиотека SY:SYSLIB.OBJ.

Модули одной библиотеки могут вызывать модули других библиотек, поэтому для правильного разрешения глобальных имен следует соблюдать в командных строках определенный порядок задания библиотек.

Пусть модуль X из библиотеки ALIB вызывает модуль Y из библиотеки BLIB. Тогда в командной строке спецификация библиотеки ALIB должна предшествовать спецификации библиотеки BLIB: *Z=B,ALIB,BLIB. Модуль B — корневой.

6.5.1. Библиотеки кратных определений.

Кроме обычных библиотек, редактор связей обрабатывает библиотеки кратных определений. Основная цель этих библиотек состоит в обеспечении специальных функций для систем реального времени. Библиотеки кратных определений отличаются от обычных библиотек тем, что они содержат несколько определений одного глобального имени. Библиотеки кратных определений указываются в командных строках так же, как и обычные библиотеки.

Модули из библиотек кратных определений всегда помещаются в корневой сегмент. При обработке этих модулей редактор связей помещает справочник библиотеки кратных определений, называемый таблицей точек входа, во внутренний буфер. Если размер внутреннего буфера недостаточен для размещения таблицы точек входа, редактор связей выдает сообщение:

LINK-F-LIBRARY EPT TOO BIG, INCREASE BUFFER WITH /G

Переключатель /G увеличивает размер буфера в соответствии с наибольшим размером таблицы точек входа.

Когда глобальное имя из модуля библиотеки кратных определений совпадает с неопределенным глобальным именем, редактор связей удаляет из списка неопределенных глобальных имен другие глобальные имена, определенные в этом модуле. Таким образом, два модуля с одинаковыми именами не будут находиться в редактируемой программе.

ПРИМЕЧАНИЕ. Порядок модулей в библиотеках кратных определений влияет на их использование редактором связей.

7. СООБЩЕНИЯ ОПЕРАТОРУ

Часть сообщений, выдаваемых редактором связей, относится к действию переключателей, для реализации которых требуется дополнительная информация (см. табл. 11). Действия по этим сообщениям описаны в п. 5.2.

Таблица 11

Сообщение	Переключатель
1	2
TRANSFER SYMBOL?	/T
STACK SYMBOL?	/M
EXTEND SECTION?	/E:N
BOUNDARY SECTION?	/Y:N
ROUND SECTION?	/U:N
LOAD SECTION: ADDRESS?	/Q
LIBRARY SEARCH?	/I
DUPLICATE SYMBOL?	/D

Сообщения приведены в том порядке, в котором они выдаются на терминал. Это следует учитывать при создании косвенных файлов.

Ниже приведены сообщения об ошибках, выдаваемые редактором связей.

?LINK—F—ADDRESS SPACE EXCEEDED

Причина. Размер программы превышает 32К слов.

Действие. Сократить размер программы, используя оверлейную структуру. Повторить редактирование.

?LINK—F—ASECT TOO BIG

Причина. Наложение абсолютного кода программы на относительный код.

Действие. Сократить размер абсолютной или относительных программных секций. Повторить редактирование.

?LINK—F—/B NO VALUE

Причина. Не указан аргумент переключателя /B.

Действие. Ввести правильную командную строку. Повторить редактирование.

?LINK—F—/B ODD VALUE

Причина. Значение аргумента переключателя /B нечетно.

Действие. Ввести правильную командную строку. Повторить редактирование.

?LINK—F—CROSS REFERENCE DEVICE FULL DEV:FILNAM.TYP

Причина. На томе, используемом для вывода таблицы перекрестных ссылок, недостаточно свободного места или полностью заполнен справочник.

Действие. Освободить место на томе или использовать другой том. Повторить редактирование.

?LINK—F—/E NO VALUE

Причина. Не указан аргумент переключателя /E.

Действие. Ввести правильную командную строку. Повторить редактирование.

?LINK—F—/E ODD VALUE

Причина. Значение аргумента переключателя /E нечетно.

Действие. Ввести правильную командную строку. Повторить редактирование.

?LINK—F—/H NO VALUE

Причина. Не указан аргумент переключателя /H.

Действие. Ввести правильную командную строку. Повторить редактирование.

?LINK—F—/H ODD VAULE

Причина. Значение аргумента переключателя /H нечетно.

Действие. Ввести правильную командную строку. Повторить редактирование.

?LINK—F—/H VALUE TOO LOW

Причина. Значение аргумента переключателя /H:N меньше допустимого.

Действие. Получить карту загрузки редактируемой программы и определить объем памяти, требуемый программе. Повторить редактирование, указав переключатель /H:N с допустимым значением аргумента.

?LINK—F—INPUT ERROR DEV:FILNAM.TYP

Причина. Ошибка при чтении входного файла.

Действие. Проверить готовность и исправность оборудования. Повторить редактирование.

?LINK—F—INSUFFICIENT MEMORY

Причина. Недостаточно памяти.

Действие. Освободить часть оперативной памяти (удалить основное задание, использовать монитор одного задания, удалить ненужные драйверы). Повторить редактирование.

?LINK—F—INTERNAL ERROR

Причина. Ошибка при обращении к программе LINK.

Действие. Повторить редактирование. При повторении этой же ошибки использовать новую копию программы редактор связей.

?LINK—F—INVALID CHARACTER

Причина. Введен знак, не используемый в RADIX-50.

Действие. Ввести правильное имя. Повторить редактирование.

?LINK—F—INVALID COMPLEX RELOCATION IN DEV:FILNAM.TYP

Причина. Ошибка в записи составного перемещения в объектном модуле входного файла.

Действие. Проверить правильность командной строки. Вновь получить объектные модули, входящие во входной файл. Повторить редактирование.

?LINK—F—INVALID DEVICE DEV:

Причина. Указанное устройство не обслуживается системой.

Действие. Установить в системе обслуживание требуемого устройства или использовать другое устройство. Повторить редактирование.

?LINK—F—INVALID GSD IN DEV:FILNAM.TYP

Причина. Ошибка в каталоге глобальных имен объектного модуля входного файла.

Действие. Проверить правильность командной строки. Вновь получить объектные модули, входящие во входной

файл. Повторить редактирование.

?LINK—F—INVALID RECORD TYPE IN DEV:FILNAM.TYP

Причина. Недопустимый формат входного файла.

Действие. Проверить правильность командной строки. Вновь получить объектные модули, входящие во входной файл. Повторить редактирование.

?LINK—F—INVALID RLD IN DEV:FILNAM.TYP

Причина. Ошибка в словаре перемещений объектного модуля входного файла.

Действие. Проверить правильность командной строки. Вновь получить объектные модули, входящие во входной файл. Повторить редактирование.

?LINK—F—INVALID RLD SYMBOL IN DEV:FILNAM.TYP

Причина. В словаре перемещений объектного модуля входного файла используется глобальное имя, не указанное в каталоге глобальных имен этого модуля.

Действие. Проверить правильность командной строки. Вновь получить объектные модули, входящие во входной файл. Повторить редактирование.

?LINK—F—/K INVALID VALUE

Причина. Указано недопустимое значение аргумента переключателя /K.

Действие. Ввести правильную командную строку. Повторить редактирование.

?LINK—F—/K NO VALUE

Причина. Не указан аргумент переключателя /K.

Действие. Ввести правильную командную строку. Повторить редактирование.

?LINK-F-LIBRARY EPT TOO BIG, INCREASE BUFFER WITH /G

Причина. Переполнение таблицы точек входа.

Действие. Повторить редактирование, указав переключатель /G в первой командной строке.

?LINK-F-LIBRARY LIST OVERFLOW, INCREASE SIZE WITH /P

Причина. Переполнение таблицы имен библиотечных программ.

Действие. Повторить редактирование, увеличив с помощью переключателя /P размер таблицы имен библиотечных программ.

?LINK—F—/M ODD VALUE

Причина. В качестве адреса стека указано нечетное значение.

Действие. Ввести правильную командную строку. Повторить редактирование.

?LINK—F—MAP DEVICE FULL DEV:FILNAM.TYP

Причина. На томе, используемом для вывода файла карты загрузки, недостаточно свободного места или полностью заполнен справочник.

Действие. Освободить место на томе или использовать другой том.

?LINK—F—OLD LIBRARY FORMAT IN DEV:FILNAM.TYP

Причина. Формат библиотечного файла не соответствует формату, используемому в ФОДОС-2 версии В03.00.

Действие. Вновь сформировать библиотечный файл с помощью программы библиотекарь. Повторить редактирование.

?LINK—F—PROTECTED FILE ALREADY EXISTS DEV:FILNAM.TYP

Причина. Попытка открыть файл с таким же именем, как у имеющегося защищенного файла.

Действие. Отменить защиту имеющегося файла или использовать другое имя для открытия нового файла. Повторить редактирование.

?LINK—F—/R ODD VALUE

Причина. Значение аргумента переключателя /R нечетно.

Действие. Ввести правильную командную строку. Повторить редактирование.

?LINK—F—REL WRITE BEYOND EOF

Причина. Переполнение файла загрузочного модуля в формате REL при записи информации о перемещениях.

Действие. Повторить редактирование, используя в командной строке конструкцию [N] для задания размера файла загрузочного модуля.

?LINK—F—SAV DEVICE FULL DEV:FILNAM.TYP

Причина. На томе, используемом для вывода файла загрузочного модуля, недостаточно свободного места или полностью заполнен справочник.

Действие. Освободить место на томе или использовать другой том. Повторить редактирование.

?LINK—F—SAV READ ERROR

Причина. Ошибка при считывании формируемого файла загрузочного модуля.

Действие. Проверить готовность и исправность оборудования. Повторить редактирование.

?LINK—F—SAV WRITE ERROR

Причина. Ошибка при записи файла загрузочного модуля.

Действие. Проверить готовность и исправность оборудования. Проверить том на плохие блоки. Повторить редактирование.

?LINK—F—SIZE OVERFLOW OF SECTION AAAAAA

Причина. Размер программной секции превышает 32K слов.

Действие. Сократить размер программной секции AAAAAA или всей программы. Повторить редактирование.

?LINK—F—STB DEVICE FULL DEV:FILNAM.TYP

Причина. На томе, используемом для вывода файла определений имен, недостаточно свободного места или полностью заполнен справочник.

Действие. Освободить место или использовать другой том. Повторить редактирование.

?LINK—F—STB NOT ALLOWED WITH /S AND A MAP

Причина. Попытка получить файл определений имен и карту загрузки одновременно с указанием переключателя /S.

Действие. Отдельно получить файл определений имен или карту загрузки. Повторить редактирование.

?LINK—F—STB WRITE ERROR

Причина. Ошибка при записи файла определений имен.

Действие. Проверить готовность и исправность оборудования. Проверить том на наличие плохих блоков. Повторить редактирование.

?LINK—F—STORING TEXT BEYOND HIGH LIMIT

Причина. Ошибка во входных объектных модулях или на томе, используемом для вывода файла в формате LDA, недостаточно свободного места.

Действие. Вновь получить объектные модули и повторить редактирование.

?LINK—F—SYMBOL TABLE OVERFLOW

Причина. Переполнение таблицы имен.

Действие. Повторить редактирование, указав переключатель /S. При появлении этой же ошибки освободить часть оперативной памяти (удалить основное задание, использовать монитор одного задания, удалить ненужные драйверы) и повторить редактирование.

?LINK—F—/T ODD VALUE

Причина. В качестве адреса точки входа редактируемой программы указано нечетное значение.

Действие. Ввести правильную командную строку. Повторить редактирование.

?LINK—F—TOO MANY PROGRAM SEGMENTS

Причина. Указано более 1777 (восьмеричное) программных сегментов.

Действие. Сократить число оверлейных сегментов. Повторить редактирование.

?LINK—F—TOO MANY VIRTUAL OVERLAY REGIONS

Причина. Указано более 8 оверлейных областей (окон) в расширенной памяти, включая корневой сегмент.

Действие. Сократить число оверлейных областей в расширенной памяти. Повторить редактирование.

?LINK—F—/U OR /Y VALUE NOT A POWER OF 2

Причина. Значение аргумента переключателя /U или /Y не является степенью числа 2.

Действие. Ввести правильную командную строку. Повторить редактирование.

?LINK—F—WORD RELOCATION ERROR IN FILNAM

Причина. Попытка обратиться к слову по нечетному адресу во время объединения данных программных секций.

Действие. Поместить директиву АССЕМБЛЕРа .EVEN в конце программных секций, содержащих данные, чтобы обеспечить правильность обращения к данным.

?LINK—F—/Y NO VALUE

Причина. Не указан аргумент переключателя /Y.

Действие. Ввести правильную командную строку. Повторить редактирование.

?LINK—F—/Y ODD VALUE

Причина. Значение аргумента переключателя /Y нечетно.

Действие. Ввести правильную командную строку. Повторить редактирование.

?LINK-F-VIRTUAL OVERLAY LOGICAL ADDRESS SPACE EXCEEDED

Причина. Размер оверлеев превышает 96К слов расширенной памяти.

Действие. Повторить редактирование, указав раздел М в переключателе /V:N[:M] (см. п. 5.2.21), который совместно используют несколько сегментов.

?LINK—W—ADDITIVE REVERENCE OF NNNNNN AT SEGMENT #MMMMMM

Причина. Попытка передать управление в оверлейный сегмент MMMMMM не в точку входа этого сегмента;

NNNNNN — точка входа. Редактор связей формирует команду передачи управления непосредственно в точку входа сегмента. Редактирование продолжается.

Действие. Исправить исходную программу. Повторить редактирование.

?LINK—W—BOUNDARY SECTION NOT FOUND

Причина. В корневом сегменте нет программной секции, к которой относится действие переключателя /Y. Переключатель /Y игнорируется, редактирование продолжается.

Действие. Повторить редактирование, указав имя требуемой программной секции.

?LINK—W—BYTE RELOCATION ERROR AT NNNNNN

Причина. Ненулевой старший байт перемещаемой величины. NNNNNN — адрес, по которому находится эта величина. Редактор связей усекает перемещаемую величину до 8 разрядов, если формат загрузочного модуля SAV или LDA. Для загрузочного модуля в формате REL усечение не производится. Редактирование продолжается.

Действие. Исправить исходную программу и повторить редактирование.

?LINK-W-COMPLEX RELOCATION DIVIDE BY 0 IN DEV:FILNAM.TYP:

Причина. Деление на 0 в записи составного перемещения в указанном файле. Результат операции равен 0. Редактирование продолжается.

Действие. Исправить исходную программу. Повторить редактирование.

?LINK—W—COMPLEX RELOCATION OF AAAAAA

Причина. Попытка составного перемещения глобального имени AAAAAA при редактировании основного задания.

Действие. Удалить все составные перемещения. Повторить редактирование.

?LINK—W—CONFLICTING SECTION ATTRIBUTES NNNNNN

Причина. При повторном описании программной секции NNNNNN значения признаков отличаются от значений из первого описания. Редактор связей использует признаки из первого определения. Редактирование продолжается.

Действие. Исправить исходную программу. Повторить редактирование.

?LINK—W—CROSS REFERENCE INPUT ERROR

Причина. Ошибка при чтении таблицы перекрестных ссылок. Часть карты загрузки, содержащая таблицу перекрестных ссылок, удаляется, редактирование продолжается.

Действие. Проверить готовность и исправность оборудования. Повторить редактирование.

?LINK—W—CROSS REFERENCE OUTPUT ERROR

Причина. Ошибка при записи таблицы перекрестных ссылок. Часть карты загрузки, содержащая таблицу перекрестных ссылок, удаляется, редактирование продолжается.

Действие. Проверить готовность и исправность оборудования. Повторить редактирование.

?LINK-W-DEFAULT SYSTEM LIBRARY NOT FOUND SYSLIB.OBJ

Причина. Не найден файл системной библиотеки SY:SYSLIB.OBJ.

Действие. Сформировать файл системной библиотеки или исправить исходную программу. Повторить редактирование.

?LINK-W-DUPLICATE SYMBOL 'SYMBOL' DEFINED IN DEV:FILNAM.TYP

Причина. Попытка дублировать небиблиотечный файл с помощью переключателя /D.

Действие. Поместить модуль, содержащий глобальное имя 'SYMBOL', в библиотеку объектных модулей. Повторить редактирование.

?LINK-W-DUPLICATE SYMBOL 'SYMBOL' IS FORCED TO THE ROOT

Причина. Модуль, содержащий глобальное имя 'SYMBOL', помещен в корневой сегмент.

Действие. Устранить все обращения к глобальному имени 'SYMBOL'. Повторить редактирование.

?LINK—W—EXTEND SECTION NOT FOUND

Причина. Не найдена программная секция, к которой относится действие переключателя /E. Переключатель /E игнорируется, редактирование продолжается.

Действие. Повторить редактирование, указав имя требуемой программной секции.

?LINK—W—FILE NOT FOUND DEV:FILNAM.TYP

Причина. Не найден указанный входной файл.

Действие. Проверить, существует ли файл с указанным именем. Ввести правильную командную строку. Повторить редактирование.

?LINK—W—INVALID OPTION: /X

Причина. Переключатель /X недопустим или используется в недопустимой комбинации. Если ошибка допущена в первой командной строке, то редактор связей игнорирует эту командную строку. Если ошибка допущена в последующих командных строках, то редактор связей игнорирует только переключатель /X.

Действие. Ввести правильную командную строку. Повторить редактирование.

?LINK—W—LOAD ADDRESS ODD

Причина. Указан нечетный адрес загрузки программной секции, к которой относится действие переключателя /Q.

Действие. Ввести правильную командную строку, повторить редактирование.

?LINK—W—LOAD ADDRESS TOO LOW PSECT

Причина. Значение адреса загрузки программной секции меньше допустимого.

Действие. Повторить редактирование, указав больший адрес загрузки программной секции.

?LINK—W—LOAD SECTION NOT FOUND PSECT

Причина. Не найдена указанная секция загрузки в корневом сегменте. Секция игнорируется, редактирование продолжается.

Действие. Повторить редактирование, поместив секцию загрузки в корневой сегмент.

?LINK—W—MAP WRITE ERROR

Причина. Ошибка при записи файла карты загрузки. Файл карты загрузки не выводится. Редактирование продолжается.

Действие. Проверить готовность и исправность оборудования. Проверить том на наличие плохих блоков. Повторить редактирование.

?LINK—W—MULTIPLE DEFINITION OF 'SYMBOL'

Причина. Многократное определение глобального имени 'SYMBOL'. Редактор связей использует первое встреченное определение имени 'SYMBOL'. Редактирование продолжается.

Действие. Исправить исходную программу. Повторить редактирование.

?LINK—W—NO LOAD ADDRESS

Причина. Не указан адрес загрузки программной секции, к которой относится действие переключателя /Q.

Действие. Ввести правильную командную строку. Повторить редактирование.

?LINK—W—/O OR /V OPTION ERROR
RE—ENTER LINE

Причина. Переключатели /O или /V указаны в неправильном порядке или используются в недопустимой комбинации.

Действие. Ввести правильную командную строку. Повторить редактирование.

?LINK—W—ROUND SECTION NOT FOUND
AAAAAA

Причина. В корневом сегменте нет программной секции, к которой относится действие переключателя /U. Переключатель /U игнорируется, редактирование продолжается.

Действие. Повторить редактирование, указав имя требуемой программной секции.

?LINK—W—STACK ADDRESS UNDEFINED OR IN OVERLAY

Причина. В корневом сегменте не определено глобальное имя, к которому относится действие переключателя /M. Переключатель /M игнорируется, редактирование продолжается.

Действие. Повторить редактирование, указав требуемое глобальное имя.

?LINK-W-TRANSFER ADDRESS UNDEFINED OR IN OVERLAY

Причина. В корневом сегменте не определено глобальное имя, к которому относится действие переключателя /T. Переключатель /T игнорируется, редактирование продолжается.

Действие. Повторить редактирование, указав требуемое глобальное имя.

?LINK—W—UNDEFINED GLOBALS:
AAAAAA
BBBBBB
.
.
.
.

Причина. Глобальные имена АААААА, ВВВВВВ .... не определены.

Действие. Проверить правильность командной строки. Исправить исходную программу. Повторить редактирование.

ПЕРЕЧЕНЬ ССЫЛОЧНЫХ ДОКУМЕНТОВ

1. Операционная система ФОДОС-2
   АССЕМБЛЕР
   Описание языка
2. Операционная система ФОДОС-2
   Командный язык системы
3. Операционная система ФОДОС-2
   Монитор расширенной памяти
   Руководство программиста
4. Операционная система ФОДОС-2
   АССЕМБЛЕР
   Руководство программиста

БИБЛИОТЕКАРЬ

РУКОВОДСТВО ОПЕРАТОРА

1. НАЗНАЧЕНИЕ ПРОГРАММЫ И УСЛОВИЯ ВЫПОЛНЕНИЯ ПРОГРАММЫ

Программа библиотекарь (LIBR) предназначена для создания и корректировки библиотек (библиотечных файлов) операционной системы ФОДОС-2.

В операционной системе имеется два типа библиотек:

— библиотека объектных модулей;

— библиотека макроопределений.

По умолчанию имя системной библиотеки объектных модулей — SYSLIB.OBJ. Для библиотек объектных модулей библиотекарь позволяет выполнять такие операции как создание новых библиотек, включение новых модулей в библиотеку, удаление модулей из библиотеки, извлечение модулей из библиотеки.

По умолчанию имя системной библиотеки макроопределений — SYSMAC.SML. Для библиотеки макроопределений библиотекарь позволяет выполнять только операцию создания новых библиотек.

2. ФОРМАТ БИБЛИОТЕЧНЫХ ФАЙЛОВ

Каждый библиотечный файл имеет заголовок, таблицу точек входа (таблицу глобальных имен или таблицу макроопределений), называемую каталогом, содержащую записи о расположении конкретного модуля библиотечного файла на диске и набор модулей. Библиотека заканчивается специальным блоком конца библиотеки (рис. 1).

Заголовок библиотеки
Каталог
Набор модулей
Блок конца библиотеки

Рис. 1

2.1. Формат заголовка библиотеки.

Блок заголовка библиотеки описывает текущее состояние библиотеки (табл. 1, табл. 2).

Таблица 1

ФОРМАТ ЗАГОЛОВКА БИБЛИОТЕКИ ОБЪЕКТНЫХ МОДУЛЕЙ

Смещение	Содержимое	Описание
1	2	3
0	1	Код блока заголовка библиотеки
2	42	
4	7	Код библиотекаря
6	500	Номер версии библиотеки
10	0	Зарезервировано
12		Дата (0, если не указана)
14		Время
16		
20	0	1 ,если библиотека создана с использованием переключателя /X
22	0	Зарезервировано
24	0	Зарезервировано
26	10	Относительный адрес начала каталога
30		Количество байтов в каталоге
32	0	Зарезервировано
34		Относительный номер блока для включения следующего модуля в библиотеку
36		Следующий байт внутри блока
40		Начало каталога

Таблица 2

ФОРМАТ ЗАГОЛОВКА БИБЛИОТЕКИ МАКРООПРЕДЕЛЕНИЙ

Смещение	Содержимое	Описание
1	2	3
0	1001	Тип библиотеки и код идентификации
2	500	Номер версии библиотеки
4	0	Зарезервировано
6		Дата (0, если не указана)
10		Время
12		
14	0	Зарезервировано
16	0	Зарезервировано
20	0	Зарезервировано
22	0	Зарезервировано
24	0	Зарезервировано
26	0	Зарезервировано
30	0	Зарезервировано
32	10	Размер записей каталога
34		Относительный номер блока начала каталога
36		Количество записей, которые можно разместить в каталоге (по умолчанию 200)
40		Количество доступных записей каталога

2.2. Формат таблицы точек входа

Таблица точек входа состоит из записей размером 4 слова, содержащих имя точки входа (слова 0,1) и адрес модуля, соответствующий точке входа (слова 2,3). Если при создании библиотеки используется переключатель /N, то в 15 разряде

слова, содержащего относительный номер блока, устанавливается 1. Формат таблицы точек входа приведен на рис. 2.

Имя точки входа	
(RADIX-50)	
Относительный номер блока	
Зарезервировано (7 битов)	Относительный байт в блоке (9 битов)

Рис. 2

По умолчанию под таблицу точек входа макробиблиотеки отводится 2 блока. Размер таблицы точек входа библиотеки объектных модулей заранее не определяется.

2.3. Формат блока конца библиотеки

Каждая библиотека заканчивается блоком конца библиотеки (рис. 3).

1	Заголовок блока
10	Длина блока
10	Код блока конца библиотеки
0	Зарезервировано
357	Байт контрольной суммы

Рис. 3

3. ВЫПОЛНЕНИЕ ПРОГРАММЫ

3.1. Пуск программы

Для вызова программы библиотекарь (LIBR) с системного устройства следует подать с терминала команду .R LIBR <ВК> или LIBR <ВК> после того, как монитор напечатает на терминале точку.

После вызова библиотекарь печатает звездочку и ожидает ввода командной строки. Если в это время нажать клавишу <ВК>, то библиотекарь печатает номер своей версии.

ПРИМЕЧАНИЕ. Операции, выполняемые программой библиотекарь, могут быть выполнены по команде LIBR (см. [1]).

3.2. Команды оператора

Режим работы программы библиотекарь задается введением с терминала командной строки следующего формата:
выходспф,лстспф = входспф,...,входспф [/прк...],
где выходспф — спецификация выходного файла (устройство, имя и тип файла);
лстспф — спецификация листинга каталога библиотечного файла;
входспф — спецификация входного файла;
/прк — переключатель (табл. 3).

По умолчанию предполагается:
— тип входного и выходного файла .OBJ;
— тип файла листинга .LST.

Команда может состоять из нескольких командных строк (п. 3.4).

Входной файл может состоять из одного или нескольких объектных модулей. После помещения входного файла в библиотечный файл к любому из составляющих его модулей можно обращаться только по имени, указанному в директиве АССЕМБЛЕРа .TITLE или в операторах ФОРТРАНа PROGRAM и SUBROUTINE, а не по имени файла.

Таблица 3

ПЕРЕКЛЮЧАТЕЛИ ПРОГРАММЫ БИБЛИОТЕКАРЬ

Переключатель	Функция
1	2
/A	Заносит в каталог библиотечного файла все глобальные имена, включая все абсолютные глобальные имена
/C или //	Продолжение; позволяет печатать команду более чем на одной строке
/D	Удаление; удаляет модули и соответствующие им глобальные имена из библиотечного файла
/E	Выборка; выбирает модуль из библиотечного файла и запоминает его в файле типа .OBJ.
/G	Удаление глобальных имен; удаляет глобальные имена из каталога библиотечного файла
/N	Наименование; включает имена модулей в каталог библиотечного файла

1	2
/R	Замена; заменяет модули в библиотечном файле
/U	Корректировка; включает и заменяет модули в библиотечном файле
/X	Позволяет вносить многократные определения глобальных имен в каталог библиотечного файла
/W	Указывает формат для файла листинга.
/M	MACRO; создает файл макробиблиотеки.

Для выхода из программы LIBR и передачи управления монитору следует подать команду СУ/С, если LIBR ожидает ввода с терминала, или дважды СУ/С, если LIBR выполняет операцию.

3.3. Переключатель /А.

При использовании переключателя /А в каталог библиотечного файла заносятся не только глобальные имена, но и абсолютные глобальные имена.

Пример:

```
*ALIB=MOD1,MOD2/A
```

Создается библиотечный файл ALIB.OBJ, в каталог которого заносятся все глобальные и абсолютные глобальные имена модулей файлов MOD1, MOD2.

3.4. Продолжение (/С или //).

Максимальное количество входных файлов, которые можно записать в одной строке, — шесть. Чтобы ввести более шести файлов, нужно использовать переключатель /С или //.

Переключатель /С печатается в конце первой строки и повторяется в конце последующих строк. На последней строке переключатель /С не указывается. Каждая строка продолжения командной строки может содержать только входные спецификации файлов.

Переключатель // печатается в конце первой и в конце последней строки.

Примеры:

1.
```
*ALIB,LIBLST=MAIN,TEST,FXN/C
*TRAS
```

Создается библиотечный файл ALIB.OBJ, который будет состоять из модулей, входящих в файлы MAIN,TEST,FXN, TRAC, также создается листинг LIBLST.LST.

2.
```
*BLIB=MAIN,TEST,FXN//
*TRACK
*//
```

Создается библиотечный файл с именем BLIB.OBJ из модулей, находящихся в файлах MAIN, TEST,FXN, TRACK.

3.5. Создание библиотечного файла.

При создании библиотечного файла его имя указывается в выходной спецификации командной строки.

Пример:

*BOT=AP,ET

Библиотечный файл, называемый BOT.OBJ, будет состоять из модулей, находящихся в файлах AP.OBJ и ET.OBJ.

3.6. Включение модулей в библиотеку.

Если для входного файла в командной строке не указан ни один из переключателей, то выполняется операция включения модулей в библиотечный файл. В этом случае библиотечный файл должен быть указан в выходной и входной спецификации командной строки.

Примеры:

1. *EXY=EXY, DX1:PA,PB,PC

Модули в файлах PA.OBJ,PB.OBJ,PC.OBJ на устройстве DX1: будут включены в библиотечный файл EXY.OBJ на устройстве DK:.

2. *DXYNEW=DXY,FA,FB,FC

В этом примере новый библиотечный файл состоит из модулей библиотечного файла DXY и новых модулей из файлов FA, FB, FC.

3.7. Удаление (/D).

По переключателю /D удаляются модули и соответствующие им глобальные имена из каталога библиотечного файла, а не из самого библиотечного файла. Когда используется переключатель /D, библиотекарь выдает: MODULE NAME?

Пользователю необходимо ввести имя удаляемого модуля и возврат каретки. Этот процесс надо продолжать до тех пор, пока не будут напечатаны имена всех удаляемых модулей. После подачи только возврата каретки начинается выполнение командной строки.

Пример:

```
*DX1:TRAP=DX1:TRAP/D,AEC
MODULE NAME? AA
MODULE NAME? BB
MODULE NAME?
```

Модули AA.OBJ и BB.OBJ удаляются из библиотечного файла TRAP.OBJ, а модули из файла AEC.OBJ включаются в библиотечный файл.

3.8. Выборка (/E).

По переключателю /E выбирается модуль из библиотеч-

ного файла и запоминается в файле типа .OBJ. Когда используется переключатель /E, библиотекарь печатает: GLOBAL?

Пользователю следует ввести глобальное имя из выбираемого модуля и возврат каретки. После подачи только возврата каретки начинается выполнение командной строки. Библиотекарь выбирает весь модуль, которому принадлежит глобальное имя. Переключатель /E не должен использоваться в командной строке с другими переключателями.

Пример:

```
*DX1:ATAP=SYSLIB/E
GLOBAL? ATAP
GLOBAL?
```

Выбирается модуль ATAP из библиотечного файла SYSLIB.OBJ на устройстве DK: и запоминается в файле с именем ATAP.OBJ на устройстве DX1:.

3.9. Удаление глобальных имен (/G).

По переключателю /G удаляются глобальные имена из каталога библиотечного файла. Когда используется переключатель /G, библиотекарь печатает: GLOBAL?

Пользователю следует ввести глобальное имя, которое должно быть удалено, и возврат каретки. Этот процесс надо продолжать до тех пор, пока не будут перечислены все глобальные имена, которые необходимо удалить. Только после подачи возврата каретки начинается выполнение командной строки.

Пример:

```
*ROL=ROL/G
GLOBAL? MEA
GLOBAL? MEB
GLOBAL?
```

Удаляются глобальные имена MEA и MEB из каталога библиотечного файла ROL.OBJ.

По переключателю /G удаляются глобальные имена только из каталога, а не из самого библиотечного файла.

3.10. Наименование (/N).

По переключателю /N включаются имена модулей в каталог библиотечного файла. Если имена модулей не включены в каталог библиотечного файла, то колонка MODULE листинга каталога остается пустой. Символ плюс (+) в колонке MODULE указывает на продолжение строки для печати всех глобальных имен модуля.

Если библиотечный файл не имеет в своем каталоге имен модулей, можно создать новый библиотечный файл для включения имен модулей. Ниже приводится пример, в котором

временно создается новый библиотечный файл из текущего библиотечного файла, и каталог выводится на терминал. Текущий библиотечный файл OLDLIB.OBJ остается неизменным.

Пример:

```
*DX1:TEMP,TT:=OLDLIB/N
FODOS LIBRARIAN BO3.00 TUE 09-DEC-83 14:40:30
TEMP                   TUE 09-DEC-83 14:40:29
MODULE   GLOBALS  GLOBALS  GLOBALS
IRAD50   IRAD50   RAD50
MYC      MYC
CAB      CAB
BOB      BOB
```

Включаются имена всех модулей из библиотечного файла OLDLIB.OBJ в каталог библиотечного файла TEMP.OBJ, и листинг этого каталога выводится на терминал.

3.11. Замена (/R).

По переключателю /R заменяются модули библиотечного файла на модули с теми же именами из входных файлов. Переключатель /R должен следовать за каждой входной спецификацией файла, содержащей модули для замены.

Пример:

*TFIL=TFIL, INA, INB/R, INC

Модули в файле INB.OBJ заменяют существующие модули с теми же именами в библиотечном файле TFIL.OBJ. Модули из файлов INA.OBJ и INC.OBJ будут включены в библиотечный файл TFIL.OBJ.

3.12. Корректировка (/U).

По переключателю /U корректируется библиотечный файл путем объединения операций замены и включения. Если модули из входного файла уже существуют в библиотечном файле, они будут заменять старые модули; если нет, они будут включены в библиотечный файл. Переключатель /U должен следовать за каждой входной спецификацией файла, которая содержит модули для корректировки.

Пример:

*BALIB=BALIB,FOT/U,TAC,BAT/U

Модули в файлах FOT.OBJ и BAT.OBJ заменяют модули с теми же именами в библиотечном файле BALIB.OBJ. Если нет модулей для замены, то эти модули будут включены в библиотечный файл BALIB.OBJ. Модуль из файла TAC.OBJ будет также включен в библиотечный файл BALIB.OBJ.

3.13. Переключатель /X

Переключатель /X позволяет создавать библиотечные файлы, в которых два или несколько библиотечных модуля мо-

гут иметь одно и то же глобальное имя. При использовании переключателя /X библиотекарь не выдает сообщение:

?LIBR—W—ILLEGAL INSERT OF AAAAAA,

когда встречается дубликат глобального имени и глобальное имя заносится в каталог библиотечного файла.

Пример:

```
*MLTLIB,TT:=MOD1,MOD2,MOD3/X/A
FODOS LIBRARIAN B03.00 THU 25-APR-83 09:45:31
DK:MLTLIB.OBJ          THU 25-APR-83 09:45:31
MODULE     GLOBALS     GLOBALS     GLOBALS
MOD1       OMA&R       SWP&        ATP&
MOD2       ATP&        OMA&R       MER&CR
           LBM
MOD3       ATP&        OMA&R       MEP&CR
           ENTZ
```

В этом примере создается библиотечный файл MLTLIB из модулей MOD1, MOD2, MOD3 и каталог библиотечного файла выводится на терминал. Библиотечный файл состоит из модулей, которые используют одинаковые глобальные имена. Кроме того, модуль MOD3 содержит абсолютные глобальные имена, поэтому используется переключатель /A.

3.14. Переключатель /W дает возможность получать листинг каталога библиотечного файла с шестью колонками GLOBAL. Такой листинг может быть получен на построчно-печатающем устройстве или терминале, который имеет 132 колонки.

3.15. Получение листинга каталога.

Режим получения листинга каталога библиотечного файла задается введением с терминала командной строки следующего формата:

*,лстспф=входспф, где лстспф — спецификация листинга каталога библиотечного файла (устройство, имя и тип файла);

входспф — спецификация библиотечного файла.

Пример:

```
*,LIST=LIBFIL
```

Файл листинга каталога LIST.LST библиотечного файла LIBFIL.OBJ выводится на устройство DK:.

Пример:

```
*,TT:=SYSLIB
FODOS LIBRARIAN B03.00 TUE 09-DEC-83 14:50:40
SYSLIB                 TUE 09-DES-83 14:50:39
MODULE     GLOBALS     GLOBALS     GLOBALS
           DSO¤        ECO¤        FCO¤
  +        GCO¤        RCI
           DIC¤IS      DTC¤MS      DIC¤PS
  +        DIC¤CC
```

Первая строка листинга указывает версию библиотекаря, который использовался, текущую дату и время. Вторая строка печатает наименование библиотечного файла, время и дату его создания. Имена модулей не включены в этот пример. Символ плюс (+) в колонке MODULE указывает на продолжение командной строки для печати всех глобальных имен модуля.

3.16. Объединение библиотечных файлов.

Два или несколько библиотечных файлов могут быть объединены под одним именем. Для этого пользователю необходимо указать все библиотечные файлы, которые будут объединены, в одной команде.

Пример:

```
*FORT=A,B,C
```

Библиотечные файлы A.OBJ, B.OBJ и C.OBJ будут объединены в библиотечный файл под именем FORT.OBJ.

3.17. Объединение операций библиотекаря.

Пользователь может запрашивать в одной команде несколько операций. Библиотекарь выполняет операции в следующем порядке:

— продолжение;
— удаление;
— удаление глобальных имен;
— корректировка;
— замена;
— включение;
— листинг.

Пример:

```
*FILE,LP:=FILE/D,MODX,MODY/R
MODULE NAME? XYZ
MODULE NAME? A
MODULE NAME?
```

Операции выполняются в следующем порядке:

— удаляются модули XYZ.OBJ и A.OBJ из библиотечного файла FILE.OBJ;
— заменяются модули из библиотечного файла FILE.OBJ модулями из файла MODY.OBJ;
— включаются модули файла MODX.OBJ в библиотечный файл FILE.OBJ;
— выводится листинг каталога библиотечного файла FILE.OBJ на построчно-печатающее устройство.

3.18. MACRO (/M[:N]).

Переключатель /M:N создает файл макробиблиотеки из входного файла, который содержит макроопределения. Аргу-

мент N — восьмеричное число, которое определяет количество записей имен макрокоманд в каталоге макробиблиотеки.

Чтобы задать десятичное число, надо за N поставить точку (N.). Каждые 64 (десятичное) имени макрокоманд занимают 1 блок в каталоге макробиблиотеки. По умолчанию N равно 128 (десятичное), этого достаточно для записи 128 имен макрокоманд, которые будут занимать 2 блока в каталоге макробиблиотеки:

Пример:

*SYSMAC.SML=SYSMAC/M

Создается макробиблиотека SYSMAC.SML из входного файла SYSMAC.MAC.

4. СООБЩЕНИЯ ОПЕРАТОРУ

Ниже приведены сообщения, выдаваемые программой LIBR.

?LIBR—F—EOF DURING EXTRACT

Причина. Конец входного файла был обнаружен раньше конца выбираемого модуля.

Действие. Вновь создать библиотечный файл, повторить операцию.

?LIBR—F—FILE NOT FOUND DEV:FILNAM.TYP

Причина. Один из входных файлов, указанных в командной строке, не найден.

Действие. Исправить и заново ввести командную строку.

?LIBR—F—INPUT ERROR IN DEV:FILNAM.TYP

Причина. Ошибка при чтении входного файла.

Действие. Проверить готовность и исправность оборудования. Повторить операцию.

?LIBR—F—INSUFFICIENT MEMORY

Причина. Недостаточно оперативной памяти для выполнения операции.

Действие. Освободить часть оперативной памяти (удалить ненужные драйверы, удалить основное задание, использовать монитор одного задания), повторить операцию.

?LIBR—F—INTERNAL ERROR

Причина. Сбой в работе операционной системы. Возможно, запорчен текст LIBR.

Действие. Повторить операцию. При появлении этой же ошибки получить новую копию программы LIBR.

?LIBR—F—INVALID DEVICE DEV:

Причина. В командной строке указано недопустимое устройство.

Действие. Проверить и заново ввести командную строку.

?LIBR—F—INVALID GSD IN DEV:FILNAM.TYP

Причина. Ошибка в каталоге глобальных имен (GSD). Файл является неправильным объектным модулем.

Действие. Вновь протранслировать исходную программу, для получения правильного объектного модуля и повторить операцию.

?LIBR—F—INVALID INPUT FILE DEV:FILNAM.TYP

Причина. Входной файл не является библиотечным.

Действие. Указать правильное имя файла и повторить командную строку.

?LIBR-F-INVALID LIBRARY FOR LISTING OR EXTRACT

Причина. Входной файл, предназначенный для операции выборки или получения листинга каталога, не является библиотечным.

Действие. Проверить и заново ввести командную строку.

?LIBR—F—INVALID OPTION:/Y

Причина. Данный переключатель («Y») не является переключателем библиотекаря.

Действие. Исправить и заново ввести командную строку.

?LIBR—F—INVALID OPTION COMBINATION

Причина. Указаны переключатели, которые выполняют несовместимые операции. Например, если указан /E, нельзя использовать никакой другой переключатель; если указан /M, то за ним может следовать только переключатель продолжения (/C или //).

Действие. Исключить переключатели, вызывающие ошибку, и повторить операцию.

?LIBR—F—INVALID RECORD TYPE IN DEV:FILNAM.TYP

Причина. Тип кода двоичной записи объектного файла не в пределах от 1 до 10 (восьмеричное).

Действие. Вновь протранслировать исходную программу для получения правильного объектного файла, повторить операцию.

?LIBR—F—MACRO NAME TABLE FULL, UZE/M:N

Причина. Переполнение каталога имен макрокоманд в макробиблиотеке.

Действие. Увеличить размер каталога имен макрокоманд с помощью переключателя /M:N.

?LIBR—F—NO VALUE ALLOWED :/N

Причина. В командной строке за переключателем следует аргумент, что недопустимо.

Действие. Исправить и заново ввести командную строку.

?LIBR—F—OUTPUT AND INPUT FILNAMES THE SAME

Причина. Входной и выходной файлы, предназначенные для создания макробиблиотеки, имеют одинаковые спецификации в командной строке.

Действие. Переименовать входной или выходной файлы.

?LIBR—F—OUTPUT DEVICE FULL DEV:FILNAM.TYP

Причина. На указанном устройстве недостаточно места для библиотечного файла или файла листинга каталога библиотечного файла.

Действие. Сжать том по команде SQUEEZE. Удалить или записать на другой том ненужные файлы. Использовать другой том.

?LIBR—F—OUTPUT ERROR DEV:FILNAM.TYP

Причина. Ошибка при записи выходного файла или заблокирована запись на устройстве вывода.

Действие. Проверить готовность и исправность оборудования. Повторить операцию.

?LIBR—F—OUTPUT FILE FULL

Причина. Выходной файл мал для библиотечного файла или файла листинга.

Действие. Увеличить размер выходного файла по команде LIBR/ALLOCATE:N или с помощью конструкции [:N] *в спецификации выходного файла* (см. [1]).

?LIBR-F-PROTECTED FILE ALREADY EXISTS DEV:FILNAM.TYP

Причина. Попытка включить в защищенный библиотечный файл новый модуль или создать библиотечный файл, когда существует защищенный файл с тем же именем.

Действие. Отменить защиту файла по команде RENAME/NO-PROTECT или PIP/Z или использовать другое имя для нового библиотечного файла.

?LIBR-F-/R OR /U GIVEN ON LIBRARY FILE DEV:FILNAM.TYP

Причина. В командной строке за спецификацией библиотечного файла следует переключатель /R или /U, что недопустимо.

Действие. Исправить и заново ввести командную строку.

?LIBR—F—/U GIVEN ON LIBRARY FILE DEV:FILNAM.TYP

Причина. Переключатель /U указан за спецификацией библиотечного файла, что недопустимо. Переключатель

/U можно указывать только за спецификацией входного файла.

Действие. Исправить командную строку и повторить операцию.

?LIBR—W—DUPLICATE FORM NAME OF FORMNM

Причина. Во входной спецификации даны два файла с одинаковыми именами и после второго имени не указан переключатель /U или /UPDATE. Файл, имя которого встречено первым, включается в библиотечный файл. Все дубликаты имен игнорируются.

Действие. Использовать в командной строке переключатель /U или /UPDATE.

?LIBR—W—DUPLICATE MACRO NAME OF MACNAM

Причина. Во входном файле имеются две макрокоманды с одним и тем же именем. Макрокоманда, имя которой встречено первым, заносится в выходной файл. Все дубликаты имен игнорируются.

Действие. Присвоить макрокомандам разные имена.

?LIBR—W—DUPLICATE MODULE NAME OF AAAAAA

Причина. В библиотечный файл был включен новый модуль, но он имеет то же имя, что и модуль, уже существующий в библиотечном файле. Программа библиотекарь вводит повторно имя этого модуля в каталог.

Действие. Не требуется.

?LIBR—W—INVALID CHARACTER

Причина. Введенное имя содержит символ, не относящийся к RADIX-50.

Действие. Исправить командную строку, повторить операцию.

?LIBR—W—INVALID DELETE OF AAAAAA

Причина. Удаляемый модуль AAAAAA в каталоге библиотечного файла не существует.

Действие. Не требуется.

?LIBR—W—INVALID EXTRACT OF AAAAAA

Причина. Указанное глобальное имя не найдено в каталоге библиотечного файла.

Действие. Исправить командную строку, повторить операцию.

?LIBR—W—INVALID INSERT AAAAAA

Причина. Включаемый в библиотечный файл модуль AAAAAA содержит ту же точку входа, что и модуль, уже существующий в библиотечном файле. Точка входа

игнорируется, но модуль включается в библиотечный файл.

Действие. Не требуется.

?LIBR—W—INVALID—REPLACEMENT OF AAAAAA

Причина. Заменяемый в библиотечном файле модуль AAAAAA не существует. Модуль игнорируется.

Действие. Не требуется.

?LIBR—W—NULL LIBRARY

Причина. Создаваемый библиотечный файл не содержит глобальных имен.

Действие. Входной файл должен иметь по крайней мере одно глобальное имя.

?LIBR—W—ONLY CONTINUATION ALLOWED

Причина. Введена командная строка без переключателя продолжения после строки, использующей переключатель продолжения.

Действие. Подать правильную команду.

ПЕРЕЧЕНЬ ССЫЛОЧНЫХ ДОКУМЕНТОВ

1. Операционная система ФОДОС-2. Командный язык системы

ОТЛАДЧИК И ВИРТУАЛЬНЫЙ ОТЛАДЧИК

РУКОВОДСТВО ОПЕРАТОРА

1. НАЗНАЧЕНИЕ ПРОГРАММЫ И УСЛОВИЯ ВЫПОЛНЕНИЯ ПРОГРАММЫ

Программа отладчик предназначена для отладки программ пользователя путем прогона их определенными участками и проверки на ожидаемые результаты в различных точках.

Отладка программ пользователя с помощью отладчика осуществляется в режиме диалога пользователя с ЭВМ.

Отладчик позволяет осуществлять:

1) просмотр содержимого любой ячейки памяти и внесение необходимых изменений;

2) прогон всей отлаживаемой программы или любой ее части;

3) поиск слов, байтов с определенной битовой комбинацией;

4) поиск исполнительных адресов;

5) вычисление смещений относительных адресов;

6) заполнение указанного блока памяти заданными словами или байтами;

7) преобразование кодов в символы КОИ—7 или в символы RADIX-50;

8) распечатку указанного блока памяти.

Во время отладки необходимо иметь листинг отлаживаемой программы. Программу отладчик в виде перемещаемого объектного модуля ODT.OBJ рекомендуется хранить на системном диске.

Для отладки программы необходимо связать программу отладчик и отлаживаемую программу с помощью редактора связей.

2. ОБЩИЕ ПОНЯТИЯ И ОБОЗНАЧЕНИЯ

2.1. Точки разрыва — это определенные пользователем ячейки, в которых выполнение отлаживаемой программы должно временно приостанавливаться. Они используются для облегчения отладки программ пользователя. Одновременно пользователь может установить до 8 точек разрыва с нумерацией от 0 до 7.

С помощью команд 'R;G' и 'K;P' (см. п. 4.4.4) программа отладчик устанавливает в точках разрыва команду BPT (000003), запоминая первоначальное содержимое этих ячеек, и передает управление программе пользователя. Программа пользователя выполняется до тех пор, пока не встретит точку разрыва (команда BPT).

По команде BPT выполнение программы пользователя приостанавливается, и управление передается программе отладчик, которая восстанавливает первоначальное содержимое ячеек точек разрыва и ожидает от пользователя очередной команды. При использовании точек разрыва пользователю необходимо учитывать следующие ограничения:

1) отлаживаемая программа не должна обращаться к слову, где была установлена точка разрыва;

2) не следует устанавливать точку разрыва в ячейке, которая очищает Т-разряд.

2.2. Регистры перемещения. В процессе трансляции исходных модулей АССЕМБЛЕР создает перемещаемые объектные модули с базовым адресом, равным нулю. Адреса всех ячеек программы рассматриваются относительно этого базового адреса.

Величиной перемещения модуля называется разность между адресом загрузки модуля и адресом, определяемым при его трансляции. Скомпонованная программа может содержать несколько модулей, каждый из которых имеет свою величину перемещения.

Программа отладчик содержит 8 специальных ячеек (регистров перемещения) для записи величин перемещения модулей. Для того, чтобы адреса в программе соответствовали адресам листинга, пользователь заносит в регистры перемещения значения, соответствующие величинам перемещения объектных модулей.

2.3. Принятые обозначения. Команды оператора подаются с терминала в виде последовательности символов (табл. 4). Обозначения общих элементов команд приведены в табл. 1.

Таблица 1

ОБОЗНАЧЕНИЯ ОБЩИХ ЭЛЕМЕНТОВ КОМАНД

Обозначение	Наименование
1	2
R	Адрес ячейки, вычисляемый программой отладчик как 16-разрядное значение
N	Целое число от 0 до 7
K	Восьмеричное число от 0 до 177777

Если оператор вводит число, имеющее более 6 цифр или имеющее значение более 177777, отладчик учитывает только последние 6 цифр, содержащиеся в 16 младших разрядах.

Если оператор указывает отрицательное число, программа отладчик представляет это число как дополнение до двух.

Примеры представления чисел:

1	000001
—1	177777
400	000400
—177730	000050
1234567	034567

2.4. Форматы печати. Отладчик может выдавать адреса в абсолютном или относительном формате. Форматы печати адресных выражений приведены в табл. 2.

Таблица 2

ФОРМАТЫ ПЕЧАТИ АДРЕСНЫХ ВЫРАЖЕНИЙ (R)

Формат печати	Вид выражения R	Значение выражения R
1	2	3
Абсолютный	K	Число K
	C	содержимое регистра константы C
Относительный	N,K	Сумма содержимого регистра перемещения N и числа K
	C,K	Сумма содержимого регистра перемещения, определяемого регистром C и числа K
	N,C	В зависимости от занимаемой позиции C определяет номер регистра перемещения или значение, заданное в регистре C
	C,C	

Примеры адресных выражений приведены в табл. 3 для случая: N=3, C=000003. В регистре перемещения содержится 3400.

Таблица 3

Формат печати	Вид выражения R	Значение выражения R
1	2	3
Абсолютный	5	000005
»	—17	177761
Относительный	3,0	003400
»	3,150	003550
»	3, —1	003377
Абсолютный	C	000003
Относительный	C,0	003400
»	C,10	003410
»	3,C	003403
»	C,C	003403

Обычно отладчик выдает адреса в относительном формате (N,K). При этом отладчик просматривает все регистры перемещения и отыскивает тот, значение которого близко, но не превосходит адреса ячейки, которую необходимо проверить. Затем отладчик выдает адрес относительно содержимого этого регистра перемещения. Если соответствующего регистра перемещения нет, адрес выдается в абсолютном формате. Поскольку при инициализации в регистрах перемещения устанавливаются значения —1, первоначально адреса выдаются в абсолютном формате.

Пример.

```
*1000;1R                    ; регистр перемещения 1 содер-
                            ; жит 1000
*1,4  ;2R                   ; регистр перемещения 2 содер-
                            ; жит 1004
*774/000000        <ПС>     ; абсолютный адрес 774
000776/012345      <ПС>     ; абсолютный адрес 776
1,000000/000000    <ПС>     ; абсолютный адрес 1000
1,000002/000356    <ПС>     ; абсолютный адрес 1002
2,000000/112713    <ПС>     ; абсолютный адрес 1004
```

Форматы печати адресов контролируются регистром формата '¤F/'. Обычно он содержит 0, в этом случае отладчик выдает адреса, где это возможно, в относительном формате. Пользователь может открыть '¤F' и изменить его содержи-

мое на ненулевое. В этом случае все адреса выдаются в абсолютном формате.

3. ВЫПОЛНЕНИЕ ПРОГРАММЫ

Перед началом работы следует связать отладчик и программу пользователя в единый загрузочный модуль с помощью редактора связей.

Чтобы в результате выполнения отлаживаемой программы не произошло наложение на отладчик, его желательно расположить в памяти первым, т. е. в младших адресах. Для этого в отлаживаемой программе следует использовать именованные П-секции (директива .PSECT). Поскольку редактор связей располагает неименованные П-секции в памяти ниже, чем именованные, отладчик будет расположен в памяти ниже, чем отлаживаемая программа.

Пример.

Если в программе MYPROG имеется директива .PSECT MYPROG, то в результате выполнения следующей команды редактор связей создает загрузочный модуль MYPROG.SAV, в котором ODT располагается ниже, чем MYPROG:

.LINK/MAP:TT/DEBUG <BK>

Если пользователь помещает отладчик в памяти выше по отношению к программе, то буфер программы пользователя следует располагать в пределах программы.

Если программа отладчик объединена с программой оверлейной структуры, то отладчик должен быть расположен в корневом сегменте.

Возможные варианты объединения отлаживаемой программы MYPROG и программы отладчик показаны на рис. 1.

Адрес точки входа программы отладчик (O.ODT) может быть определен из карты загрузки, выдаваемой редактором связей.

Вызов программы отладчик осуществляется по командам монитора R, GET, START. Отладчик выдает на терминал '*' и ждет дальнейших команд (см. табл. 4). Точками входа программы отладчик являются точки O.ODT, O.ODT+2 и O.ODT+4.

При повторном пуске с адреса O.ODT сохраняется содержимое регистров перемещения и регистров общего назначения. Точки разрыва в отлаживаемой программе удаляются.

При перезапуске с адреса O.ODT+2 регистры перемещения инициализируются (принимают значение —1), содержи-

мое регистров общего назначения сохраняется. Точки разрыва в отлаживаемой программе удаляются.

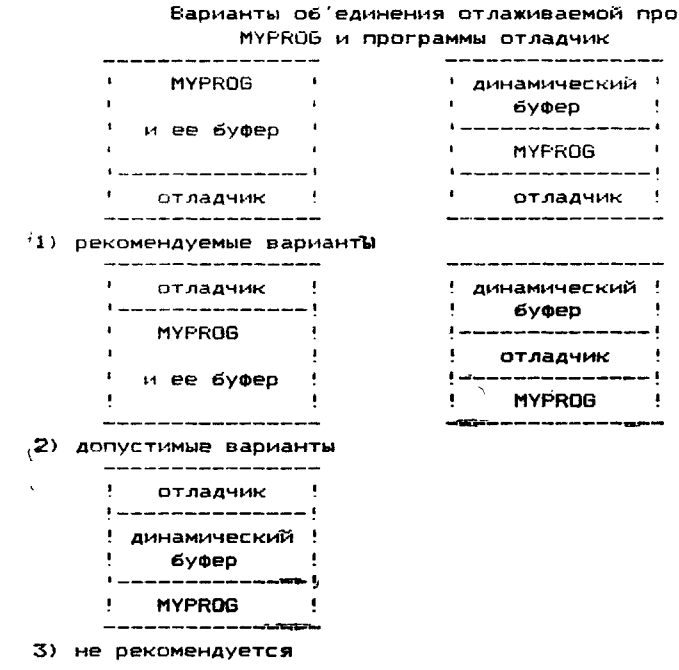


Рис. 1

При перезапуске с адреса O.ODT+4 сохраняется содержимое регистров перемещения и регистров общего назначения. Точки разрыва сохраняются. Отладчик выводит на терминал сообщение о плохом входе (BE). Точки разрыва, установленные до перезапуска, сбрасываются по команде ;G, команда ;P недопустима после сообщения BE (см. п. 4.4.3).

Программу отладчик можно перезапустить по команде монитора REENTER, если отлаживаемая программа устанавливает разряд перезапуска в слове состоянии задания и располагается в памяти ниже программы отладчик.

Для передачи управления монитору следует подать команду СУ/С. Монитор выдает на терминал ΛС.

ПРИМЕЧАНИЕ. Для отладки программ под управлением монитора основного — фонового задания отлаживаемую программу рекомендуется помещать в основную область памяти, отладчик — в фоновую область памяти.

Примеры:

1. С помощью переключателя DEBUG редактор связей (LINK) связывает программу отладчик и отлаживаемую программу MYPROG. Первой в памяти располагается программа отладчик. Запуск MYPROG (по команде R) вызывает автоматическое выполнение программы отладчик.

```
.LINK/MAP:TT:/DEBUG MYPROG
FODOS LINK B03.00      LOAD MAP  THURSDAY 10-JAN-85 12:45
MYPROG.SAV        TITLE:  ODT      IDENT:   B03.00
SECTION ADDR   SIZE  GLOBAL VALUE GLOBAL VALUE GLOBAL VALUE
. ABS.  000000 001000      (RW, I, GBL, ABS, OVR)
$ODT$    001000  006152     (RW, I, LCL, REL, CON)
                             O.ODT     001232
PROG     007152  002052     (RW, I, LCL, REL, CON)
                            START     007152
TRANSFER ADDRESS = 001232, HIGH LIMIT = 011222 = 2377. WORDS
.
.R MYPROG
 ODT B03.00
*
```

2. Отлаживаемая программа MYPROG связана с программой отладчик. С помощью переключателя TRANSFER пользователь задает точку входа O.ODT. Первой в памяти расположена программа MYPROG. При вызове MYPROG отладчик вызывается автоматически.

```
.LINK/MAP:TT: MYPROG,ODT/TRANSFER
TRANSFER SYMBOL? O.ODT
FODOS LINK B03.00      LOAD MAP  THURSDAY 10-JAN-85 12:45
MYPROG.SAV      TITLE:    MYPROG    IDENT:   B03.00
SECTION ADDR   SIZE  GLOBAL VALUE GLOBAL VALUE GLOBAL VALUE
.ABS.   000000 001000     (RW, I, GBL, ABS, OVR)
PROG     001000  002052   (RW, I, LCL, REL, CON)
                            START     001000
$ODT$    003052  006152    (RW, I, LCL, REL, CON)
                            O.ODT     003304
TRANSFER ADDRESS = 003304, HIGH LIMIT = 011222 = 2377. WORDS
.
.R MYPROG
 ODT B03.00
*
```

3. Этот пример похож на предыдущий, за исключением того, что загрузка программы MYPROG осуществляется по команде монитора GET, а пуск — по команде монитора START с адреса 3304.

```
.LINK/MAP:TT: MYPROG,ODT
FODOS LINK 803.00        LOAD MAP   THURSDAY 10-JAN-85 12:45
MYPROG.SAV      TITLE:     MYPROG     IDENT:    803.00
SECTION ADDR   SIZE  GLOBAL VALUE GLOBAL VALUE GLOBAL VALUE
.ABS.    000000 001000    (RW, I, GBL, ABS, OVR)
%PROG     001000  002052   (RW, I, LCL, REL, CON)
                            START      001000



%ODT%     003052  006152   (RW, I, LCL, REL, CON)
                           O.ODT      003304
TRANSFER ADDRES = 003304, HIGH LIMIT = 011222 = 2377. WORDS
.GET MYPROG
.START 3304
 ODT 803.00
*
```

4. С помощью переключателя BOTTOM связаны программа MYPROG и программа отладчик. По команде GET последовательно осуществляется загрузка обеих программ, пуск программы отладчик осуществляется по команде START с адреса 4232.

```
.LINK/MAP:TT: ODT/BOTTOM:4000
FODOS LINK 803.00        LOAD MAP   THURSDAY 10-JAN-85 12:45
ODT.SAV         TITLE:   ODT      IDENT:    803.00     /B:004000
SECTION ADDR   SIZE  GLOBAL VALUE GLOBAL VALUE GLOBAL VALUE
.ABS.     000000  004000     (RW, I, GBL, ABS, OVR)
%ODT%     004000  006152     (RW, I, LCL, REL, CON)
                             O.ODT      004232
TRANSFER ADDRES = 004232, HIGH LIMIT = 012150 = 2612. WORDS
.GET ODT.SAV
.GET MYPROG.SAV
.START 004232
 ODT 803.00
*
```

По команде START O.ODT+2 или по команде START O.ODT+4 можно осуществить повторный пуск программы отладчик, например:

```
.START 4234
*
```

или

```
.START 4236
BE004242
*
```

4. КОМАНДЫ ОПЕРАТОРА

В табл. 4 приведены команды программы отладчик.

Таблица 4

КОМАНДЫ ОПЕРАТОРА

Команда	Наименование
1	2
R/ или / R\ или \	Открыть слово Открыть байт
<ПС> ^	Открыть следующую ячейку Открыть предыдущую ячейку
<ВК> — (подчеркивание)	Закрыть ячейку Открыть ячейку по относительному адресу
@	Открыть ячейку по абсолютному адресу
>	Открыть ячейку, определяемую смещением команды ветвления
<	Вернуться к прерванной последовательности команд
¤N/	Открыть регистр общего назначения N
¤C/	Открыть регистр состояния программы
¤X/	Открыть внутренний регистр (X — имя регистра)
R;B	Задать точку разрыва по адресу R
R;NB	Задать точку разрыва по адресу R с номером N
;B	Удалить все точки разрыва
;NB	Удалить точку разрыва с номером N
R;G	Пустить программу с адреса R
R;P	Продолжить выполнение программы с точки разрыва
K;P	Продолжить выполнение программы, игнорируя точку разрыва (K—1) раз
R;O	Вычислить смещение от текущей открытой ячейки до ячейки с адресом R
R;E	Найти исполнительный адрес R
R;W	Найти слово
;KS	Установить режим одиночных команд
;S	Выйти из режима одиночных команд
K;NR	Записать K в регистр перемещения R
;NR	Инициализировать регистр перемещения N
;R	Инициализировать все регистры перемещения
N!	Вычислить смещение программы относительно значения регистра перемещения N
!	Вычислить смещение программы относительно ближайшего значения регистра перемещения
N№R	Вычислить смещение программы, используя содержимое открытой ячейки
R;KA	Ввод и вывод в символах КОИ-7
X	Ввод и вывод символах RADIX-50

1	2
K;C	Записать K в регистр константы
;F	Заполнить блок памяти содержимым регистра константы
;I	Заполнить блок памяти младшим байтом регистра константы

ПРИМЕЧАНИЯ:

1. Код команды '^' — 136.

2. Ввод недопустимой команды или символа (например, 9 или <ЗБ>) вызывает сообщение об ошибке (см. раздел 5). Это свойство можно использовать для отмены уже напечатанной команды.

4.1. Команды открытия и закрытия ячеек.

Открытой ячейкой является ячейка, содержимое которой выведено на терминал для проверки или изменения.

Закрытой ячейкой является ячейка, содержимое которой недоступно для изменения.

Содержимое открытой ячейки может быть изменено путем указания нового значения и ввода одной из команд: <ВК>, <ПС>, ^, _, @, > или <. Любая команда (кроме, /, \), введенная для открытия ячейки, когда другая ячейка уже открыта, закрывает текущую открытую ячейку.

Если подается команда открытия ячейки по четному адресу, то автоматически устанавливается режим слова, и все последующие команды будут оперировать со словами.

4.1.1. Открыть слово.

Формат команды: R/

Команда 'R/' используется для указания содержимого ячейки с адресом R в виде шестизначного восьмеричного числа, а также внесения изменений.

Для выполнения команды необходимо после '*' набрать с клавиатуры терминала 'R/'. Для внесения изменений следует указать новое содержимое, прежде чем закрыть ячейку.

Пример.

```
*1000;1R                    ;в регистр перемещения запи-
                            ;сывается 1000
*1,0/012746 012534 <ВК>     ;изменяется содержимое ячей-
                            ;ки 1000
*/012534                    ;вновь открывается ячейка
                            ;001000
```

4.1.2. Открыть байт.

Формат команды: R\

Команда 'R\' используется для проверки и изменения содержимого байта.

Для выполнения команды необходимо после '*' набрать с клавиатуры терминала 'R\'. Адрес для открытия ячейки байта может быть указан как четный, так и нечетный.

При введении этой команды на терминал выводится величина байта и его символьное представление, соответствующее коду КОИ—7, если таковое существует, в противном случае, вопросительный знак.

По команде '\' открывается байт, который был открыт последним.

Пример.

```
*1000;1R                ;в регистр перемещения записывается
                        ;1000
*1,1\024=? <ВК>         ;открывается байт 1001
*\024=?    <ПС>         ;вновь открывается байт 1001
001002\041=!            ;открывается байт 1002
```

4.1.3. Закрыть ячейку.

Формат команды: <ВК>

Если нет необходимости изменять содержимое открытой ячейки, оператор подает команду <ВК>. Открытая ячейка закрывается, отладчик выводит на терминал '*' и ждет следующую команду.

Если содержимое открытой ячейки необходимо изменить, оператор указывает новое значение и команду <ВК>.

Пример.

```
*001000/012345 <ВК>  ;ячейка 001000 закрывается без из-
                     ;менения
```

4.1.4. Открыть следующую ячейку.

Формат команды: <ПС>

По команде <ПС> закрывается открытая ячейка и открывается следующая.

Пример.

```
*001020/004532 5120 <ПС>  ;изменяется содержимое ячей-
                          ;ки 001020
001022/013262             ;открывается ячейка 001022
```

Если открыта ячейка байта, то по команде <ПС> открывается следующий байт, например:

```
*1,1\024=? <ПС>
1,2\041=!
```

4.1.5. Открыть предыдущую ячейку

Формат команды: ∧

По команде '∧' закрывается текущая ячейка и открывается предыдущая (слово или байт).

Пример.

```
*001002/030441 ∧
001000/012345                    ; открывается ячейка 1000
```

4.1.6. Открыть ячейку по относительному адресу

Формат команды: —

По команде '—' содержимое открытой ячейки интерпретируется как индексное слово относительного метода адресации и открывается ячейка по относительному адресу.

Пример.

```
*001010/000352 770_              ; индексное слово 770
002002/001227                    ; открывается ячейка 002002
```

Если открытая ячейка содержит нечетное значение, то по команде '—' открывается ячейка байта.

4.1.7. Открыть ячейку по абсолютному адресу

Формат команды: @

По команде '@' открытая ячейка закрывается, и ее содержимое используется как адрес, по которому открывается ячейка.

Пример.

```
*001006/1044 2100 @
002100/000167          ; открывается ячейка по адресу 2100
```

4.1.8. Открыть ячейку, определяемую смещением команды ветвления.

Формат команды: >

По команде '>' младший байт открытой ячейки интерпретируется как смещение команды ветвления, и открывается ячейка, определяемая этим смещением.

Пример.

```
*1032/000407 301>
000636/104400          ; открывается ячейка 000636
                       (636 = —77*2 + 1032 +2)
```

Значение К в последнем примере отрицательно:

К = 301 = —77

4.1.9. Вернуться к прерванной последовательности команд

Формат команды: <

Если последовательность выполнения команд была прервана одной из команд: '>', '—', '@' — и необходимо к ней вернуться, то используется команда '<'.

Если до ввода команды '<' не была подана ни одна из команд '>', '—' или '@', то действие команды '<' аналогично команде <ПС>.

Пример.

```
*1062/001402 @          ; вызывает изменение последовательно-
                        ; сти
001402/000600 <         ; возвращает к первоначальной после-
                        ; довательности
001064/104002 <ВК>
*276/012767 336_        ; вызывает изменение последовательно-
                        ; сти
000636/104400 <         ; возвращает к первоначальной после-
                        ; довательности
300/000240 <            ; действует как <ПС>
302/001000 <ВК>
```

4.2. Обращение к регистрам программы пользователя

Программа отладчик имеет специальные ячейки, которые используются для запоминания текущих значений регистров общего назначения и слова состояния программы пользователя во время останова.

4.2.1. Открыть регистр общего назначения N

Формат команды: ¤N/
где N— номер регистра (0—7).

По команде '¤N/' открывается регистр общего назначения программы пользователя. Оператор может изменить содержимое открытой ячейки, указав новое значение перед вводом команды <ВК>.

При обращении к регистрам общего назначения можно использовать команды <ПС>, '^', '—', '@'.

Пример.

```
*¤2/000050 464<ВК>  ; изменяется содержимое регистра R2
*/000464            ; вновь открывается R2
```

4.3. Обращение к специальным внутренним регистрам

Программа отладчик содержит специальные ячейки, которые необходимы для отладки программ пользователя. Это внутренние регистры. Внутренние регистры доступны пользователю так же, как и любые ячейки памяти. Обращение к этим регистрам осуществляется через их имена (табл. 5).

Таблица 5

Регистр	Наименование	Функция
1	2	3
¤B	Регистр нулевой точки разрыва	Содержит первое слово таблицы точек разрыва

1	2	3
¤C	Регистр константы	Сохраняет преобразованное значение
¤F	Регистр формата	Указывает формат печати адресов
¤M	Регистр маски	Указывает, какие разряды будут просмотрены во время поиска указанной комбинации разрядов
¤P	Регистр приоритета	Определяет рабочий приоритет программы отладчик
¤R	Нулевой регистр перемещения	Содержит базовый адрес таблицы перемещения
¤S	Регистр состояния программы	Содержит коды условий ветвления (разряды с 0 по 3) и уровень приоритета прерывания (разряды с 5 по 7)

Команда вида '¤X/', где X — имя регистра, используется для открытия соответствующего внутреннего регистра. Команды '¤B/', '¤C/' и '¤M/' описаны в пп. 4.4.3, 4.5.4 и 4.5.1 соответственно.

4.3.1. Открыть регистр формата

Формат команды: ¤F/

По команде '¤F/' регистр формата открывается для проверки или изменения.

Регистр формата предназначен для задания формата печати адресов и устанавливается пользователем.

Пример.

```
*¤F/000000 3<ВК>     ;изменяется содержимое регистра
*                    ;формата
```

Обычно регистр формата содержит ноль. В этом случае отладчик печатает адреса в относительном формате (N;K), где это возможно. Если содержимое регистра формата изменить на ненулевое значение, то адреса будут выводиться на терминал в абсолютном формате.

4.3.2. Открыть регистр приоритета

Формат команды: ¤P/

По команде '¤P/' регистр приоритета открывается для проверки и изменения его содержимого.

Регистр приоритета — это ячейка, содержащая рабочий приоритет программы отладчик.

Пример.

```
*¤P/000006 4<ВК> ;наименьший приоритет для разреше-
                 ;ния прерывания с терминала
```

Регистр приоритета может содержать значения от 0 до 7 в соответствии с заданным приоритетом, с которым работает программа отладчик. Если регистр приоритета содержит значение 377, программа отладчик работает на уровне приоритета процессора. Первоначально регистр приоритета содержит 7.

Если программа отладчик работает под управлением монитора основного — фонового задания, то в регистре приоритета должен быть ноль.

4.3.3. Открыть регистр состояния программы

Формат команды: ¤S/

По команде '¤S/' открывается регистр состояния программы (РСП).

Регистр состояния программы пользователя — это ячейка, содержащая коды условий ветвления и уровень приоритета прерывания программы.

Пример.

```
*¤S/000311 <ВК>     ;открывается и закрывается РСП
```

В ответ на команду '¤S/' отладчик выдает на терминал 16-разрядное слово, из которого только 8 младших разрядов имеют смысл. Разряды с 0 по 3 коды условий ветвления Z, N, C и V, указывающие на результат последней операции центрального процессора. Эти разряды устанавливаются следующим образом:

1) Z=1, если результат равен 0;

2) N=1, если результат отрицательный;

3) C=1, если в результате выполнения операции произошел перенос из самого старшего разряда, или, если при сдвиге вправо или влево из самого младшего или самого старшего разряда была выдвинута единица;

4) V=1, если в результате выполнения команды произошло арифметическое переполнение.

Разряды с 5 по 7 содержат уровень приоритета прерывания (в диапазоне от 0 до 7) программы.

4.3.4. Открыть нулевой регистр перемещения

Формат команды: ¤R/

По команде '¤R/' нулевой регистр перемещения открывается для проверки и изменения. Последовательно подавая команду <ПС>, можно открыть остальные 7 регистров перемещения.

Пример.

```
*¤R/177777 <ПС>  ;открывается нулевой регистр переме-
                 ;щения
```

```
NNNNNN/001000      ;открывается первый регистр переме-
                   ;щения
```

NNNNNN — внутренний адрес программы отладчик.

Первоначально регистры перемещения содержат —1.

4.3.5. Записать K в регистр перемещения N

Формат команды: K;NR

Команда 'K;NR' (K — величина перемещения модуля, N — номер регистра от 0 до 7) используется для внесения величины перемещения объектного модуля в регистр перемещения N.

Пример.

```
*2000;2R           ;в регистр перемещения 2 вносится 2000
*2,100;2R          ;содержимое регистра перемещения 2
                   ;увеличивается на 100
```

При записи в нулевой регистр перемещения номер регистра можно не указывать.

Если программа пользователя связана с адреса, который меньше указанного в листинге, то в регистр перемещения следует внести отрицательное число. Например, программа пользователя связана с адреса 001000, адрес программы в листинге — 005000. В регистр перемещения пользователем вносится число —4000:

```
*—4000;1R
```

По команде ';NR' в регистр перемещения N записывается —1.

Пример.

```
*;1R               ;регистр перемещения 1 инициали-
                   ;зируется
*¤1R/177777 <ПС>
NNNNNN/177777 <ВК>
*
```

В данном примере NNNNNN — внутренний адрес отладчика.

По команде ';R' все регистры перемещения (0—7) инициализируются.

4.3.6. Вычислить смещение относительно значения N-го регистра перемещения

Формат команды: N!

Команда 'N!' используется для определения смещения абсолютного адреса открытой ячейки относительно N-го регистра перемещения (N=0, 1...7).

Пример.

```
*2500;2R
*3100/012767 2! = 2,000400     ;ячейка 000400 в листин-
                               ;ге соответствует ячейке
                               ;003100 в памяти
```

Команда '!' выполняется аналогично команде 'N!', однако, отладчик выбирает тот регистр перемещения, значение которого наиболее близко адресу открытой ячейки, но не превосходит его.

4.3.7. Вычислить смещение, используя содержимое открытой ячейки

Формат команды: NR

По команде 'NR' на терминал выводится номер регистра перемещения и восьмеричное число, равное разности содержимого открытой ячейки и регистра перемещения N.

Пример.

```
*1000;1R
*6000/007600 1R = 1,006600        ;отладчик выдает номер
                                  ;регистра перемещения и
                                  ;вычисленное смещение
```

По команде 'R' отладчик выбирает регистр перемещения, содержимое которого наиболее близко, но не превосходит содержимого открытой ячейки.

Пример.

```
*3000;2R
*4000;3R
*6000/007600 2R = 2,004600 3R = 3,003600 R = 3,003600
```

В данном примере по команде 'R' был выбран третий регистр перемещения.

4.4. Команды управления

Отладчик осуществляет управление программой пользователя с помощью команд:

1) задания точек разрыва;

2) запуска программы и продолжения выполнения программы с точки разрыва.

Программа пользователя может выполняться в одном из режимов: нормальном или режиме одиночных команд.

Нормальный режим устанавливается после пуска программы отладчик.

Режим одиночных команд вводится по команде ';KS'.

Точки разрыва задаются пользователем. В качестве точек разрыва нельзя использовать ячейки, в которых находятся данные. Ячейки, содержащие команды прерывания IOT, EMT, TRAP, могут использоваться в качестве точек разрыва лишь в том случае, если выходы из подпрограмм обработки этих прерываний осуществляются по команде RTI. В противном случае управление программе отладчик возвращено не будет.

4.4.1. Установить точку разрыва с номером N по адресу R

Формат команды: R;NB

По команде 'R;NB' пользователь устанавливает точку разрыва с номером N по адресу R.

Пример.

```
*1030;1B          ;задается точка разрыва с номером 1
                  ;по адресу 001030
*
```

Если точки разрыва задаются последовательно, то номера точек разрыва можно не указывать.

Пример.

```
*1020;B           ;задается точка разрыва с номером 0 по
                  ;адресу 1200
*1030;B           ;задается точка разрыва с номером 1 по
                  ;адресу 1030
*1040;B           ;задается точка разрыва с номером 2 по
                  ;адресу 1040
*1034;4B          ;задается точка разрыва с номером 4 по
                  ;адресу 1034
*1120;7B          ;задается точка разрыва с номером 7 по
                  ;адресу 1120
*
```

другие точки разрыва не установлены.

4.4.2. Удалить точку разрыва с номером N

Формат команды: ;NB

По команде ';NB' удаляется точка разрыва с номером N.

Пример.

```
*;2B              ; удаляется точка разрыва с номером 2
*
```

По команде ';B' удаляются все точки разрыва в программе пользователя.

4.4.3. Открыть регистр нулевой точки разрыва

Формат команды: ¤B/

По команде '¤B/' открывается ячейка, содержащая адрес точки разрыва с номером 0. Адреса следующих 7 точек разрыва могут быть выведены на терминал последовательно по команде <ПС>.

Адрес точки разрыва можно изменить, указав новое содержимое. Если точка разрыва не задана, то на терминал выводится адрес ячейки программы отладчик.

Пример.

```
*¤B/001020 <ПС>       ;адрес точки разрыва с номе-
                      ;ром 0
NNNNNN/001030 <ПС>    ;адрес точки разрыва с номе-
                      ;ром 1
NNNNNN/001040 <ПС>    ;адрес точки разрыва с номе-
                      ;ром 2
```

```
NNNNNN/007522 <ПС>      ; адрес точки разрыва с номе-
                        ; ром 3 не задан
NNNNNN/001034 <ПС>      ; адрес точки разрыва с номе-
                        ; ром 4
NNNNNN/007522 <ПС>      ; адрес точки разрыва с номе-
                        ; ром 5 не задан
NNNNNN/007522 <ПС>      ; адрес точки разрыва с номе-
                        ; ром 6 не задан
NNNNNN/001120 <ПС>      ; адрес точки разрыва с номе-
                        ; ром 7
```

В данном примере NNNNNN — внутренний адрес отладчика.

Девятая ячейка этой последовательности содержит адрес останова программы пользователя в режиме одиночных команд (см. п. 4.4.6). Последующие 8 ячеек содержат число прохождений через каждую точку разрыва, в девятой по счету ячейке содержится число выполняемых команд в режиме одиночных команд.

4.4.4. Команды запуска и продолжения программы

Формат команд: R;G и R;P

Команда 'R;G' вызывает выполнение программы пользователя, 'R;P' продолжает выполнение после останова в точке разрыва. Программа выполняется до появления программного останова или точки разрыва. Когда встречается точка разрыва, отладчик выдает на терминал:

BN;R

где N — номер точки разрыва, R — адрес точки разрыва.

Пример.

```
*1010;1B        ; задается точка разрыва с номером 1
*1000;G         ; программа пользователя выполняется с адре-
                ; са 001000 до точки разрыва 001010
*
```

Если при выполнении программы пользователя встречается незаданная точка разрыва, то программа отладчик выдает на терминал:

BENNNNNN

где NNNNNN — адрес точки разрыва.

Незаданные точки разрыва появляются при использовании в программе пользователя недопустимой команды BPT или возникновении прерывания по Т-разряду.

По команде ';P' выполнение программы пользователя продолжается с текущей точки разрыва до следующей.

Команда 'K;P' используется, если точка разрыва установлена в цикле. По этой команде отладчик заносит в ячейку

величину К — число прохождений программы через текущую точку разрыва (см. п. 4.4.3) и продолжает выполнение программы пользователя. В результате (К—1), раз точка разрыва игнорируется, К-й раз выполнение программы пользователя прекращается. Программа отладчик получает управление и выводит на терминал:

BN;R

Пример.

```
*1040; 1B
*1060; 2B
*1000;G
  B1;001040
*2;P
  B2;001060
*1000;G
  B2;001060
*1000;G
  B1;001040
*
```

По команде '2;P', поданной после вывода на терминал точки разрыва 001040, отладчик игнорирует эту точку разрыва, встретив ее первый раз. Выполнение программы пользователя прекращается, встретив точку разрыва 001040 второй раз. Отладчик выводит на терминал:

B1;001040

4.4.5. Установить режим одиночных команд

Формат команды: ;KS

где К — любое восьмеричное число.

Режим одиночных команд позволяет выполнить заданное число команд программы пользователя, прежде чем отладчик приостановит выполнение программы. Точки разрыва запрещены в режиме одиночных команд.

Программа пользователя начинает выполняться после установки режима одиночных команд и ввода команды 'R;G' (см. п. 4.4.4).

Если пользователь предварительно не установил ячейку, содержащую число выполняемых команд для режима одиночных команд (см. п. 4.4.3), выполняется одна команда программы пользователя, управление передается программе отладчик и на терминал выводится:

B8;NNNNNN

*

где NNNNNN — адрес первой невыполненной команды про-

граммы пользователя. Этот адрес заносится в ячейку, следующую за точкой разрыва с номером 7.

Чтобы продолжить выполнение программы пользователя, следует подать команду 'K;P', где K определяет число команд, которое необходимо выполнить.

4.4.6. Выйти из режима одиночных команд

Формат команды: ;S

По команде ';S' программа отладчик выходит из режима одиночных команд.

4.5. Команды поиска и записи

Ниже приводятся команды, позволяющие осуществлять поиск слов или адресов в заданном блоке памяти, а также занесение констант в заданный блок памяти.

4.5.1. Открыть маску поиска

Формат команды: ¤M/

По команде '¤M/' открывается регистр маски поиска (логический множитель). Следующие две ячейки, которые открываются по команде <ПС>, содержат границы поиска (нижнюю и верхнюю).

Задать и изменить маску и границы поиска можно обычным способом. Первоначально эти ячейки содержат нули.

Пример.

```
*¤M/000000 177400<ПС>     ;задается маска
000466/000000 1000<ПС>    ;задается нижняя граница по-
                          ;иска
000470/000000 1040        ;задается верхняя граница по-
                          ;иска
```

4.5.2. Найти слово

Формат команды: R;W

где R — объект поиска.

По команде 'R;W' производится поиск слова в заданном блоке памяти. Перед вводом команды 'R;W' пользователь должен определить маску и границы поиска (см. п. 4.5.1). Во время поиска отладчик просматривает только те разряды слова, которые в маске установлены в 1. Затем пользователь задает объект поиска R. Над каждым выбранным словом блока памяти и объектом поиска R производится операция 'исключающее ИЛИ', над результатом этой операции и маской поиска производится операция 'логическое И'. Если результат последней операции равен нулю, то на терминал выводится выбранное слово и его адрес.

Пример.

```
*¤M/000000 177400<ПС>     ;задается маска
R,NNNNNN/000000 1000<ПС>  ;задается нижняя граница
```

```
R,NNNNNN/000000 1040<ВК>  ; задается верхняя граница
*400;W
001010/000770             ; выбранное слово находится
001034/000404             ; в ячейках 001010 и 001034
*
```

в данном примере NNNNNN — внутренний адрес отладчика.

Если маска поиска равна нулю, то на терминал выводятся все ячейки указанного блока.

Если во время поиска нажать СУ/U, то поиск прекращается и на терминал выдается '*'.

4.5.3. Найти исполнительный адрес R

Формат команды: R;E

По команде 'R;E' производится поиск ячеек памяти, в которых содержатся адреса или команды, вызывающие обращение к заданному адресу R программы.

Перед вводом команды 'R;E' следует задать границы поиска (см. п. 4.5.1). В результате на терминал выводятся ячейки, содержащие абсолютный адрес, индексное слово относительного метода адресации или команду ветвления к заданному адресу R.

Пример.

```
¤M/000000 <ПС>            ; открывается регистр маски
R,NNNNNN/001000 <ПС>      ; задается нижняя граница поиска
R,NNNNNN/001060 <ПС>      ; задается верхняя граница поиска
*1034;E                   ; поиск адреса 1034
001016/001006             ; команды ветвления к адресу
001054/002767             ; 001034 находятся в ячейках
                          ; 001016 и 001054
*
```

4.5.4. Открыть регистр константы

Формат команды: ¤C/

По команде '¤C/' открывается регистр константы.

Регистр константы — специальная ячейка, которая устанавливается пользователем. Содержимое регистра константы используется для заполнения блока памяти, занесения константы в ячейку и вычисления адресов.

Пример.

```
*¤C/000000 326<ВК>  ; в регистр константы записывается
                    ; 326
*
```

4.5.5. Записать K в регистр константы

Формат команды: K;C

Записать число K в регистр константы можно, не открывая регистр. По команде 'K;C' в регистр константы записывается константа K.

Пример.

```
*6644;C = 006644   ; в регистр константы записывается 006644
*—17;C = 177761    ; в регистр константы записывается —17
*
```

Регистр константы можно использовать совместно с регистром перемещения. Например, по команде 'N,C;C' регистра перемещения N складывается с содержимым регистра константы и результат записывается в регистр константы.

Пример.

```
*2000;1R
*5000;C = 005000   ; в регистр константы записывается
                   ; 005000
*1,C;C = 007000    ; в регистр константы записывается
                   ; 007000
*
```

Для занесения содержимого регистра константы в открытую ячейку используется команда С.

Пример.

```
*6630/012321 C<ВК>
*6630/007000              ; в ячейку 006630 записывается
                          ; 007000
```

4.5.6. Заполнить блок памяти содержимым регистра константы.

Формат команды: ;F

Команда ';F' используется для последовательного заполнения ячеек указанного блока памяти содержимым регистра константы. Перед вводом команды ';F' следует задать границы блока памяти (см. п. 4.5.1).

Пример.

```
*¤M/000000 <ПС>
R,NNNNNN/000000 7000<ПС>  ; задается нижняя граница
R,NNNNNN/000000 7050<ВК>  ; задается верхняя граница
*12237;C = 12237
*;F
*7000/012237 <ПС>         ; выборочная проверка ячеек
007002/012237 <ВК>        ; блока памяти
*7046/012237 <ПС>
007050/012237 <ПС>
007052/000240
```

4.5.7. Заполнить блок памяти младшим байтом регистра константы

Формат команды: ;I

Команда ';I' используется для последовательного заполнения указанного блока памяти младшим байтом регистра

константы. Перед вводом команды ';I' следует установить границы блока памяти (см. п. 4.5.1).

Пример.

```
*¤M/000000 <ПС>
R,NNNNNN/000000 7200<ПС>  ;задается нижняя граница
R,NNNNNN/000000 7400<ПС>  ;задается верхняя граница
*15062;C = 015062
*;I
*7200\062  <ПС>           ;выборочная проверка ячеек
007201\062  <ПС>          ;блока памяти
007202/062  <ПС>
007203/062  <ВК>
*7400\062
```

4.6. Вычисление смещения

Относительный метод адресации и команды ветвления используют смещение, определяющее адрес перехода. Смещение представляет собой количество слов или байтов вперед или назад от адреса текущей открытой ячейки до адреса перехода. Для определения смещения используется команда 'R;O'.

4.6.1. Вычислить смещение от текущей открытой ячейки до ячейки с адресом R

Формат команды: R;O

Команда 'R;O' используется, если необходимо изменить относительный адрес или адрес перехода команды ветвления посредством замены одного смещения другим. По этой команде отладчик выдает на терминал 16-разрядное смещение (относительный адрес) и 8-разрядное смещение от текущей открытой ячейки до ячейки с адресом R. 8-разрядное смещение выдается на терминал в том случае, если оно находится в диапазоне от —128 до +127 и 16-разрядное смещение четное. После вычисления смещения содержимое текущей открытой ячейки можно изменить.

Пример.

```
*346/000034 414;O 000044 022 22<ВК>
*/000022
```

В этом примере на терминал выведено 16-разрядное смещение 44 и 8-разрядное смещение 22. Оператор указал новое значение 22 и проверил правильность записи.

Величину смещения в команде ветвления можно изменить следующим образом.

Пример.

```
*1024/012467 1052;O 000024 012\067 = ? 12 <ВК>
*/012412
```

В данном примере на терминал выведено смещение от ячейки 001024 до ячейки 001052. По команде '\' оператор открыл ячейку байта, содержащую 067, и указал новое значение 12. Измененный младший байт объединен со старшим.

4.7. Дополнительные команды печати

Ниже описываются команды, используемые для ввода и вывода текстовой информации в символах КОИ—7 или RADIX-50.

4.7.1. Ввод и вывод в символах КОИ—7.

Формат команды: R;KA

где R — адрес, K — счетчик символов.

По команде 'R;KA' отладчик преобразует K байт в символы КОИ—7.

Если K в команде не указано, то подразумевается 1.

Если код не соответствует символу КОИ—7 или коду символа <ВК> или <ПС>, то на терминал выводится знак '?'.

Пример.

```
*1500/040410 <ВК>          ; открывается ячейка 001500
*1500;2A?A                 ; ячейка 001500 содержит код,
<ВК>                       ; не соответствующий символу
                           ; КОИ—7
*1502/044510 1500;4A?AHJ   ; ячейки 001501, 001502, 001503
                           ; содержат коды символов A,
                           ; H, J
```

После выполнения преобразования следует подать команду <ВК> или <ПС>. По команде <ВК> осуществляется возврат каретки, перевод строки и на терминал выводится '*'. По команде <ПС> открывается байт, следующий за последним преобразованным байтом. По желанию, оператор может изменить содержимое открытых ячеек, вводя новую последовательность символов. Можно вводить текст длиной не более K символов.

Если символов меньше 'K', строку следует закончить командой CY/U.

Если вводится ровно 'K' символов текста, программа отладчик указывает адрес следующего доступного байта текста и входит в режим ожидания следующей команды, т. е. осуществляет возврат каретки, перевод строки и выводит на терминал '*'.

4.7.2. Ввод и вывод в символах RADIX-50.

Формат команды: X

Чтобы преобразовать содержимое слова в символы RADIX-50, необходимо открыть слово и ввести команду 'X'.

При этом на терминал выводится трехзначный эквивалент слова в коде RADIX-50. После этого можно выполнить любое из следующих действий:

1) командой <ВК> закрыть ячейку;

2) командой <ПС> закрыть текущую ячейку и открыть следующую;

3) командой '^' закрыть текущую ячейку и открыть предыдущую;

4) указать три символа из разрешенных для представления в RADIX-50. К ним относятся: '.', '¤', пробел, цифры от 0 до 9 и буквы латинского алфавита от A до Z.

Пример.

```
*1260/014712 X = DEB     ; содержимое ячеек 001260 преоб-
                         ; разовано в символы RADIX-50
```

Символы RADIX-50 можно использовать для обозначения адреса ячейки. Это допустимо после ввода команды 'X'.

Пример.

```
*1471/054321 <BK>
*1300/042431 X = KBI DEB/05431
```

В первом примере по команде 'X' содержимое ячейки 001260 выведено на терминал в символах RADIX-50. Коду 014712 соответствуют символы DEB.

Во втором примере после преобразования содержимого ячейки 001300 в символы RADIX-50 оператором указаны символы D, E, B и введена команда '/'. Открыта ячейка по адресу 014712.

Содержимое открытой ячейки можно изменить, указав перед вводом команды <ВК> символы RADIX-50.

Пример.

```
*1000;1R
*1,602/034567 X=IGIAZH<BK>  ; ячейка 001602 содержит
*1,602/005130 X = AZH       ; код 005130, соответству-
                            ; ющий символам RADIX-
                            ; 50, A, Z, H
```

5. СООБЩЕНИЯ ОПЕРАТОРУ

Ниже описаны сообщения, выдаваемые программой отладчик в процессе отладки, вызвавшие их причины и действия оператора.

(Недопустимая команда)?

```
*
```

Причина. Задана недопустимая команда.

Действие. Подать команду правильно.

```
?MON—F—TRAP TO 4 R
```

Причина. Произошло обращение к несуществующей ячейке памяти.

Действие. Повторить отладку. Если отлаживается программа запрашивает прерывание с помощью .TRPSET EMT, программа получает управление с адреса TRPSET.

BE NNNNNN

*

Причина. Встретилась незаданная точка разрыва.

Действие. Устранить точку разрыва.

6. ОТЛАДКА ПРОГРАММ В РАСШИРЕННОЙ ПАМЯТИ С ПОМОЩЬЮ ВИРТУАЛЬНОГО ОТЛАДЧИКА

Программа виртуальный отладчик (VDT) используется для отладки виртуальных и привилегированных заданий в системах с расширенной памятью и мультитерминальных системах. Виртуальный отладчик можно также использовать для отладки заданий в системах с монитором основного — фонового задания и монитором одного задания.

Прежде чем приступить к отладке программы с помощью виртуального отладчика, необходимо связать в загрузочный модуль отлаживаемую программу и виртуальный отладчик VDT.OBJ. Для этого в переключателе /DEBUG редактора связей следует указать имя виртуального отладчика, например:

.LINK/MAP:TT/DEBUG:VDT MYPROG <BK>

В этом примере виртуальный отладчик VDT.OBJ и модуль MYPROG.OBJ отлаживаемой программы образуют загрузочный модуль MYPROG.SAV.

Точка входа для VDT — O.ODT.

Формат команд виртуального отладчика VDT такой же, как и у команд отладчика ODT. Ниже приводятся особенности виртуального отладчика VDT.

Виртуальный отладчик не содержит подпрограмм обслуживания прерывания или подпрограмм приоритета.

Виртуальный отладчик выполняется с тем же приоритетом, что и программа пользователя, и использует программные запросы .TTYIN и .TTYOUT для выполнения ввода-вывода на терминал, что позволяет выполнять VDT с системного терминала.

Поскольку виртуальный отладчик изменяет содержимое слова состояния задания, первоначальное содержимое слова состояния задания должно быть сохранено. Первоначальное

содержимое слова состояния задания можно получить по команде ¤J.

Программа виртуальный отладчик работает в режиме пользователя, а не во внутреннем режиме.

Виртуальные задания, отлаживаемые с помощью виртуального отладчика, не имеют доступа к защищенным системным областям (монитор, векторы, страница ввода — вывода). Привилегированные задания, отлаживаемые с помощью виртуального отладчика, имеют доступ к защищенным областям.

ПАСКАЛЬ

ОПИСАНИЕ ЯЗЫКА

1. ОБЩИЕ СВЕДЕНИЯ

Разработка языка ПАСКАЛЬ преследовала две основные цели. Первая — дать язык, пригодный для обучения программированию как систематической дисциплине, основанной на ряде фундаментальных понятий, ясно и естественно отраженных в этом языке. Вторая — осуществить реализацию этого языка, которая была бы надежна и в то же время эффективна на существующих вычислительных машинах.

Для преподавания программирования требовался новый язык, так как конструкции и свойства используемых ранее языков часто не поддаются убедительному и логическому объяснению. Они слишком часто не соответствуют приученному к систематическому рассуждению мышлению. К этому следует присоединить убеждение, что язык, на котором обучают выражать свои мысли, оказывает глубокое воздействие на навыки мышления и изобретательские способности, и что царящий в существующих языках беспорядок непосредственно сказывается на стиле программирования, формирующемся у обучающихся.

За основу для ПАСКАЛЯ был взят АЛГОЛ—60, откуда в первую очередь заимствованы принципы структурирования и форма выражения.

Главные дополнения по сравнению с АЛГОЛОМ—60 относятся к способам структурирования данных, поскольку недостаточность последних в АЛГОЛЕ—60 была признана главной причиной относительной узости области его применения. Введение записей и файлов сделало возможным решать с помощью ПАСКАЛЯ коммерческие задачи или по крайней мере позволило успешно демонстрировать подобные задачи в курсах программирования.

Алгоритмы и программы вычислительной машины состоят

из двух частей: описания действий, которые должны быть осуществлены, и описания обрабатываемых этими действиями данных. Действия задаются так называемыми операторами, а данные — описаниями и определениями.

Данные представлены значениями переменных. Каждая переменная, входящая в некоторый оператор, должна быть введена некоторым описанием переменной, которое приписывает этой переменной идентификатор и тип данных. Именно тип данных определяет множество значений, которое может принимать данная переменная. В ПАСКАЛЕ тип может быть задан либо прямо в описании переменной, либо указан идентификатором типа, в последнем случае этот идентификатор должен быть введен явным определением типа.

Базисными являются скалярные типы данных. Их определение задает упорядоченное множество значений, т. е. вводит идентификаторы, обозначающие каждое значение этого множества. Помимо скалярных типов, вводимых определениями, в ПАСКАЛЕ имеется четыре стандартных скалярных типа: логический «BOOLEAN», целый «INTEGER», литерный «CHAR» и вещественный «REAL». Значения этих типов, кроме логического, обозначаются не идентификаторами, а числами и заключенными в кавычки литералами, которые синтаксически отличны от идентификаторов. Множество значений литерного типа — это множество литер, имеющихся на печатном устройстве данной установки (КОИ—7).

Тип можно определить как отрезок некоторого скалярного типа — указанием наименьшего и наибольшего значений отрезка.

Структурные типы данных определяются заданием типа их компонент и указанием способа организации. Способы организации отличаются механизмом извлечения компонент из переменной сложного типа. В ПАСКАЛЕ имеется четыре способа организации данных: массив, запись, множество и файл.

При организации данных в массив все компоненты относятся к одному и тому же типу. Отдельная компонента выделяется селектором массива, или вычисляемым индексом, тип которого указывается в определении типа массива. Тип индекса должен быть скалярным. Как правило, это тип, определяемый программистом, или отрезок целого типа. По значению, принадлежащему типу индекса, селектор массива выдает значение, имеющее тип компонент массива. Таким образом, каждая переменная — массив может рассматриваться как отображение типа индекса на тип компонент. Время вы-

борки не зависит от селектора (индекса), поэтому массивы называются структурами данных с произвольным доступом.

При организации данных в запись компоненты (называемые полями) могут быть разного типа. Чтобы тип выбираемой компоненты можно было установить прямо по тексту программы (без ее исполнения), селекторы записей не содержат вычисляемых значений, а являются идентификаторами, однозначно определяющими извлекаемую компоненту. Эти идентификаторы описываются в определении типа записи, поэтому время доступа к выбираемой компоненте не зависит от селектора и, следовательно, записи также являются структурами с произвольным доступом.

Может быть указано, что тип записи состоит из нескольких вариантов. Это значит, что различные переменные одного и того же типа могут иметь значениями определенным образом отличающиеся записи. Различия могут касаться числа и типов компонент. Вариант, являющийся текущим значением переменной — записи, отмечается в общем для всех вариантов поле, называемым полем признака. Как правило, общая всем вариантам часть будет содержать несколько компонент, включая и поле признака.

Организация данных во множестве определяет множество значений, являющихся множеством — степенью базового типа. Базовый тип должен быть скалярным и определяться программистом или представлять собой отрезок целого типа.

Файл — это последовательность однотипных компонент. Она определяет естественное упорядочение компонент. В каждый момент непосредственно доступна только одна компонента. Доступ к другим компонентам можно получить в результате последовательного продвижения по файлу. Файл заполняется последовательным присоединением новых компонент к концу файла. Определение типа файла не задает числа его компонент.

Переменные, заданные явными описаниями, называются статическими. Описание связывает с переменной некоторый идентификатор, используемый для указания этой переменной. С другой стороны, переменные могут создаваться некоторым оператором. При таком динамическом создании переменной выдается так называемый указатель (заменяющий явный идентификатор). Он используется впоследствии для указания на такую переменную. Этот указатель может быть присвоен переменным типа указатель. Каждая переменная — указатель может принимать только значения, являющиеся указателями на значения одного и того же типа «Т»; говорят, что она за-

креплена за этим типом «Т». Она может, впрочем, получить также значение «NIL», которое не указывает ни на какую переменную. Поскольку переменные — указатели могут быть компонентами создаваемых динамических переменных структурных типов, то с помощью этих указателей можно представлять любые конечные графы.

Основным оператором является оператор присваивания, который приписывает присвоить только что вычисленное значение некоторой переменной (или компоненте переменной). Значение может быть получено в результате вычисления выражения. Выражения состоят из переменных, констант, множеств, операций и функций, которые выполняют действия над изображенными величинами и выдают новые значения. Переменные, константы и функции либо описываются в программе, либо являются стандартными объектами. ПАСКАЛЬ содержит фиксированный набор операций, каждая из которых может рассматриваться как отображение из типов операндов в тип результата. Все операции делятся на четыре группы:

1) арифметические операции — сложение, вычитание, изменение знака, умножение, деление и нахождение остатка;

2) логические операции — отрицание, дизъюнкция, конъюнкция;

3) операции над множествами — объединение, пересечение, разность;

4) операции — отношения — равенство, неравенство, порядок, принадлежность и включение (для множеств).

Результаты операций отношений принадлежат логическому типу. Отношения порядка применимы только к скалярным типам.

Оператор процедуры вызывает исполнение соответствующей процедуры (см. ниже). Операторы присваивания и процедуры являются «кирпичиками» или компонентами, из которых строятся структурные операторы, предписывающие последовательное, выборочное или повторное исполнение своих компонент. Последовательное исполнение операторов задается составным оператором, условное или выборочное исполнение — оператором «Если» и оператором «Выбор», а повторное исполнение — циклами. Оператор «Если» служит для подчинения исполнения оператора значению выражения логического типа, а оператор «Выбор» делает возможным выбор между несколькими операторами в соответствии со значением некоторого селектора. Цикл с параметрами (оператор «Для») употребляется, когда число итераций известно заранее, а в остальных случаях используются циклы с условием продол-

жения (оператор «Пока») или циклы с условием окончания (оператор «Повтор»).

Оператору может быть дано имя (идентификатор), с помощью которого можно обращаться к этому оператору. Такой оператор называется процедурой, а его описание — описанием процедуры. Описание может содержать множество описаний переменных, определение типов и дальнейшее описание процедур. Так описанные переменные, типы и процедуры могут упоминаться только в пределах самой процедуры и поэтому называются локальными для данной процедуры. Их идентификаторы имеют смысл только в пределах программного текста, составляющего описание процедуры и называемого областью определения этих идентификаторов. Поскольку одна процедура может быть описана как локальная по отношению к другой, области определения могут быть вложены одна в другую. Объекты, описанные в главной программе, т. е. не локальные ни для какой процедуры, называются глобальными.

Процедура имеет фиксированное число параметров, каждый из которых изображается в пределах процедуры некоторым идентификатором, называемым формальным параметром. При обращении к процедуре каждому формальному параметру должна быть сопоставлена некоторая фактическая величина. Эта величина называется фактическим параметром. Существует три рода параметров: значения, переменные, процедуры (функций). В первом случае фактический параметр является выражением, значение которого вычисляется один раз. Соответствующий формальный параметр является локальной переменной, которой присваивается результат этого вычисления перед выполнением процедуры (функции). В случае параметра — переменной фактический параметр является переменной и формальный параметр служит обозначением этой переменной. Значения индексов, если они есть, вычисляются перед выполнением процедуры (функции). В случае параметра — процедуры или параметра — функции фактическим параметром является идентификатор процедуры или функции.

Описание функции аналогичны описаниям процедур. Отличие заключается в том, что функции выдают результат, тип которого ограничен скалярными типами и должен быть специфицирован в описании функции, поэтому функции могут быть использованы как составляющие части выражений. Для устранения побочных эффектов в пределах тела функции следует избегать присваиваний значений нелокальным переменным.

2. СПОСОБ ОПИСАНИЯ ЯЗЫКА

В соответствии с традиционными формами Бэкуса — Наура синтаксические конструкции обозначаются словами, заключенными в угловые скобки «<» и «>». Эти слова описывают также природу или смысл данных конструкций и используются в последующем описании семантики. Возможное повторение некоторой конструкции указывается звездочкой (* — ноль или более повторений), а также знаком логическое «И» (& — одно или несколько повторений). Если повторяемая конструкция состоит более чем из одного элемента, она заключается в метаскобки «(*» и «*)», что означает повторение ноль или несколько раз.

Основная лексика состоит из символов, подразделяющихся на буквы, цифры и специальные символы:

<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!Z!
А!Б!В!Г!Д!Е!Ж!З!И!Й!К!Л!М!Н!О!П!Р!С!Т!У!
Ф!Х!Ц!Ы!Ь!Я

<Восьмер.цифра>::=0!1!2!3!4!5!6!7

<Цифра>::=<восьмер.цифра>!8!9

<Специальный символ>::=+!−!*!/! NOT!AND!OR!=!<
>!>!<!<=!>=!(!)![!]!:=!.!,!;!:! !^!DIV!MOD!
NIL!IN!IF!THEN!ELSE!CASE!OF!REPEAT!
UNTIL!WHILE!DO!FOR!TO!DOWNTO!BEGIN!
END!WITH!GOTO!CONST!VAR!TYPE!ARRAY!
RECORD!SET!FILE!FUNCTION!PROCEDURE!
LABEL!PACKED

Конструкция:

<Скобка><любая последовательность символов, не включающая <скобку> и «Ш, Щ»,><скобка> называется комментарием и может быть вставлена между любыми двумя идентификаторами, числами или специальными символами. Комментарий может быть удален из текста программы без изменения его значения.

В ПАСКАЛЕ допускается три типа <скобок> для выделения комментариев: <Скобка>::={...}! (*...*)! /*...*/

Все типы скобок являются взаимозаменяемыми: для начала комментария можно использовать один тип, а для завершения — другой тип скобок.

3. ЭЛЕМЕНТЫ И ОСНОВНЫЕ КОНСТРУКЦИИ ЯЗЫКА

Идентификаторы служат для обозначения констант, типов, переменных, процедур и функций. В пределах области определенности идентификатора, т. е. в процедуре или в функции, в которой он описан, смысл идентификатора должен быть однозначным.

<ИДЕНТИФИКАТОР>::=<БУКВА><БУКВА ИЛИ ЦИФРА>*

<БУКВА ИЛИ ЦИФРА>::=<БУКВА>!<ЦИФРА>

ПАСКАЛЬ допускает любую длину идентификаторов, причем все знаки являются значимыми. Знаки кириллицы (за исключением ч,ш,щ,э,ю) могут также использоваться, но будут интерпретироваться как и соответствующие латинские. Следует заметить, что в стандарте ПАСКАЛЯ рекомендуется, чтобы первые 8 знаков идентификаторов были уникальными.

В связи с ограничениями на формат объектного модуля идентификаторы внешних фортрановских процедур должны быть уникальными по первым 6-ти знакам.

Для чисел, являющихся константами целого или вещественного типа, используется обычная десятичная система записи буква «Е», предшествующая порядку, читается как «Умножить на 10 в степени».

<ЦЕЛОЕ БЕЗ ЗНАКА>::= <ЦИФРА>&!<ВОСЬМЕР. ЦИФРА>&B

<ВЕЩЕСТВЕННОЕ БЕЗ ЗНАКА>::=
<ЦИФРА>&. <ЦИФРА>&!
<ЦИФРА>&.<ЦИФРА>&E<ПОРЯДОК>!
<ЦИФРА>&E<ПОРЯДОК>

<ЧИСЛО БЕЗ ЗНАКА>::= <ЦЕЛОЕ БЕЗ ЗНАКА>!
<ВЕЩЕСТВЕННОЕ БЕЗ ЗНАКА>

<ПОРЯДОК>::= <ЦИФРА>&!<ЗНАК><ЦИФРА>&

<ЗНАК>::= +!—

Примеры:

1 100 0.1 5E—3 87.35E+8 12B

Последовательности литер, заключенные в кавычки, называются текстами. Тексты, состоящие из единственной литеры, являются константами литерного типа. Тексты, состоящие из N (N>1) литер, заключенных в кавычки, являются константами типа: ARRAY[1..N] OF CHAR

ПРИМЕЧАНИЕ. Если текст должен содержать кавычку, то она повторяется дважды: <ТЕКСТ>::= '<ЛИТЕРА>&'

Примеры:

'A' ';' '''' 'PASCAL'

4. ЭЛЕМЕНТЫ И ВВОД/ВЫВОД ДАННЫХ

4.1. Определения констант

Определение константы вводит идентификатор как синоним некоторой константы:

<ИДЕНТИФИКАТОР КОНСТАНТЫ>::= <ИДЕНТИФИКАТОР>
<КОНСТАНТА БЕЗ ЗНАКА>::= <ЧИСЛО БЕЗ ЗНАКА> !
<ТЕКСТ>!<ИДЕНТИФИКАТОР КОНСТАНТЫ>!
NIL
<КОНСТАНТА>::= <ЧИСЛО БЕЗ ЗНАКА> ! <ЗНАК><ЧИСЛО
БЕЗ ЗНАКА>
<ИДЕНТИФИКАТОР КОНСТАНТЫ>!
<ЗНАК><ИДЕНТИФИКАТОР КОНСТАНТЫ> !
<ТЕКСТ>
<ОПРЕДЕЛЕНИЕ КОНСТАНТЫ>::= <ИДЕНТИФИКАТОР> =
<КОНСТАНТА>

В каждой реализации определены следующие стандартные идентификаторы:
EOL — равен управляющему символу, обозначающему конец строки;
ALFALENG — равен 10.

4.2. Определения типов данных

Тип данных задает множество значений, которые могут принимать переменные этого типа, и связывает с этим типом идентификатор.

<ТИП>::= <ПРОСТОЙ ТИП> ! <СТРУКТУРНЫЙ
ТИП> ! <ТИП УКАЗАТЕЛЯ>
<ОПРЕДЕЛЕНИЕ ТИПА>::= <ИДЕНТИФИКАТОР>
= <ТИП>

4.2.1. Простые типы

<ПРОСТОЙ ТИП>::= <СКАЛЯРНЫЙ ТИП> ! <ОТРЕЗОК ТИПА> ! <ИДЕНТИФИКАТОР ТИПА>
<ИДЕНТИФИКАТОР ТИПА>::= <ИДЕНТИФИКАТОР>

4.2.1.1. Скалярные типы

Скалярный тип определяет упорядоченное множество значений посредством перечисления идентификаторов, обозначающих эти значения:

<СКАЛЯРНЫЙ ТИП>::=(<ИДЕНТИФИКАТОР>(*,
<ИДЕНТИФИКАТОР>*))

Примеры:
(КРАСН, ОРАНЖ, ЖЕЛТ, ЗЕЛ, ГОЛУБОЙ)
(ЯНВАРЬ,ФЕВРАЛЬ,МАРТ)
(ЗИМА,ВЕСНА,ЛЕТО,ОСЕНЬ)

Ко всем скалярным типам (исключая REAL) применимы следующие функции:
SUCC — следующее значение (в порядке перечисления);
PRED — предшествующее значение (в порядке перечисления).

4.2.1.2. Стандартные скалярные типы

Следующие типы являются стандартными в ПАСКАЛЕ:

Целый тип (INTEGER) — значения образуют подмножество целых чисел в границах (—32768..32767). Беззнаковые целые могут быть объявлены с помощью диапазона (0..65535). Следует помнить, что арифметические переполнения определяются только при умножении и делении знаковых целых.

TYPE UNSIGNEDIGIT=0..65535

Для целых чисел определен идентификатор MAXINT=32767.

Вещественный тип (REAL) — значения являются подмножеством множества вещественных чисел вещественные числа (REAL) имеют диапазон 1E—38..1E+38 и точность порядка 7-ми десятичных знаков. Арифметические переполнения определяются для всех операций, а потеря значимости не диагностируется и приводит к нулевому результату. Стандартные трансцедентные функции имеют точность до 6 десятичных цифр.

Логический тип (BOOLEAN) — значения представляют собой истинные значения, обозначаемые идентификаторами TRUE и FALSE;

Литерный тип (CHAR) — значения суть множество литер, зависящее от конкретной реализации, и изображаются самими литерами, заключенными в кавычки. В данной реализации использует полный набор знаков КОИ—7. Каждый знак хранится в одном байте (8 разрядов).

4.2.1.3. Отрезки типов

Тип может быть определен как отрезок другого скалярного типа посредством указания наибольшего и наименьшего значений отрезка. Первая константа задает нижнюю границу, которая не должна быть больше верхней.

<ОТРЕЗОК ТИПА>::=<КОНСТАНТА>..<КОНСТАНТА>

Примеры:

1..100
—10..+10
Пн..Пт

4.2.2. Структурные типы

Структурный тип данных характеризуется типом (или типами) своих компонент и способом их организации.

<СТРУКТУРНЫЙ ТИП>::=<ТИП МАССИВА> !
<ТИП ЗАПИСИ> !
<ТИП МНОЖЕСТВА> ! <ТИП ФАЙЛА>

Стандартный ПАСКАЛЬ содержит процедуры «PACK» и «UNPACK» для эффективной работы с компонентами структурного типа. В данном ПАСКАЛЕ атрибут «PACKED», допустимый в стандарте, игнорируется, а использование процедур «PACK» и «UNPACK» недопустимо.

4.2.2.1. Типы массивов

Тип массива — это структура, состоящая из фиксированного числа компонент одного и того же типа, называемого типом компонент. Элементы массивов выбираются по индексам — значениям, принадлежащим к так называемому типу индексов. Определение типа массива задает как тип компонент, так и тип индексов:

<ТИП МАССИВА>::=
ARRAY[<ТИП ИНДЕКСА>(*,<ТИП ИНДЕКСА>*)]
OF <ТИП КОМПОНЕНТ>
<ТИП ИНДЕКСА>::=<ПРОСТОЙ ТИП>
<ТИП КОМПОНЕНТ>::=<ТИП>

Если задано N типов индексов, то тип массива называется N-мерным; его компоненты выбираются с помощью N индексов.

Примеры:

ARRAY[1..100] OF REAL
ARRAY[1..10,1..20] OF 0..99
ARRAY[BOOLEAN] OF ЦВЕТ

4.2.2.2. Типы записей

Тип записи — это структура, состоящая из фиксированного числа компонент, возможно, разных типов. Определение типа записи задает для каждой компоненты, называемой полем, ее тип и обозначающий это поле идентификатор. Областью определенности этих идентификаторов поля является само определение записи, однако они доступны также в выборках поля, ссылающихся на переменную — запись данного типа.

Тип записи может иметь несколько вариантов; в таком случае некоторое поле отмечается как поле признака: значение этого поля показывает, какой из вариантов переменной — записи рассматривается в данный момент. Каждый вариант идентифицируется меткой выбора, являющейся константой типа поля признака:

```
<ТИП ЗАПИСИ>::=RECORD<СПИСОК ПОЛЕЙ>END
<СПИСОК ПОЛЕЙ>::=<ОБЩАЯ ЧАСТЬ>!
                 <ОБЩАЯ ЧАСТЬ>;<ВАРИАНТНАЯ ЧАСТЬ>!
                 <ВАРИАНТНАЯ ЧАСТЬ>
<ОБЩАЯ ЧАСТЬ>::=<СЕКЦИЯ ЗАПИСИ>
               (*;<СЕКЦИЯ ЗАПИСИ>*)
<СЕКЦИЯ ЗАПИСИ>::=<ИДЕНТИФИКАТОР ПОЛЯ>
                 (*,<ИДЕНТИФИКАТОР ПОЛЯ>*):<ТИП>
<ВАРИАНТНАЯ ЧАСТЬ>::=CASE<ПОЛЕ ПРИЗНАКА>:
          <ИДЕНТИФИКАТОР ТИПА> OF <ВАРИАНТ>
          (*;<ВАРИАНТ>*)
<ВАРИАНТ>::=<СПИСОК МЕТОК ВЫБОРА>:
            (<СПИСОК ПОЛЕЙ>)!
            <СПИСОК МЕТОК ВЫБОРА>
<СПИСОК МЕТОК ВЫБОРА>::=<МЕТКА ВЫБОРА>
                       (*,<МЕТКА ВЫБОРА>*)
<МЕТКА ВЫБОРА>::=<КОНСТАНТА БЕЗ ЗНАКА>
<ПОЛЕ ПРИЗНАКА>::=<ИДЕНТИФИКАТОР>
```

Примеры:

```
RECORD ДЕНЬ:1..31;
       МЕС:1..12;
       ГОД:INTEGER
  END
RECORD ИМЯ,ФАМИЛИЯ:ALFA;
       ВОЗРАСТ:0..99;
       ЖЕНАТ:BOOLEAN
  END
RECORD X,Y:REAL;
       ПЛОЩАДЬ:REAL;
    CASE Ф: ФОРМА OF
    ТРЕУГОЛЬНИК:      (СТОРОНА:REAL;
                       НАКЛОН,УГОЛ1,УГОЛ2:УГОЛ);
    ПРЯМОУГОЛЬНИК:  (СТОРОНА,СТОРОНА2:REAL;
                       СКЛОН,УГОЛ3:УГОЛ);
    КРУГ:(ДИАМЕТР:REAL)
   END
```

4.2.2.3. Типы множеств

Тип множества определяет множество значений, являющихся множеством — степенью (множеством всех подмножеств) соответствующего базового типа. Базовыми типами не могут быть структурные типы. Над каждым типом множеств определены операции:

\+ объединение
* пересечение
— разность множеств (бинарное дополнение)
IN отношение принадлежности (членства)

```
<ТИП МНОЖЕСТВА>::=SET OF <ТИП БАЗЫ>
<ТИП БАЗЫ>::=<ПРОСТОЙ ТИП>
```

В ПАСКАЛЕ, реализованном для ДВК, число элементов множества не может превышать 64. Определено стандартное множество «SET OF CHAR», что эквивалентно «SET OF SPACE..UNDERLINE», где SPACE = CHR(40B) и UNDERLINE=CHR(137B). Таким образом, «SET OF CHAR» не включает букв кириллицы и управляющих знаков, имеющих коды менее 40 (восьмеричное).

4.2.2.4. Типы файлов

Определение типа файла задает структуру, состоящую из последовательности однотипных компонент. Число компонент, называемое длиной файла, определением типа файла не фиксируется. Файл, имеющий ноль компонент, называется пустым; файлы, тип компонент которых литерный, называются текстовыми файлами.

<ТИП ФАЙЛА>::=FILE OF <ТИП>

Следующий тип является стандартным:

TYPE TEXT = FILE OF CHAR

Данный ПАСКАЛЬ не позволяет использовать файлы файлов, однако, можно применять массивы файлов, а также записи, содержащие в качестве компонентов файлы.

4.2.3. Типы указателей

Доступ к описанным в программе переменным можно получить через их идентификаторы. Они существуют все время, пока выполняется процедура (область определенности), в которой эти переменные локальны, в связи с чем последние называются статическими или статически размещаемыми. С другой стороны, переменные могут создаваться и динамически, т. е. без всякой связи со структурой программы. Эти динамические переменные создаются стандартной процедурой «NEW»; поскольку они не входят в явные описания переменных, они не могут быть обозначены именем. Доступ к таким переменным осуществляется через указательные значения (ссылки), выдаваемые после создания динамической переменной. Каждый тип указателей состоит, таким образом, из неограниченного множества указывающих на однотипные элементы значений. Над указателями не определено никаких операций, кроме проверки на равенство. Значение «NIL» принадлежит всем типам указателей, но оно не указывает ни на какой элемент.

<ТИП УКАЗАТЕЛЯ>::=^<ИДЕНТИФИКАТОР ТИПА>

Примеры определений типов:

ЦВЕТ = (КРАСН, ОРАНЖ, ЖЕЛТ, ЗЕЛ, ГОЛУБОЙ, СИНИЙ, ФИОЛ)

ПОЛ = (МУЖ,ЖЕН)

```
TEXT = FILE OF CHAR
ФОРМА = (ТРЕУГОЛЬНИК,ПРЯМОУГОЛЬНИК,КРУГ)
КАРТА = ARRAY[1..80] OF CHAR
ALFA = ARRAY[1..ALFALENG] OF CHAR
КОМПЛ = RECORD RE, IM: REAL END
ЛИЦО = RECORD ФАМИЛИЯ,ИМЯ:ALFA;
          ВОЗРАСТ:INTEGER;
          ЖЕНАТ:BOOLEAN;
        ОТЕЦ,РЕБЕНОК,БРАТ:^ЛИЦО;
CASE РОД: ПОЛ OF
      МУЖ: (ВОЕННООБЯЗАН,ЛЫС:BOOLEAN);
      ЖЕН: (БЕРЕМ:BOOLEAN;
            МЕРКА:ARRAY[1...3] OF INTEGER)
END
```

4.3. Описание и изображение переменных

Описания переменных состоят из списка идентификаторов, обозначающих новые переменные, за которыми следует их тип:

<ОПИСАНИЕ ПЕРЕМЕННОЙ>::=
<ИДЕНТИФИКАТОР> (*,<ИДЕНТИФИКАТОР>*):
<ТИП>

Каждое описание переменной — файла «Ф» с компонентами типа «Т» подразумевает дополнительное описание так называемой буферной переменной типа «Т». Эта буферная переменная изображается Ф^ и служит для присоединения к файлу компонент при его заполнении и для получения доступа к файлу при чтении.

В отдельных случаях бывает удобно иметь возможность обращаться к абсолютным ячейкам памяти, например, к регистрам внешних устройств или таблицам операционной системы. Для этой цели в данном ПАСКАЛЕ следует использовать ключевое слово ORIGIN в объявлении переменной. Например, следующая программа считывает знак с клавиатуры системного терминала на физическом уровне:

```
CONST   READY =200B;
VAR     KBCSR ORIGIN 177560B,
        KBDAT ORIGIN 177562B: INTEGER;
        CH:CHAR;
BEGIN
        WHILE (KBCSR AND READY)=0 DO; (*ждать*)
        C := CHR(KBDAT);
END.
```

Примеры:

```
X, Y, Z: REAL
U, V: КОМПЛ
I, J: INTEGER
K: 0..9
P, Q: BOOLEAN
ОПЕРАЦИЯ: (ПЛЮС, МИНУС, УМНОЖ)
A: ARRAY [0..63] OF REAL
B: ARRAY [ЦВЕТ, BOOLEAN] OF КОМПЛ
Ц: ЦВЕТ
F: FILE OF CHAR
ОТТЕНОК1, ОТТЕНОК2: SET OF ЦВЕТ
Р1, Р2: ^ЛИЦО
```

Изображения переменных обозначают либо полную переменную, либо некоторую компоненту переменной, либо переменную, на которую ссылается некоторый указатель. Предполагается, что переменные в последующих примерах описаны, как указано выше.

<ПЕРЕМЕННАЯ>::= <ПОЛНАЯ ПЕРЕМЕННАЯ> ! <КОМПОНЕНТНАЯ ПЕРЕМЕННАЯ> ! <УКАЗУЕМАЯ ПЕРЕМЕННАЯ>

4.3.1. Полные переменные

Полная переменная изображается своим идентификатором:

<ПОЛНАЯ ПЕРЕМЕННАЯ>::= <ИДЕНТИФИКАТОР ПЕРЕМЕННОЙ>

<ИДЕНТИФИКАТОР ПЕРЕМЕННОЙ>::= <ИДЕНТИФИКАТОР>

4.3.2. Компонентные переменные

Компонентная переменная изображается соответствующей переменной, за которой следует задающий нужную компоненту селектор. Форма селектора зависит от способа организации типа переменной:

<КОМПОНЕНТНАЯ ПЕРЕМЕННАЯ>::=<ПЕРЕМЕННАЯ С ИНДЕКСАМИ>! <УКАЗАТЕЛЬ ПОЛЯ> ! <БУФЕР ФАЙЛА>

4.3.2.1. Переменные с индексами

Компонента N-мерной переменной — массива обозначается с помощью этой переменной, за которой следует список из N индексных выражений:

<ПЕРЕМЕННАЯ С ИНДЕКСОМ>::= <ПЕРЕМЕННАЯ—МАССИВ>[<ВЫРАЖЕНИЕ>

(*,<ВЫРАЖЕНИЕ>*)]
<ПЕРЕМЕННАЯ—МАССИВ>::= <ПЕРЕМЕННАЯ>

Типы индексных выражений должны соответствовать типам индексов в описании соответствующего типа массива.

Примеры:

A [12]
A [I+J]
B [КРАСН, TRUE]. IM
B [SUCC(Ц),P AND Q]

4.3.2.2. Указатели полей

Компонента переменной — записи обозначается через соответствующую переменную — запись, за которой следует идентификатор поля нужной компоненты:

<УКАЗАТЕЛЬ ПОЛЯ>::= <ПЕРЕМЕННАЯ—ЗАПИСЬ>.
<ИДЕНТИФИКАТОР ПОЛЯ>
<ПЕРЕМЕННАЯ—ЗАПИСЬ>::= <ПЕРЕМЕННАЯ>
<ИДЕНТИФИКАТОР ПОЛЯ>::= <ИДЕНТИФИКАТОР>

Примеры:

U.RE
B [КРАСН, TRUE]. IM
P2∧.МЕРКА

4.3.2.3. Буферы файлов

В любой момент непосредственно доступна только одна компонента, определяемая текущей позицией файла (например, головкой считывания — записи). Эта компонента называется текущей компонентой файла; она представляется переменной, называемой буфером файла:

<БУФЕР ФАЙЛА>::= <ПЕРЕМЕННАЯ—ФАЙЛ>
<ПЕРЕМЕННАЯ—ФАЙЛ>::= <ПЕРЕМЕННАЯ>

4.3.3. Указуемые переменные

<УКАЗУЕМАЯ ПЕРЕМЕННАЯ>::= <ПЕРЕМЕННАЯ — УКАЗАТЕЛЬ>∧
<ПЕРЕМЕННАЯ — УКАЗАТЕЛЬ>::= <ПЕРЕМЕННАЯ>

Если «Р» является переменной — указателем, закрепленной за типом «Т», то «Р» обозначает эту переменную и ее указательное значение, тогда как Р∧ обозначает переменную типа «Т», на которую ссылается «Р»;

Примеры:

P1∧.ОТЕЦ
P1∧.БРАТ∧.РЕБЕНОК

5. ВЫРАЖЕНИЯ

Выражения — это конструкции, задающие правила вычислений для определения значений переменных и получения но-

вых значений применением действий. Они состоят из операндов, т. е. переменных и констант, знаков операций и функций.

Правила композиций задают приоритеты знаков операций, разбивая их на четыре класса. Наивысший приоритет имеет знак операции NOT, затем следуют так называемые знаки операций типа умножения, затем — знаки операций типа сложения и, наконец, знаки отношений. Последовательность операций одного приоритета исполняется слева направо. Эти отношения приоритетов отражены в следующем синтаксисе:

```
<МНОЖИТЕЛЬ> ::= <ПЕРЕМЕННАЯ> ' <КОНСТАНТА БЕЗ ЗНАКА> '
                <ОБРАЩЕНИЕ К ФУНКЦИИ> ' <МНОЖЕСТВО>
               (<ВЫРАЖЕНИЕ>) ' NOT<МНОЖИТЕЛЬ>
<МНОЖЕСТВО> ::= [<ВЫРАЖЕНИЕ> (*, <ВЫРАЖЕНИЕ> *) ] ' [ ]
<ТЕРМ> ::= <МНОЖИТЕЛЬ> '
          <ТЕРМ><ЗНАК ОПЕРАЦИИ ТИПА УМНОЖЕНИЯ><МНОЖИТЕЛЬ>
<ПРОСТОЕ ВЫРАЖЕНИЕ> ::= <ТЕРМ> ' <ПРОСТОЕ ВЫРАЖЕНИЕ>
                        <ЗНАК ОПЕРАЦИИ ТИПА СЛОЖЕНИЯ><ТЕРМ> '
                        <ЗНАК ОПЕРАЦИИ ТИПА СЛОЖЕНИЯ><ТЕРМ>
<ВЫРАЖЕНИЕ> ::= <ПРОСТОЕ ВЫРАЖЕНИЕ> '
               <ПРОСТОЕ ВЫРАЖЕНИЕ><ЗНАК ОТНОШЕНИЯ><ПРОСТОЕ
                ВЫРАЖЕНИЕ>
```

Выражения, являющиеся элементами множества, должны быть одного и того же типа, являющегося типом базы данного множества. «[ ]» обозначает пустое множество.

Примеры:

МНОЖИТЕЛИ:	X
	15
	(X+Y+Z)
	SIN(X+Y)
	[КРАСН, Ц, ЗЕЛ]
	∧P
ТЕРМЫ:	X*Y
	I/(1—I)
	P AND Q
	(X<=Y) AND (Y<Z)
ПРОСТЫЕ ВЫРАЖЕНИЯ:	X+Y
	—X
	ОТТЕНОК1 OR ОТТЕНОК 2
	I*J+1
ВЫРАЖЕНИЯ:	X=15
	Ц IN ОТТЕНОК1

5.1. Операция отрицания

Знак операции «NOT», применимый к операнду логического типа, обозначает отрицание.

5.2. Операции типа умножения

<ЗНАК ОПЕРАЦИИ ТИПА УМНОЖЕНИЯ>::=
*!/!DIV!MOD!AND

Таблица 1

Знак операц.	Операция	Тип операндов	Тип результата
1	2	3	4
*	Умножение	REAL, INTEGER	INTEGER, если оба операнда имеют этот тип, REAL в остальных случаях
*	Пересечение множеств	Любой тип множеств T	T
/	Деление	REAL, INTEGER	REAL
DIV	Деление нацело (с усечением)	INTEGER	INTEGER
MOD	Нахождение остатка от деления нацело	INTEGER	INTEGER
AND	Коньюнкция	BOOLEAN	BOOLEAN

5.3. Операции типа сложения

<ЗНАК ОПЕРАЦИИ ТИПА СЛОЖЕНИЯ> ::=!+!—!OR

Таблица 2

Знак операции	Операция	Тип операндов	Тип результата
1	2	3	4
+	Объединение множеств	Любой тип множества T	T
+	Сложение	REAL, INTEGER	INTEGER, если оба операнда имеют этот тип, REAL в остальных случаях
—	Вычитание	REAL, INTEGER	
—	Разность множеств	Любой тип множества T	T
OR	Дизъюнкция	BOOLEAN	BOOLEAN

Знаки операций «—» и «+», если они используются с одним и тем же операндом, обозначают соответственно изменение знака и тождественную операцию.

5.4. Логические операции с целыми числами

Булевские операции AND, OR и NOT имеют расширенную область действия и могут применяться для целых чисел. Операция будет выполняться над 16-ю разрядами операндов. Это расширение позволяет производить проверку и установку отдельных разрядов в целых переменных (например, для регистров внешних устройств).

Пример: BYTE := ORD(CH) AND 377B;

5.5. Знаки отношений

<ЗНАК ОТНОШЕНИЯ> ::= ! = ! <> ! < ! <= !>= ! > ! IN

Таблица 3

Знак отношения	Тип операндов	Результат
1	2	3
= <>	Любой тип, кроме типа файла	BOOLEAN
< > <= >=	Любой скалярный тип или отрезок типа	BOOLEAN
IN	Любой скалярный тип или отрезок типа и связанный с ним соответствующий тип множеств	BOOLEAN

Заметим, что все скалярные типы определяют упорядоченные множества значений, в частности, FALSE<TRUE. Знаки отношений «<=» и «>=» можно использовать для сравнения значений типа множества, при этом они обозначают соответственно отношения включения.

Знаки отношений «<=», «<», «>», «>=» можно также применять к упакованным массивам литерного типа; в этом случае они обозначают алфавитное отношение порядка, соответствующее наличному множеству литер.

5.6. Указатели функции

Указатель функции предписывает обращение к соответствующей процедуре — функции. Оно состоит из обозначающего эту функцию идентификатора и списка фактических параметров. Параметры, которыми могут быть переменные, выражения, процедуры и функции, подставляются вместо соответствующих формальных параметров:

<УКАЗАТЕЛЬ ФУНКЦИИ> ::= <ИДЕНТИФИКАТОР ФУНКЦИИ> !
<ИДЕНТИФИКАТОР ФУНКЦИИ>
(<ФАКТИЧЕСКИЙ ПАРАМЕТР>
(*, <ФАКТИЧЕСКИЙ ПАРАМЕТР> *))
<ИДЕНТИФИКАТОР ФУНКЦИИ> ::= <ИДЕНТИФИКАТОР>

Примеры:

```
SUM(A,100)
SIN(X+Y)
EOF (F)
ORD(F^)
```

5.7. Адресный оператор «@»

В состав данного языка ПАСКАЛЬ входит унарный оператор, обозначаемый «@». Если его использовать с переменной типа «А», то результатом будет значение типа «^А» (указатель на «А»). Адресная операция может использоваться для работы со списковыми структурами данных или для передачи адресов в программы, написанные на языке МАКРОАССЕМБЛЕР. Например:

```
TYPE    BLOCK=ARRAY[0..511] OF CHAR;
VAR     BUFFER:BLOCK;
BEGIN
        (* запомнить адрес буфера *)
        XRB.XRLOC:=@BUFFER
END;
```

6. ОПЕРАТОРЫ

Операторы обозначают алгоритмические действия. Про них говорят, что они могут быть выполнены. Операторы могут быть снабжены меткой, к которой могут отсылать операторы перехода:

```
<ОПЕРАТОР> ::= <НЕПОМЕЧЕННЫЙ ОПЕРАТОР> !
                 <МЕТКА>:<НЕПОМЕЧЕННЫЙ
                 ОПЕРАТОР>
<НЕПОМЕЧЕННЫЙ ОПЕРАТОР> ::= <ПРОСТОЙ
                             ОПЕРАТОР> !
                           <СТРУКТУРНЫЙ
                           ОПЕРАТОР>
<МЕТКА> ::= <ЦЕЛОЕ БЕЗ ЗНАКА>
```

6.1. Простые операторы

Простые операторы — это операторы, никакая часть которых не содержит в себе других операторов.

```
<ПРОСТОЙ ОПЕРАТОР> ::= <ОПЕРАТОР ПРИСВАИ-
                       ВАНИЯ> !
                       <ОПЕРАТОР ПРОЦЕДУ-
                       РЫ> !
                       <ОПЕРАТОР ПЕРЕХО-
                       ДА> !
                       <ПУСТОЙ ОПЕРАТОР>
```

6.1.1. Операторы присваивания

Оператор присваивания служит для замены текущего значения переменной новым значением, задаваемым некоторым выражением:

<ОПЕРАТОР ПРИСВАИВАНИЯ> ::= <ПЕРЕМЕННАЯ> :=<ВЫРАЖЕНИЕ> |
<ИДЕНТИФИКАТОР ФУНКЦИИ> :=<ВЫРАЖЕНИЕ>

Переменная (соответственно функция) и выражение должно быть одного типа; допускаются следующие исключения:

1) переменная имеет вещественный тип, а выражение — целый или его отрезок;

2) тип выражения является отрезком типа соответствующей переменной или наоборот.

Примеры:

X:=T+Z
P:=(1<=I) AND (I<100)
I:=SQR(K) — (I*J)
Оттенок:=[ГОЛУБОЙ,SUCC(Ц)]

6.1.2. Операторы процедур

Оператор процедуры служит для выполнения процедуры, обозначенной идентификатором процедуры. Оператор процедуры может содержать список фактических параметров, подставляемых на место соответствующих им формальных параметров, определенных в описании процедуры. Соответствие устанавливается по позициям параметров в списках формальных и фактических параметров. Существует четыре вида параметров: так называемые параметры-значения, переменные, процедуры (соответствующий фактический параметр должен быть идентификатором процедуры) и параметры-функции (фактические параметры — идентификатор функции).

Стандартные процедуры и функции (оформленные в языке) не могут передаваться в качестве параметров. Это ограничение можно обойти путем определения второй процедуры, которая просто вызывает стандартную, например:

```
FUNCTION SINE(X:REAL): REAL;
BEGIN SINE:=SIN(X) END;
```

В случае параметров-значений фактический параметр должен быть выражением (в частности переменной). Если формальный параметр является локальной переменной вызываемой процедуры, то текущее значение выражения присваивается этой переменной в качестве ее начального значения. В случае параметров-переменных фактическим параметром должна быть переменная. Соответствующий формальный параметр представляет эту фактическую переменную во все время

выполнения данной процедуры. Если эта переменная является компонентой некоторого массива, то ее индекс вычисляется при вызове процедуры. Параметр-переменную следует использовать во всех случаях, когда параметр служит для представления результата процедуры:

<ОПЕРАТОР ПРОЦЕДУРЫ>::= <ИДЕНТИФИКАТОР ПРОЦЕДУРЫ> !
<ИДЕНТИФИКАТОР ПРОЦЕДУРЫ>(<ФАКТИЧЕСКИЙ ПАРАМЕТР>
(*, <ФАКТИЧЕСКИЙ ПАРАМЕТР>*))
<ИДЕНТИФИКАТОР ПРОЦЕДУРЫ>::=<ИДЕНТИФИКАТОР>
<ФАКТИЧЕСКИЙ ПАРАМЕТР>::=<ВЫРАЖЕНИЕ> !
<ПЕРЕМЕННАЯ> !
<ИДЕНТИФИКАТОР ПРОЦЕДУРЫ> ! <ИДЕНТИФИКАТОР ФУНКЦИИ>

Примеры:

TRANSPOSE(A,N,M)
BTSECT(FST,—1.0,+1.0,X)

6.1.3. Операторы перехода

Оператор перехода служит для указания, что дальше должна выполняться та часть программы, перед которой помещена соответствующая метка:

<ОПЕРАТОР ПЕРЕХОДА>::=GOTO <МЕТКА>

На использование меток накладываются следующие ограничения:

1) областью действия метки является описание процедуры, в пределах которой она определена. Таким образом, извне нельзя войти внутрь процедуры;

2) если оператор перехода передает управление вовне процедуры, то его метка должна быть специфицирована описанием метки в заголовке процедуры, в которой эта метка определена.

6.1.4. Пустой оператор

Пустой оператор никак не обозначается и не вызывает никаких действий.

<ПУСТОЙ ОПЕРАТОР>::=

6.2. Структурные операторы

Структурные операторы являются конструкциями, составленными из других операторов, которые должны быть выполнены или последовательно (составной оператор), или повторно (циклы).

<СТРУКТУРНЫЙ ОПЕРАТОР>::=<СОСТАВНОЙ ОПЕРАТОР> !
<УСЛОВНЫЙ ОПЕРАТОР> ! <ОПЕРАТОР ЦИКЛА>
<ОПЕРАТОР НАД ЗАПИСЬЮ>

6.2.1. Составные операторы

Составной оператор предписывает выполнение составляющих его операторов в порядке их написания. Символы BEGIN и END действуют как операторные скобки.

<СОСТАВНОЙ ОПЕРАТОР>::=BEGIN<ОПЕРАТОР>(*;<ОПЕРАТОР>*)END

Пример:

```
BEGIN Z:=X; X:=Y; Y:=Z END
```

6.2.2. Условные операторы

Условный оператор назначает к выполнению один из составляющих его операторов:

<УСЛОВНЫЙ ОПЕРАТОР>::=<ОПЕРАТОР «IF»>!
<ОПЕРАТОР «CASE»>

6.2.2.1. Оператор «IF»

Оператор «IF» указывает, что некоторый оператор должен выполняться, если только некоторое условие (логическое выражение) истинно (TRUE). Если же оно имеет значение FALSE, то либо не выполняется никакой оператор, либо выполняется оператор, следующий за символом ELSE:

```
<ОПЕРАТОР "IF">::=IF<ВЫРАЖЕНИЕ>THEN<ОПЕРАТОР> '
                  IF<ВЫРАЖЕНИЕ>THEN<ОПЕРАТОР>ELSE<ОПЕРАТОР>
```

Выражение между символами IF и THEN должно иметь логический тип.

ПРИМЕЧАНИЕ. Синтаксическая неоднозначность, возникающая в конструкции:

IF <ВЫРАЖЕНИЕ1> THEN IF <ВЫРАЖЕНИЕ2> THEN <ОПЕРАТОР1>
ELSE <ОПЕРАТОР2>

разрешается толкованием этой конструкции как эквивалента для:

```
IF <ВЫРАЖЕНИЕ1>
THEN BEGIN IF <ВЫРАЖЕНИЕ2>
                THEN<ОПЕРАТОР1>
                ELSE<ОПЕРАТОР2>
     END
```

Примеры:

```
IF X=1.5 THEN Z:=X+Y ELSE Z:=1.5
IF P1<>NIL THEN P1:=P1^.Отец
```

6.2.2.2. Оператор «CASE»

Оператор «CASE» состоит из выражения (селектора) и списка операторов, каждый из которых помечен константой селекторного типа. Оператор «CASE» указывает, что должен

быть выполнен один оператор: тот, метка которого равна текущему значению селектора. Если ни одна из меток в операторе CASE не совпадает со значением «селекторной» переменной, то выполняется последовательность операторов, следующая за символом ELSE. Использование «;» недопустимо между ELSE и END в операторе CASE:

```
<ОПЕРАТОР "CASE">::=CASE<ВЫРАЖЕНИЕ> OF
                <АЛЬТЕРНАТИВА> {;<АЛЬТЕРНАТИВА>}
               [ELSE <ОПЕРАТОР>] END
<АЛЬТЕРНАТИВА>::=<СПИСОК МЕТОК ВЫБОРА>:<ОПЕРАТОР>
<СПИСОК МЕТОК ВЫБОРА>::=<МЕТКА ВЫБОРА {,МЕТКА ВЫБОРА}>
    Примеры:
CASE операция OF                    CASE I OF
     плюс:X:=X+Y;                         1:X:=SIN(X);
     минус:X:=X-Y;                        2:X:=COS(X);
     умнож:X:=X*Y;                        3:X:=EXP(X);
END                                       4:X:=LN(X);
                                    END

        REPEAT
                READLN(CH);
                CASE CH OF
                        A :APPEND;
                        D':DELETE;
                        I':INSERT;
                        N':NEWFILE;
                        Q':;
                       ELSE
                        WRITELN('"',CH,'" IS NOT LEGAL')
                END;
        UNTIL CH = 'Q';
```

6.2.3. Циклы

Операторы цикла предписывают многократное выполнение некоторых операторов. Если число повторений известно заранее, т. е. до начала выполнения цикла, то подходящей конструкцией является оператор «FOR»; в остальных случаях следует использовать оператор «WHILE» или оператор «REPEAT».

<ОПЕРАТОР ЦИКЛА>::=<ОПЕРАТОР «WHILE»> !
<ОПЕРАТОР «REPEAT»>!
<ОПЕРАТОР «FOR»>

6.2.3.1. Оператор «WHILE»

<ОПЕРАТОР «WHILE»>::=WHILE<ВЫРАЖЕНИЕ>
DO<ОПЕРАТОР>

Выражение, управляющее повторениями, должно иметь логический тип. Оператор повторно используется до тех пор, пока значением выражения не станет «FALSE». Если значение выражения с самого начала «FALSE», то оператор не выполняется.

Оператор «WHILE»:
WHILE E DO S
эквивалентен конструкции:

```
IF E THEN
   BEGIN S;
      WHILE E DO S
   END
```

Примеры:

```
WHILE A(I) <> X DO I:=I+1
WHILE I>0 DO
BEGIN IF ODD(I) THEN Z:=Z*X;
      I:=IDIV2;
      X:=SQR(X)
END
WHILE NOT EOF(F) DO
BEGIN P(F^); GET (F)
END
```

6.2.3.2. Оператор «REPEAT»

<ОПЕРАТОР «REPEAT»>::=
REPEAT<ОПЕРАТОР>(*,<ОПЕРАТОР>*)UNTIL <ВЫРАЖЕНИЕ>

Выражение, управляющее повторениями, должно иметь логический тип. Последовательность операторов между REPEAT и UNTIL выполняется повторно (и не менее одного раза до тех пор, пока значением соответствующего выражения не станет TRUE.

Оператор «REPEAT»:
REPEAT S UNTIL E
эквивалентен оператору:

```
BEGIN S;
IF NOT E THEN
   REPEAT S UNTIL E
END
```

Примеры:

```
REPEAT K:=I MOD J;
       I:=J;
       J:=K;
UNTIL J=0
REPEAT P(F);GET(F)
UNTIL EOF(F)
```

6.2.3.3. Оператор «FOR»

Оператор «FOR» предписывает повторное выполнение соответствующего оператора по мере присваивания переменной,

называемой параметром цикла, некоторой последовательности значений.

<ОПЕРАТОР «FOR»>::=FOR<ПАРАМЕТР ЦИКЛА>:=<СПИСОК ЦИКЛА>DO <ОПЕРАТОР>
<СПИСОК ЦИКЛА>::=<ПЕРВОЕ ЗНАЧЕНИЕ> TO <ПОСЛЕДНЕЕ ЗНАЧЕНИЕ> | <ПЕРВОЕ ЗНАЧЕНИЕ> DOWNTO <ПОСЛЕДНЕЕ ЗНАЧЕНИЕ>
<ПАРАМЕТР ЦИКЛА>::=<ИДЕНТИФИКАТОР>
<ПЕРВОЕ ЗНАЧЕНИЕ>::=<ВЫРАЖЕНИЕ>
<ПОСЛЕДНЕЕ ЗНАЧЕНИЕ>::=<ВЫРАЖЕНИЕ>

Параметры цикла, первое значение и последнее значение должны быть одного и того же скалярного типа (или его отрезка) и не должны меняться повторяемым в цикле оператором.

Цикл с параметром вида:

FOR V:= E1 TO E2 DO S

эквивалентен последовательности операторов:

V:=E1; S; V:=SUCC(V); S;...; V:=E2; S;

а цикл с параметром вида:

FOR V:=E1 DOWNTO E2 DO S

эквивалентен последовательности операторов:

V:=E1; S; V:=PRED(V); S;...; V:=E2; S;

ПРИМЕЧАНИЕ. Последнее значение параметра остается неопределенным после завершения цикла.

Примеры:

```
FOR I:=2 TO 100 DO IF A[I]>MAX THEN MAX:=A(I)
FOR I:=1 TO N DO
FOR J:=1 TO N DO
BEGIN X:=0;
   FOR K:=1 TO N DO X:=X+A(I,J)*B(K,J)
   C[I,J]:=X
END
FOR Ц:=КРАСН TO ГОЛУБОЙ DO Q(Ц)
```

6.2.3.4. Оператор EXIT

Оператор EXIT используется для выхода из тела одного из итерационных операторов (WHILE, REPEAT, FOR). Например, поиск по таблице:

```
FOUND := FALSE;
FOR   I := 1 TO TABLESIZE DO
          IF TABLE[I] = KEY
             THEN BEGIN FOUND:=TRUE; EXIT END;
```

6.2.4. Оператор над записями «WITH»

<ОПЕРАТОР «WITH»> ::= WITH<СПИСОК ПЕРЕМЕННЫХ—ЗАПИСЕЙ>
DO <ОПЕРАТОР>

<СПИСОК ПЕРЕМЕННЫХ—ЗАПИСЕЙ> ::= <ПЕРЕМЕННАЯ ЗАПИСЬ>
(*,<ПЕРЕМЕННАЯ ЗАПИСЬ>*)

В пределах внутреннего оператора в операторе «WITH» компоненты (поля) переменной-записи, заданной в его заголовке, могут обозначаться одним только своим идентификатором поля, т. е. без предшествующего им написания полной переменной записи. Фактически операторы над записями расширяют область определенности идентификаторов поля заданной переменной-записи, так что идентификаторы поля могут выступать, как идентификаторы переменных.

Пример:

```
WITH DATA DO
IF MEC = 12 THEN
BEGIN MEC := 1; ГОД := ГОД+1
END
ELSE MEC := MEC +1
```

Этот оператор эквивалентен следующему:

```
IF DATA.MEC = 12 THEN
BEGIN DATA.MEC := 1; DATA.ГОД:=DATA.ГОД +1
END
ELSE DATA.MEC:= Data.Mec + 1
```

В операторах над записями не должно содержаться присваиваний никакой составляющей списка переменных записей.

Оператор «WITH» может иметь уровень вложенности не выше 4-х; если сложные выражения используются внутри этого оператора, то уровень вложенности может быть не более 2-х или 3-х.

7. ОПИСАНИЕ ПРОЦЕДУР

Назначение описаний процедур — определить части программы и связать с ними идентификаторы, с тем чтобы эти части могли быть активированы операторами процедур. Описание процедуры состоит из следующих частей, любая из которых, за исключением первой и последней, может быть пустой:

<ОПИСАНИЕ ПРОЦЕДУРЫ>::=<ЗАГОЛОВОК ПРОЦЕДУРЫ>
<РАЗДЕЛ ОПИСАНИЙ МЕТОК>/

<РАЗДЕЛ ОПРЕДЕЛЕНИЙ КОНСТАНТ>/
<РАЗДЕЛ ОПРЕДЕЛЕНИЙ ТИПОВ>/
<РАЗДЕЛ ОПИСАНИЙ ПЕРЕМЕННЫХ>/
<РАЗДЕЛ ОПИСАНИЙ ПРОЦЕДУР И ФУНКЦИЙ>/
<РАЗДЕЛ ОПЕРАТОРОВ>

Заголовок процедуры задает идентификатор, именующий процедуру, и идентификаторы формальных параметров (если они есть). Параметрами могут быть параметры-переменные, -значения, -процедуры или параметры-функции.

```
<ЗАГОЛОВОК ПРОЦЕДУРЫ>::=PROCEDURE<ИДЕНТИФИКАТОР>;!
  PROCEDURE<ИДЕНТИФИКАТОР>(<СЕКЦИЯ ФОРМАЛЬНЫХ ПАРАМЕТРОВ>
            (*;<СЕКЦИЯ ФОРМАЛЬНЫХ ПАРАМЕТРОВ*));
<СЕКЦИЯ ФОРМАЛЬНЫХ ПАРАМЕТРОВ>::=<ГРУППА ПАРАМЕТРОВ>!
          VAR <ГРУППА ПАРАМЕТРОВ>!
          FUNCTION <ГРУППА ПАРАМЕТРОВ> !
          PROCEDURE <ИДЕНТИФИКАТОР> (*;<ИДЕНТИФИКАТОР>*)
 ГРУППА ПАРАМЕТРОВ>::=
            <ИДЕНТИФИКАТОР>(*,<ИДЕНТИФИКАТОР>*)
            :<ИДЕНТИФИКАТОР ТИПА>
```

Группы параметров без предшествующего спецификатора считаются параметрами-значениями.

Раздел описаний меток специфицирует все метки, локализованные в данной процедуре:

```
<РАЗДЕЛ ОПИСАНИЙ МЕТОК>::=<ПУСТО>!
                    LABEL <МЕТКА> (*,<МЕТКА>*);
<РАЗДЕЛ ОПРЕДЕЛЕНИЙ КОНСТАНТ>::=<ПУСТО>!
                               CONST<ОПРЕДЕЛЕНИЕ КОНСТАНТЫ>
                                (*;<ОПРЕДЕЛЕНИЕ КОНСТАНТЫ>*);
<РАЗДЕЛ ОПРЕДЕЛЕНИЙ ТИПОВ>::=<ПУСТО>!
          TYPE<ОПРЕДЕЛЕНИЕ ТИПА>(*;<ОПРЕДЕЛЕНИЕ ТИПА>*);
<РАЗДЕЛ ОПИСАНИЙ ПЕРЕМЕННЫХ>::= <ПУСТО>!
     VAR<ОПИСАНИЕ ПЕРЕМЕННОЙ> (*;<ОПИСАНИЕ ПЕРЕМЕННОЙ>*);
<РАЗДЕЛ ОПИСАНИЙ ПРОЦЕДУР И ФУНКЦИЙ>::=
                        (*<ОПИСАНИЕ ПРОЦЕДУРЫ ИЛИ ФУНКЦИИ>;*)
<ОПИСАНИЕ ПРОЦЕДУРЫ ИЛИ ФУНКЦИИ>::=<ОПИСАНИЕ ПРОЦЕДУРЫ>!
```

Раздел операторов задает алгоритмические действия, которые должны быть выполнены после обращения к процедуре с помощью оператора процедуры:

<РАЗДЕЛ ОПЕРАТОРОВ>::=<СОСТАВНОЙ ОПЕРАТОР>

Все идентификаторы, введенные в разделе формальных параметров, разделе определений констант, типов, разделе описаний переменных и процедур или функций, локальны в данном описании процедуры, которое называется областью определенности этих идентификаторов. Они не доступны вне своей области определенности. Значения локальных переменных в момент входа в тело не определены. Использование в теле процедуры ее собственного имени подразумевает рекурсивное выполнение этой процедуры.

Синтаксически вложенность процедур не может быть более 10-ти. При выполнении на вложенность не накладывается ограничений, но большая вложенность при рекурсии может привести к исчерпыванию рабочей памяти.

Примеры описаний процедур:

```
PROCEDURE BISECT(FUNCTION F:REAL; A,B:REAL; VAR Z:REAL);
  VAR M :REAL;
 BEGIN (* предполагается что F(A)<0 И F(B)>0 *)
 WHILE ABS(A-B) > 1E-10*ABS(A) DO
 BEGIN M := (A+B)/2.0;
       IF F(M)<0 THEN A := M ELSE B :=  M
 END;
 Z:=M
 END
```

Процедура (функция) может быть вызвана раньше, чем она определена, если есть ссылка «Вперед» — FORWARD. Список параметров и тип окончательного результата записывается только в FORWARD ссылке, например:

```
PROCEDURE Q(X:T); FORWARD;
PROCEDURE P(Y:T);
    BEGIN Q(A) END;
PROCEDURE Q; (*параметры не повторяются*)
    BEGIN P(B) END;
BEGIN P(A);Q(B)
END.
```

7.1. Стандартные процедуры

Предполагается, что стандартные процедуры заранее описаны в любой реализации языка ПАСКАЛЬ. Любая реализация может дополнительно вводить такие заранее описанные процедуры. Поскольку они, как и все стандартные величины, считаются описанными в области определенности, охватывающей всю программу, написанную на ПАСКАЛЕ, то появление в этой программе описаний, переопределяющих

эти идентификаторы, не приводит к конфликтам. Стандартные процедуры перечислены и объяснены ниже.

7.1.1. Процедуры работы с файлами

PUT (F) — присоединяет значение буферной переменной F∧ к файлу «F». Определено только в случае, когда перед исполнением значение предиката EOF(F) есть TRUE; EOF(F) сохраняет значение TRUE, значение же F∧ становится неопределенным;

GET (F) — смещает текущую позицию файла (головку чтения-записи) на следующую компоненту и приписывает значение этой компоненты буферной переменной F∧. Если следующей компоненты не существует, то значением EOF(F) становится TRUE, а значение F∧ не определено. Результат GET(F) определен только тогда, когда перед исполнением соответствующего вызова EOF (F) = FALSE;

RESET(F) — возвращает текущую позицию файла в начало файла и приписывает буферной переменной F∧ значение первого элемента «F». Значением EOF(F) становится FALSE, если «F» не пуст; в противном случае F∧ не определено, а значением EOF(F) остается TRUE;

REWRITE(P) — уничтожает текущее значение «F», так что может начаться заполнение нового файла; EOF(F) принимает значение TRUE;

BREAK(F) — **вызывает немедленный вывод, возможно, еще** неполного буфера (блока) в файл. Это надо использовать при работе с терминалом, например, для вывода подсказки или сообщения. Следует отметить, что данный ПАСКАЛЬ не буферизует работу с драйверами, о которых известно, что они являются «интерактивными», такими как выходной файл по умолчанию.

Пример:

```
VAR F:TEXT;
BEGIN
   REWRITE(F,'TT:');
   WRITELN(F,'введите командную строку');
   BREAK(F;)          (* вывести данные на TT: *)
END.
```

CLOSE(F)

— пересылает на внешнее устройство не до конца заполненный буфер, устраняет связь с внешним файлом и освобождает буферную память для использования ее в других целях. Процедуры «RESET» или «REWRITE» должны предшествовать всякой работе с файловой переменной.

ПРИМЕЧАНИЕ. Процедура «CLOSE» всегда должна использоваться в конце работы с файлом, созданным программой (выходным файлом), иначе можно потерять последний буфер, который не будет записан в файл. Закрывание файлов для ввода не является обязательным, но позволяет освободить буферную память.

7.1.2. Дополнительные аргументы для «RESET» и «REWRITE»

В данной реализации расширены стандартные процедуры «RESET» и «REWRITE» тремя дополнительными параметрами для указания связи между внутренними файловыми переменными и файлами на внешних устройствах

```
PROCEDURE RESET (F:FILE;NAME,DEFEXT:STRING;
                 VAR LEN:INTEGER)
```

Процедура RESET связывает файловую переменную с существующим внешним файлом, а также устанавливает указатель файла на первый элемент (запись). «F» всегда должна присутствовать и быть файловой переменной. Имя файла «NAME» и расширение имени файла должны иметь тип «ARRAY OF CHAR».

Имя файла должно соответствовать принятому в ФОДОС-2 стандартному обозначению и может включать имя и номер устройства, а также тип файла. Если во втором параметре отсутствует тип файла, то по умолчанию будет использован тип «DAT».

Переменная «LEN» получит значение, равное числу блоков по 512 байт в файле, либо — 1, если файл не найден. Следует учесть, что если этот параметр не использован, то отсутствие файла приведет к фатальной ошибке программы.

```
PROCEDURE REWRITE(F:FILE; NAME,DEFEXT:STRING;
                  VAR LEN:INTEGER)
```

Процедура «REWRITE» создает новый файл на внешнем устройстве. Параметры «NAME» и «DEFEXT» имеют то же значение (смысл), что и для процедуры «RESET». Параметр «LEN» указывает размер создаваемого файла в блоках по 512 байт.

Пример:

```
VAR NAME: ARRAY[1..20] OF CHAR;
    INF,OUTF: TEXT;
    (* определено в системе как «FILE OF CHAR *)
    LEN: INTEGER;
BEGIN
    (* связать OUTF с устройством печати *)
    REWRITE(OUTF,'LP:');
    REPEAT
        WRITE('FILENAME: ');
        (* запросить имя файла с терминала *)
        READLN(NAME);
        (*связать INF с существующим файлом *)
        RESET(INF,NAME,'PAS',LEN)
    (* повторять, пока файл не будет открыт *)
    UNTIL LEN<>-1;
END.
```

7.1.3. Процедура динамического размещения

NEW(P) — размещает новую переменную V и приписывает ссылку на V переменной — указателю P;

DISPOSE(P)
— освобождает память, занимаемую переменной P^;

Если тип V — это тип записи с вариантами, то в вызовах «NEW» и «DISPOSE» выделяется и освобождается максимальная область памяти, необходимая для записи.

NEW(P,T1,...,TN)
— может быть использован для размещения конкретного варианта (в случае записи с вариантами) со значениями поля признака T1,...,TN. Это не означает назначения поля признака.

DISPOSE(P,T1,...,TN)
— освобождает память, занимаемую вариантом записи со значениями поля признака T1,...,TN. Величина поля признака должна быть идентична той, которая использовалась для размещения переменной.

7.1.4. Файлы прямого доступа

Предлагаемая реализация ПАСКАЛЯ содержит несколько определений процедур, которые после трансляции вместе с программой обеспечивают прямой доступ к элементам файловой переменной.

```
PROCEDURE SEEK(VAR F:RANFIL; VAR
                    DATA: USERTYPE; N:INTEGER)
```

Процедура «SEEK» возвращает N-ый элемент файла «F» в переменной «DATA». Устанавливается EOF(F), если указанный элемент не существует.

```
PROCEDURE DEPOSIT(VAR F:RANFIL; VAR
                  DATA:USERTYPE; N:INTEGER)
```

Процедура «DEPOSIT» заносит значение переменной «DATA» в файл «F» в качестве N-ого элемента (записи). При необходимости файл расширяется.

```
PROCEDURE CLOSERANDOMFILE (VAR F:RANFIL)
```

Данная процедура должна использоваться для закрытия файла прямого доступа (вместо «CLOSE»).

```
TYPE USERTYPE=ARRAY[0..255] OF INTEGER;
TYPE RANFIL=FILE OF USERTYPE;
```

Файл F открывается процедурой RESET, а создается процедурой REWRITE. При создании файла необходимо записать хотя бы один элемент.

7.2. Процедуры ввода/вывода

Стандартный ПАСКАЛЬ содержит определения двух файловых переменных «INPUT» и «OUTPUT», которые являются файлами по умолчанию для «READ» и «WRITE». Поэтому «READ(I)» эквивалентно «READ(INPUT,I)». В ПАСКАЛЕ, реализованном для ДВК, переменные «INPUT» и «OUTPUT» предварительно не определены. В качестве файла по умолчанию используется терминал для процедур «READ» и «WRITE».

7.2.1. Процедура READ

Для процедуры READ действуют следующие обозначения: F — текстовый файл; V1, ..., VN — параметры процедуры.

Стандартная процедура READ допускает произвольное число параметров, так что:

READ(F,V1,...,VN) означает READ(F,V1);...;READ(F,VN)

Параметры могут быть литерного, целого, вещественного типа и типа массив. В первом случае считывается только очередная литера, в двух следующих — считывается последовательность литер, представляющих собой целое или вещественное число в соответствии с синтаксисом языка ПАСКАЛЬ (последовательные числа должны отделяться пробелами или символами конца строк). В последнем случае знаки будут считываться, начиная с текущего положения указателя в файле, пока не встретится пробел, запятая или конец строки. При этом строка будет выравнена слева и дополнена пробелами (если строка меньше массива), или усечена справа (если строка больше массива).

7.2.2. Процедура READLN

Стандартная процедура READLN пропускает во входном файле символы до тех пор, пока не встретится конец строки.

READLN(F) работает также как:

```
WHILE NOT EOLN(F) DO GET(F);
GET(F)
```

READLN(F,V1,...,VN) означает
READ(F,V1,...,VN);READLN(F)

7.2.3. Процедура WRITE

Для процедуры WRITE действуют следующие обозначения: F — текстовый файл; P1,...,PN — параметры процедуры; E — выражение; M,N — выражения типа INTEGER (форматы вывода).

Стандартная процедура WRITE допускает произвольное число параметров, так что:

WRITE(F,P1,...,PN) означает
WRITE(F,P1);...;WRITE(F,PN)

Каждый параметр принимает одну из следующих форм:

E E:M E:M:N

E может быть литерного, целого, вещественного или логического типа либо типа массив. Значения преобразуются в последовательность из M литер. Если величина E для своего представления требует меньше символов, чем M, то добавляются пробелы, так что выводится точно M символов. Если M меньше, чем необходимо для вывода величины E, то выводится столько символов, сколько требует E для своего представления (M в этом случае игнорируется). N применимо только к параметрам, имеющим вещественное значение, и задает число цифр, печатаемых после десятичной точки. Если N опущено, то число печатается в форме с плавающей точкой. Если M опущено, то по умолчанию для разных типов назначаются следующие значения M:

литерный	1
целый	13
вещественный	13
логический	6

Процедура «WRITE» будет выводить целые числа в восьмеричном виде, если выражение для формата вывода будет отрицательным, например:

```
(* Вывести «I» в восьм.виде — 5 знаков *)
WRITE(I:-5);
```

Пример:

Пусть K=135, N=4, X=72.83, B=TRUE, C='A', тогда

WRITE(K+K:N,X:12,X:6:1,'' A'',C,B)
выводит последовательность литер
270 7.2830E+01 72.8 AA TRUE

7.2.4. Процедура WRITELN

Стандартная процедура WRITELN(F) используется, чтобы окончить выходную строку и начать новую. Она добавляет маркер конца строки к файлу F.
WRITELN(F,P1,...,PN) означает
WRITE(F,P1,...,PN);WRITELN(F)

Имя файла в процедурах READ,READLN,WRITE, WRITELN можно опустить. По умолчанию в качестве файла используется терминал.

7.2.5. Ввод/вывод с терминала

Стандартный ПАСКАЛЬ требует, чтобы первый элемент файла был получен в программе непосредственно после работы процедуры «RESET» (Буферная переменная F^ получает это значение немедленно). Это приводит к сложностям, если в качестве файла используется терминал. Например, если входной файл по умолчанию является терминалом, то стандарт требует ввода первого знака или строки перед запуском программы на выполнение.

В данной реализации ПАСКАЛЯ используется следующее решение этой проблемы. Когда выполняется процедура «RESET» для интерактивного терминала, то буферная переменная получает значение пробела и «EOLN». При этом реальной операции ввода не происходит. Каждый следующий запрос «READ» после этого дожидается требуемых данных.

Это позволяет решить большинство из проблем с терминалом удовлетворительным образом, но следует понимать, что это порождает некоторые другие трудности, например:

```
VAR LINE: ARRAY[1..72] OF CHAR;
    COUNT: INTEGER;
BEGIN
    COUNT:=0;
    WHILE NOT EOLN DO BEGIN
        COUNT:=COUNT+1;
        READ(LINE[COUNT])
        END;
    READLN
END.
```

В приведенном примере показывается стандартная схема для чтения строки знаков с терминала. Когда этот способ применяется к интерактивному файлу, то невозможно отличить пустую строку и строку, содержащую один пробел. Это

связано с тем, что «EOLN» не может быть установлен, пока не встретится знак конца строки для завершения запроса «READ».

7.3. Конец файла

В ФОДОС-2 файл представляется последовательностью блоков по 512 байт каждый. При этом в конце файла отсутствует признак конца файла. Поэтому функция «EOF» не всегда обеспечивает правильную индикацию конца файла. Решить эту проблему можно, используя счетчик записей в файле или путем использования символьной записи, которая будет выступать в качестве признака конца.

Данная проблема не относится к текстовым файлам, которые в качестве признака конца используют знак <СУ/Z>.

8. ОПИСАНИЯ ФУНКЦИЙ

Описания функций служат для определения частей программы, вычисляющих скалярное или указательное значение. Обращение к функции осуществляется с помощью указателей функции, являющихся частями выражений. Описание функции состоит из следующих семи частей, каждая из которых, за исключением первой и последней, может быть пуста.

```
<ОПИСАНИЕ ФУНКЦИИ>::= <ЗАГОЛОВОК ФУНКЦИИ>
                       <РАЗДЕЛ МЕТОК>/
                       <РАЗДЕЛ ОПРЕДЕЛЕНИЙ
                         КОНСТАНТ>/
                       <РАЗДЕЛ ОПРЕДЕЛЕНИЙ ТИПОВ>/
                       <РАЗДЕЛ ОПИСАНИЙ
                         ПЕРЕМЕННЫХ>/
                       <РАЗДЕЛ ОПИСАНИЙ ПРОЦЕДУР
                         И ФУНКЦИЙ>/
                       <РАЗДЕЛ ОПЕРАТОРОВ>
```

Заголовок функции задает идентификатор, обозначающий данную функцию, формальные параметры функции и ее тип:

```
<ЗАГОЛОВОК ФУНКЦИИ>::=FUNCTION<ИДЕНТИФИКАТОР>:
                        <ТИП РЕЗУЛЬТАТА>;!
                      FUNCTION<ИДЕНТИФИКАТОР>
                      (<СЕКЦИЯ ФОРМАЛЬНЫХ
                         ПАРАМЕТРОВ>
                      (*;СЕКЦИЯ ФОРМАЛЬНЫХ
                         ПАРАМЕТРОВ>*)):
                       <ТИП РЕЗУЛЬТАТА>;
<ТИП РЕЗУЛЬТАТА>::=<ИДЕНТИФИКАТОР ТИПА>
```

Тип функции должен быть скалярным, отрезком типа или типом указателя. В пределах описания функции должен со-

держаться хотя бы один оператор присваивания, присваивающий значение идентификатору функции. Вхождение идентификатора функции в указатель функции в пределах ее описания подразумевает рекурсивное исполнение этой функции.

Вызов функции может быть раньше ее описания, если есть FORWARD ссылка (см. п. 7).

Примеры:

```
FUNCTION SQRT (X : REAL) : REAL;
  VAR X0,X1 : REAL;
BEGIN X1:=X; (*X > 1, МЕТОД НЬЮТОНА*)
  REPEAT X0:=X1; X1:=(X0 + X/X0) * 0.5
  UNTIL ABS(X1 - X0) < EPS*X1;
  SQRT:=X0
END
FUNCTION MAX(A : VECTOR; N : INTEGER) : REAL;
        VAR X : REAL; I : INTEGER;
BEGIN X :=A[1];
        FOR I :=2 TO N DO
        BEGIN (* X=MAX(A[1]...A[I-1]) *)
        IF X<A[I] THEN X := A[I]
        END;
        (* X = MAX(A[1]...A[N]) *)
        MAX:=X
END
FUNCTION степень (X:REAL; Y:INTEGER) :REAL; (* Y>=0 *)
        VAR W,Z : REAL;I : INTEGER;
BEGIN W :=X; Z := 1; I := Y;
        WHILE I <> 0 DO
        BEGIN (* Z * W**I = X**Y *)
        IF ODD(I) THEN Z := Z*W;
        I := I DIV 2; W := SQR(W)
END (* Z = X**Y *);
СТЕПЕНЬ := Z
END
```

8.1. Стандартные функции

Предполагается, что стандартные функции заранее описаны в любой реализации языка ПАСКАЛЬ. Любая реализация может вводить дополнительно такие заранее описанные функции.

Ниже следует список стандартных функций с пояснениями.

8.1.1. Арифметические функции

ABS(X) — вычисляет абсолютную величину X. Тип X должен быть вещественным или целым, тип результата совпадает с типом X;

SQR(X) — вычисляет X во 2-й степени. Тип X должен быть вещественным или целым, тип результата совпадает с типом X;

SIN(X)
COS(X)
EXP(X)
LN(X)
SQRT(X)
ARCTAN(X)
— Тип X должен быть вещественным или целым; результат имеет вещественный тип.

EXP10(X)
LOG(X)
— аналогично EXP(X) и LN(X), но по основанию 10;

8.1.2. Предикаты

ODD(X) — тип X должен быть целым, результат равен TRUE, если X — число нечетное, т. е. «X MOD 2 = 1»;

EOF(F) — указывает, находится ли файл «F» в состоянии «конец файла».

8.1.3. Функции преобразования

TRUNC(X)
— X должен быть вещественного типа, результат имеет тип целый (INTEGER) и получается при отбрасывании дробной части X, т. е. если X>=0, то X —1<TRUNC(X)<=X; если X<0, то X+1>TRUNC(X)>=X;

ROUND(X)
— X должен быть вещественного типа, результат (целого типа) — целое число, ближайшее к X, т. е.: ROUND(X)=TRUNC(X+0.5) для X>=0; и TRUNC(X—0.5) для X<0;

ORD(X) — X должен быть литерного типа, результат (целого типа) — это порядковый номер литеры X в заданной последовательности литер. Результат ORD(X) будет в диапазоне (—128..127).

CHR(X) — X должен быть целого типа, результат (литерного типа) — это литера, порядковый номер которой равен X.

8.1.4. Функция «TIME»

Функция «TIME» возвращает значение типа REAL, соответствующее времени дня в часах после полуночи. Точность этой функции не менее одной секунды, например:

```
   PROCEDURE WRITETIME;
VAR R:REAL;
        HOURS,MINUTES:INTEGER;
         AMPM: ARRAY [1..2] OF CHAR;
BEGIN
   R:=TIME;
   HOURS:=TRUNC(TIME);
```

```
    MINUTES:=TRUNC( (R—HOURS)*60.0);
    IF HOURS > 12
          THEN AMPM:='PM'
          ELSE IF (HOURS = 12) AND (MINUTES = 0)
            THEN AMPM:='M '
            ELSE AMPM:='AM';
    WRITE('AT THE TONE THE TIME WILL BE: ');
    WRITE((HOURS+11) MOD 12 + 1 :2);
    WRITE(':', MINUTES DIV 10:1,
              MINUTES MOD 10:1, AMPM:3);
    WRITE(CHR(7));
END.
```

8.1.5. Прочие стандартные функции

SUCC(X) — X может быть любого скалярного типа или отрезком типа; результат — это значение, следующее за X (если таковое имеется);

PRED(X) — X может быть любого скалярного типа или отрезком типа, результат — это значение, предшествующее X (если оно существует).

9. ПРОГРАММЫ

Программа на языке ПАСКАЛЬ имеет вид описания процедуры без заголовка. Если заголовок все же указан, то он будет распечатываться в начале каждой страницы листинга программы. Любой параметр в заголовке программы игнорируется.

```
<ПРОГРАММА> : : = <РАЗДЕЛ ОПИСАНИЙ МЕТОК>/
                  <РАЗДЕЛ ОПРЕДЕЛЕНИЙ КОНСТАНТ>/
                  <РАЗДЕЛ ОПРЕДЕЛЕНИЙ ТИПОВ>/
                  <РАЗДЕЛ ОПИСАНИЙ ПЕРЕМЕННЫХ>/
                  <РАЗДЕЛ ОПИСАНИЙ ПЕРЕМЕННЫХ И ФУНКЦИЙ>/
                  <РАЗДЕЛ ОПЕРАТОРОВ>
```

Секция описаний может появляться несколько раз в тексте программы, но использовать имена (идентификаторы) можно только после того, как они описаны. Существует несколько способов использовать данную возможность. Например, иметь файлы с отдельными модулями. Перед трансляцией эти файлы могут объединяться вместе с основной про-

граммой. Другими словами, обеспечиваются простейшие средства работы с библиотекой модулей ПАСКАЛЯ в исходном виде.

Пример:

Файл:

```
(* ОПРЕДЕЛЕНИЕ ГРАФИЧЕСКОГО МОДУЛЯ *)
VAR ... (* ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ ФУНКЦИЙ *)
PROCEDURE (* ОПИСАНИЕ ГРАФИЧЕСКИХ ФУНКЦИЙ *)
PROCEDURE ...
(* КОНЕЦ ГРАФИЧЕСКОГО МОДУЛЯ *)
```

файл:

```
(* ФАЙЛ ОСНОВНОЙ ПРОГРАММЫ *)
VAR ... (* ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ *)
BEGIN (* ТЕКСТ ПРОГРАММЫ *) END.
```

9.1. Использование МАКРОАССЕМБЛЕРа

ПАСКАЛЬ, реализованный для ДВК, дает возможность включать в любое место программы отдельные фрагменты на языке МАКРОАССЕМБЛЕР. Части программы, написанные на МАКРОАССЕМБЛЕРе, могут использовать переменные из программы на языке ПАСКАЛЬ, хотя для этого требуется некоторое понимание процесса организации выполнения. Для включения фрагмента на АССЕМБЛЕРе используется специальный вид комментариев, например:

```
PROCEDURE EMTTRAP (N:INTEGER);
BEGIN
  (*¤C                        ; начало фрагмента на МАКРО
      MOV N(SP),—(SP)         ; параметр «N» —> в стек
      EMT 53                  ; вызвать диспетчер EMT
  *)
END   (*EMTTRAP*)
```

При использовании МАКРОАССЕМБЛЕРА следует помнить, что основанием счисления числовых констант по умолчанию является 10, а не 8.

9.2. Внешние и фортрановские подпрограммы (процедуры)

Данный ПАСКАЛЬ допускает использование «внешних» процедур и функций, которые транслируются отдельно от основной программы. Это позволяет использовать библиотеки модулей. Объявление внешней процедуры осуществляется

ключевым словом «EXTERNAL». Использование ключевого слова «FORTRAN» приведет к вызову подпрограммы в соответствии с соглашениями исполняющей системы с языка ФОРТРАН/ФОДОС-2 следует отметить, что передача параметров в подпрограмму на языке ФОРТРАН осуществляется по ссылке, а не по значению, т. е. все параметры должны быть объявлены, как VAR (переменные).

Примеры:

```
PROCEDURE ERASE; EXTERNAL
FUNCTION DIFFERENCE(VAR X,Y:REAL):REAL, FORTRAN;
```

ПАСКАЛЬ. РУКОВОДСТВО ПРОГРАММИСТА.

1. НАЗНАЧЕНИЕ И УСЛОВИЯ ПРИМЕНЕНИЯ

Система программирования ПАСКАЛЬ включена в ФОДОС-2 для расширения класса задач, решаемых операционной системой. Это связано с тем, что язык ПАСКАЛЬ удобен как для выполнения различных вычислений, так и для обработки нечисловой информации, включая экономические задачи, работу со списками и т. д.

В состав системы программирования входят удобные средства отладки программ в диалоговом режиме. Допускается возможность ведения библиотек программ, а также использование в программах на ПАСКАЛЕ подпрограмм, написанных на МАКРОАССЕМБЛЕРе и ФОРТРАНе.

Для работы с языком ПАСКАЛЬ в операционной системе ФОДОС-2 рекомендуется, чтобы в составе комплекса находилось оперативное запоминающее устройство (память) емкостью не менее 16-ти К слов. Система ПАСКАЛЬ включает следующие файлы:

PASCAL.SAV — компилятор с языка ПАСКАЛЬ на МАКРОАССЕМБЛЕР;

PASCAL.OBJ — исполняющая система, использующая только базовый набор инструкций центрального процессора;

IMP.SAV — программа обеспечивает некоторую оптимизацию текста на МАКРОАССЕМБЛЕРе, полученного в результате обработки прикладной программы транслятором ПАСКАЛЯ.

FORM.SAV — программа обеспечивает выполнение двух функций. Она осуществляет форматирование исходного текста прикладной программы на ПАСКАЛЕ для удобства восприятия ее структуры и проводит анализ корректности блочной структуры программы. В до-

полнение к этому она может создавать листинг таблицы перекрестных ссылок переменных и процедур (функций).

RANIO.OBJ — подпрограмма для выполнения операций ввода/вывода прямого доступа.

DEM.PAS — текст контрольно-демонстрационной задачи.

Дополнительно к перечисленным, необходимы файлы MACRO.SAV и LINK.SAV.

Создание и выполнение программы на языке ПАСКАЛЬ может быть выполнено с помощью следующих шагов:

— создание исходного файла с текстом программы на языке ПАСКАЛЬ;

— трансляция программы на язык МАКРОАССЕМБЛЕР;

— трансляция полученного текста на МАКРОАССЕМБЛЕРе в объектный код;

— компоновка одного модуля (или нескольких) вместе с модулями исполняющей системы ПАСКАЛЬ в программу формата загрузки;

— запуск и выполнение полученной программы.

Протокол примера действий, перечисленных выше, может иметь следующий вид:

```
.R EDIT
*EWDEM.PAS¤¤
   . . .                    ;создание файла с текстом
   . . .                    ;программы
¤EX¤¤
.R PASKAL
*DEM=DEM
.R MACRO
*DEM=DEM
*^C
.R LINK
*DEM=DEM,PASCAL  ;или DEM=DEM,SY:PASCAL, если
                 ;PASCAL.OBJ находится на SY:, а
                 ;не на DK:
*^C
.RUN DEM
   . . .                    ;выполнение полученной программы
```

В следующих разделах документа приведенные процедуры будут описаны более подробно.

2. СОЗДАНИЕ ИСХОДНОГО ФАЙЛА НА ПАСКАЛЕ.

Исходный файл с текстом программы на языке ПАСКАЛЬ является обычным текстовым файлом в смысле системы ФОДОС-2. Создание исходного файла может быть выполнено любым из системных редакторов ФОДОС-2.

Оператор языка ПАСКАЛЬ может размещаться на нескольких строках. Несколько операторов могут размещаться на одной строке. Пробелы и знаки табуляции <TAB> могут использоваться в любом месте и любом количестве для улучшения восприятия алгоритма программы (ее наглядности), а также и для цели выделения блочной структуры программы. Компилятор с языка ПАСКАЛЬ игнорирует знаки пробелов и табуляции, которые используются для форматирования программы.

Для выделения (разделения) отдельных страниц в листинге программы могут использоваться знаки <FF> (аналогично программам на МАКРОАССЕМБЛЕРе).

•

3. ТРАНСЛЯЦИЯ ПРОГРАММЫ НА ЯЗЫКЕ ПАСКАЛЬ.

Компилятор ПАСКАЛЬ использует текст программы в качестве входного файла. Во время работы компилятор создает файл с текстом программы на языке МАКРОАССЕМБЛЕР. Дополнительно к этому выходному файлу может создаваться (но не обязательно) файл с листингом, например:

```
.R PASCAL
*PROG,PROG=PROG
```

что приводит к трансляции программы на языке ПАСКАЛЬ из файла PROG.PAS и созданию файла PROG.MAC с текстом на МАКРОАССЕМБЛЕРе и файла PROG.LST с листингом.

В командной строке компилятора можно явно указывать имена устройств и ключи, управляющие трансляцией, например:

```
.R PASCAL
*PROG,LP:/N=PROG
```

что вызовет трансляцию программы PROG.PAS с получением файла PROG.MAC и листинга на устройстве печати (LP:). Листинг будет содержать только строки программы, содержащие ошибки и диагностические сообщения (указывается ключом «N»).

Если листинг вообще получать не нужно, то следует использовать, например, следующую команду:

.R PASCAL
*PROG=PROG

С помощью ключей, используемых в командной строке и/или тексте программы, можно управлять процессом трансляции.

ПРИМЕЧАНИЕ. Некоторые ключи могут использоваться только в тексте программы.

3.1. Ключ «D». Использование данного ключа приводит к включению в транслируемую программу информации для отладчика. Эта информация содержит имена переменных и процедур, а также номера операторов. В дополнение к этому в процессе компоновки в состав программы будет автоматически включен отладчик из библиотеки исполняющей системы ПАСКАЛЬ.

Использование данного ключа с ключом «S» позволит отладчику установить соответствие номеров операторов и исходного текста программы из листинга. Поэтому в этом случае нужно всегда создавать файл с листингом программы (см. раздел с описанием процедур отладки).

3.2. Ключ «E». Используется для того, чтобы определение процедуры после трансляции стало внешним. Процедуры и функции, которые в теле программы описаны как глобальные, должны транслироваться отдельно от основной программы и с использованием ключа «E», т. е. как внешние. Первые 6 символов (знаков) имени процедуры или функции используются для определения в качестве глобального символа точки входа (метку) для соответствующей подпрограммы.

3.3. Ключ «F». Использование этого ключа позволяет ускорить выполнение действий над операндами типа REAL, т. е. арифметические операции над числами в формате с плавающей запятой. Эта оптимизация может использоваться для процессоров без плавающей запятой. Оптимизация по скорости выполнения осуществляется за счет генерации компилятором соответствующих вызовов подпрограмм. В том случае, если ключ «F» не используется, то компилятор генерирует машинные инструкции для работы с плавающей запятой, а при выполнении программы исполняющая система осуществляет их эмуляцию. Время, затрачиваемое на эмуляцию, больше, чем выполнение инструкций обращения к подпрограммам.

Однако использование ключа «F» приводит к увеличению размера программы в оперативной памяти.

3.4. Ключ «L». Данный ключ разрешает полную распе-

чатку листинга текста программы на языке ПАСКАЛЬ. Он может использоваться в командной строке и исходном файле (см. ключ «N») для включения в листинг текста программы.

3.5. Ключ «N». Данный ключ запрещает распечатку исходного текста программы (операторов) в файл листинга за исключением строк (и, соответственно, операторов), содержащих ошибки.

3.6. Ключ «S». Этот ключ приводит к включению в генерируемый компилятором текст на МАКРОАССЕМБЛЕРе операторов исходного языка в качестве комментариев. Использование данного ключа облегчает анализ текста программы на МАКРОАССЕМБЛЕРе.

Ключ «S» несколько модифицирует действие ключа «D», как было описано выше.

3.7. Использование ключей в тексте программы

Все ключи, рассмотренные выше, допускают использование их в тексте программы. Это осуществляется использованием комментариев специального вида. Первым знаком комментария должен быть «¤». Следующий знак должен совпадать с символическим обозначением ключа, например, «(*¤S*)» приводит к включению в текст на МАКРОАССЕМБЛЕРе исходных операторов в качестве комментариев.

За мнемоникой ключа может использоваться знак «+» или «—». Эти знаки предназначены для «включения» действия ключа (+) или «выключения» (—), т. е. отмены его действия.

Ключи «A», «T» и «C» могут использоваться только внутри текста программы и не могут применяться в командной строке.

3.8. Ключ «A». Данный ключ отменяет генерацию проверок на допустимость значений индексов при работе с массивами — (*¤A—*), обычно компилятор генерирует инструкции, проверяющие выход значений индексов за допустимые границы. Если генерация проверок запрещена, то программа будет занимать несколько меньше оперативной памяти и будет выполняться несколько быстрее.

3.9. Ключ «T». Ключ (*¤T—*) запрещает генерацию инструкций, проверяющих переполнение стека. Всякий раз при вызове процедуры стек проверяется на наличие места для размещения локальных переменных. Отказ от использования проверок стека незначительно уменьшает размер оперативной памяти, занимаемой программой, и убыстряет ее выполнение.

3.10. Ключ «C». Специальный вид комментариев позволя-

ет включать в текст программы на языке ПАСКАЛЬ фрагменты на языке МАКРОАССЕМБЛЕР. Это осуществляется ключом «С», используемым в начале комментариев. Компилятор ПАСКАЛЬ анализирует входные строки для МАКРОАССЕМБЛЕРа для определения ссылок на соответствующие параметры и переменные. Например, для того, чтобы использовать значение глобальной переменной «ABC», нужно обращаться к ней с помощью «ABC(R5)», а для доступа к локальной переменной или параметру процедуры «XYZ» — использовать запись «XYZ(SP)»:

```
(*¤C
        TST     ABC(R5)     ;переменная ABC # 0 ?
        BNE     10¤         ;да
        MOV     XYZ(SP),R0  ;получить значение XYZ в R0
        ADD     #123.,R0    ;прибавить 123 к R0
        MOV     R0,@#100    ;записать R0 по абсолютному
                            ;адресу
10¤:    INC     XYZ(SP)     ;увеличить на 1 переменную
                            ;XYZ
*)
```

3.11. Листинг программы

Листинг, создаваемый компилятором ПАСКАЛЬ, содержит помимо исходного текста программы четыре колонки (пример листинга см. в разделе, посвященном процедурам отладки).

Первая колонка содержит номера исходных строк программы. Эта колонка в листинге обозначается как «LINE». Нумерация строк выполняется подряд, начиная с номера 1.

Вторая колонка, обозначаемая «STMT», содержит локальные номера операторов исходной программы и используется при отладке программ. Следует отметить, что один оператор может размещаться на нескольких строках. С другой стороны, несколько операторов могут размещаться на одной строке. Поэтому в колонке номеров операторов значения могут возрастать более чем на один при переходе от одной строки к другой.

Третья колонка содержит номера уровней вложенности процедур. Эта информация полезна для анализа «доступности» переменных в тех или иных процедурах, поскольку переменные, объявленные (определенные) на одном уровне, могут быть доступны на том же уровне и на уровнях, номера которых выше данного. Основная программа находится на 1-ом уровне вложенности. Процедуры и функции, определенные в программе, считаются имеющими уровень два.

В процедуре уровня два может, например, быть определена функция, которая будет иметь уже уровень вложенности три. Данная функция считается локальной и не может быть доступной из основной программы, т. е. из уровня один.

В третьей колонке отмечается уровень вложенности блоков операторов, ограниченных «BEGIN» и «END». Другими словами, операторы типа «BEGIN» и «REPEAT» приводят к увеличению на единицу номера уровня вложенности, пока не встретится соответствующий «END» или «UNTIL». Таким образом номера уровней вложенности могут использоваться для анализа программистом общей структуры программы с точки зрения вложенности блоков.

3.12. Диагностика ошибок трансляции

При обнаружении ошибки в процессе трансляции компилятор отмечает ошибочную строку несколькими знаками «*» в левой части листинга, за которым следует пояснительный текст. Знаком «^» отмечается то место в строке, которое привело к обнаружению ошибки.

Грубые ошибки, например, в блочной структуре программы, могут вызвать целый ряд (цепочку) других ошибок. В этом случае исправление первой ошибки приведет к «исчезновению» всех остальных ошибок. Использование ключа «N» при создании листинга приведет к получению в листинге только строк программы, в которых обнаружены ошибки, например:

```
.R PASCAL
*PROG,LP:/N=PROG
```

Общее число ошибок, обнаруженных во время трансляции, распечатывается в конце листинга. Одновременно выводится объем оперативной памяти, которая осталась неиспользованной до конца трансляции. Аналогичная информация выводится на экране системного терминала.

Возможны следующие сообщения об ошибках:

OVERFLOW
— переполнение;

MISSING END
— пропущено «END»;

TOO MANY SYMBOLS
— слишком много переменных и процедур;

TOO MANY LEVELS
— слишком много уровней вложенности процедур и функций;

TOO MANY WARNINGS
— слишком много предупреждений;

INPUT LINE TOO LONG
— длинная входная строка;
INVALID CHARACTER
— недопустимый знак встретился на строке;
8 OR 9 IN OCTAL CONSTANT
— в восьмеричной константе обнаружились цифры 8 или 9;
LABELS MUST BE INTEGERS
— в качестве меток допустимо использовать только целые числа;
CONSTANT OVERFLOW
— в программе слишком много констант; в компиляторе не хватает памяти для размещения всех констант;
MISSING LABEL DEFINITION
— отсутствует определение метки;
UNDEFINED FORWARD PROCEDURE OR FUNCTION
— недопустимо использовать процедуру или функцию прежде, чем они будут определены;
BAD PROGRAM NAME
— недопустимое имя программы;
IMPROPER SYMBOL
— недопустимый символ;
MISSING BEGIN
— пропущено «BEGIN»;
MISSING '.' AT PROGRAM END
— в конце программы отсутствует «.»;
ALL VAR DEFINITIONS PRECEED PROCEDURE DEFINITIONS
— определения всех переменных должно находиться в программе перед определением процедур;
BAD ORIGIN FOR VARIABLE
— недопустимое начальное значение для переменной;
BAD VARIABLE LIST
— недопустимый список переменных;
BAD TYPE
— недопустимое определение типа;
BAD LABEL
— неправильная метка;
BAD FUNCTION NAME
— недопустимое имя для функции;
BAD PROCEDURE NAME
— недопустимое имя для процедуры;
BAD FUNCTION RESULT TYPE
— недопустимый тип результата функции;

FORTRAN PARAM. MUST BE CALL BY REFERENCE
— фортрановский параметр должен всегда передаваться по ссылке;
DON'T REPEAT PARAMETER LIST
— не следует указывать список формальных параметров более одного раза;
',' USED INSTEAD OF ';'
— «,» была использована вместо «;»;
MISSING ')' AT END OF PARAM LIST
— пропущена «)» в конце списка формальных параметров;
BAD PARAMETER
— недопустимый формальный параметр;
BAD SCALAR TYPE
— недопустимый скалярный тип;
BAD SUBRANGE
— недопустимое указание интервала;
BAD TYPE SPECIFICATION
— недопустимое определение (указание) типа;
ARRAY INDEX TYPE ERROR
— ошибка в задании типа индекса для массива;
BAD RECORD
— недопустимая запись;
BAD FIELD LIST
— недопустимое определение списка полей;
FIELD LIST MUST BE IN PARENTHESES
— список полей должен указываться в скобках;
BAD VARIANT
— недопустимый вариант;
DUPLICATE FIELD NAME
— повторно используется имя для поля записи;
BAD CONSTANT
— недопустимая константа;
LABEL NOT DECLARED
— используется неопределенная метка;
LABEL REDEFINITION
— переопределение метки недопустимо;
LABEL DEFINED AT WRONG LEVEL
— определение метки на неправильном уровне;
UNDEFINED SYMBOL
— неопределенный символ;
MISSING SEMI—COLON
— пропущена «;»;

INVALID DECLARATION. PROBABLY MISSING END
— недопустимый оператор; возможно, что пропущено «END»;
INVALID SYMBOL
— недопустимый символ;
NOT IMPLEMENTED
— данная возможность не реализована в компиляторе;
MISSING LABEL
— пропущена метка;
BAD 'EXIT'
— пропущено «EXIT»;
DUPLICATE CASE LABEL
— повторно используется метка в операторе «CASE»;
ELSE NOT LAST IN CASE STMT
— «ELSE» не находится в конце оператора «CASE»;
MISSING END IN CASE STMT
— пропущено «END» в операторе «CASE»;
BAD CASE LABEL
— недопустимая метка выбора в операторе «CASE»;
MISSING UNTIL
— пропущено «UNTIL»:
BAD FOR STATEMENT
— неправильный формат оператора цикла «FOR»;
BAD WITH STATEMENT
— недопустимый оператор «WITH»;
WITH IN REG 0
— для оператора «WITH» используется регистр 0;
TOO MANY ARGUMENTS
— слишком много аргументов;
BAD ARGUMENT
— недопустимый аргумент;
TOO FEW ARGUMENTS
— указано мало аргументов;
NEW OR DISPOSE ARG NOT POINTER TYPE
— в процедурах «NEW» и «DISPOSE» может использоваться только переменная-указатель;
BAD READ STATEMENT
— недопустимый оператор «READ»;
BAD WRITE STATEMENT
— недопустимый оператор «WRITE»;
FORMAT MUST BE INTEGER TYPE
— для указания параметров формата может ис-

пользоваться только целочисленные переменные или константы;

FILE VARIABLE MISSING
— пропущена переменная файла;

BAD FILE NAME
— недопустимое имя файла;

ILLEGAL ASSIGNMENT
— недопустимое назначение;

BAD EXPRESSION
— ошибка в выражении;

MISSING ')'
— пропущена «)»;

MISSING OPERATOR
— пропущен оператор;

MISSING OPERAND
— пропущен операнд;

STRANGE '[' — BAD SET OR MISSING ARRAY DEF
— необъяснимое появление «[», что может быть связано с недопустимым множеством или неопределенным массивом;

UNDEFINED OPERAND — ASSUMING INTEGER
— неопределенный операнд; компилятор присваивает ему тип «INTEGER»;

FUNCTION ARGUMENT MISSING
— пропущен аргумент функции;

FUNCTION ARG MUST BE REAL OR INTEGER
— аргумент функции должен иметь тип «REAL» или «INTEGER»;

ARGUMENT MUST BE INTEGER TYPE
— аргумент должен иметь тип «INTEGER»;

ARG MUST BE REAL
— следует использовать аргумент типа «REAL»;

ARG MUST BE INTEGER
— следует использовать аргумент типа «INTEGER»;

ARG MUST BE NON—REAL SCALAR
— аргумент должен иметь скалярный тип, не являющийся «REAL»;

BAD 'ABS' ARG
— недопустимый аргумент функции «ABS»;

ODD' APPLIED TO NON—INTEGER EXPRESSION
— процедура «ODD» должна использоваться только с целочисленным аргументом (переменной, выражением и т. д.);

BAD SET ELEMENT
— недопустимый элемент множества;
MISSING FIELD VARIABLE
— пропущено указание поля записи;
UNDEFINED POINTER BASE TYPE
— неопределенный тип, на который ссылается переменная указатель;
BAD INDEX TYPE
— недопустимый тип индекса;
ILLEGAL OPERATOR
— недопустимый оператор;
ILLEGAL TYPE OF OPERAND
— недопустимый тип операнда;
INCOMPATIBLE ARRAYS
— несовместимые массивы;
BAD 'IN' OPERANDS
— недопустимые операнды для оператора «IN»;
OUT OF FLOATING AC'S
— для выполнения программы не хватает аккумуляторов процессора плавающей запятой (встретилось сложное выражение);
OUT OF REGISTERS
— не хватает регистров для выполнения программы (встретилась сложная конструкция);
BOOLEAN EXPRESSION NEEDED
— в данном месте программы необходимо использовать логическое выражение;
INCOMPATIBLE TYPE
— несовместимый тип;
MUST BE SIMPLE VARIABLE
— следует использовать простую переменную;
UNRESOLVED FORWARD TYPE REFERENCE
— неопределен тип из-за ссылок вперед на неопределенные типы данных;
ARRAY INDEX OUT OF RANGE
— индекс массива выходит за заданные границы;
FATAL ERROR
— фатальная ошибка компилятора; возможен аппаратурный сбой и ошибка в самом компиляторе;
TOO MANY ERRORS IN THIS LINE
— слишком много ошибок в данной строке;
EXPECTED '.' MISSING. ASSUMED WHERE INDICATED
— должна присутствовать «.»; показано подразумеваемое положение «.»;

ALL CHARACTERS IGNORED UNTIL ...
— все знаки до «...» игнорируются;
INDEX OUT OF RANGE
— индекс выходит за допустимые границы.

4. ТРАНСЛЯЦИЯ ПРОГРАММЫ МАКРОАССЕМБЛЕРОМ

Компилятор транслирует программы с языка ПАСКАЛЬ на язык МАКРОАССЕМБЛЕР (как было описано выше). Полученная программа нуждается в трансляции МАКРОАССЕМБЛЕРом для получения объектного модуля, например:

```
.R MACRO
*PROG=PROG
```

МАКРОАССЕМБЛЕР может выдавать также листинг транслируемой программы, но обычно он не представляет особого интереса. Если все же листинг программы на МАКРОАССЕМБЛЕРе необходим, то рекомендуется выполнять трансляцию компилятором с языка ПАСКАЛЬ, используя ключ «S», что приводит к включению в текст на МАКРОАССЕМБЛЕРе исходных строк на ПАСКАЛЕ в качестве комментариев.

ПРИМЕЧАНИЕ. Следует помнить, что компилятор ПАСКАЛЯ и МАКРОАССЕМБЛЕР создают файлы листинга, имеющие одинаковый тип «.LST». Поэтому последовательность команд:

```
.R PASCAL
*PROG/S,PROG=PROG
.R MACRO
*PROG,PROG=PROG
```

приведет к тому, что файл с листингом, полученный компилятором ПАСКАЛЯ, будет замещен одноименным файлом листинга, созданным МАКРОАССЕМБЛЕРом.

5. КОМПОНОВКА ПРОГРАММЫ НА ЯЗЫКЕ ПАСКАЛЬ

После получения объектного файла его необходимо скомпоновать с исполняющей системой ПАСКАЛЬ из объектной библиотеки (PASCAL.OBJ) для получения программы в формате загрузки, например:

```
.R LINK
*PROG=PROG,PASCAL
```

Библиотека PASCAL.OBJ с исполняющей системой ПАСКАЛЬ содержит набор модулей, к которым обращается программа, полученная компилятором ПАСКАЛЯ. Системный

компоновщик ФОДОС-2 включит в состав программы нужные модули из библиотеки и создаст программу в формате загрузки.

Внешними называются процедуры, которые компилируются и ассемблируются отдельно от основной программы. Включение внешних процедур в компонуемую программу осуществляется перечислением в командной строке компоновщику спецификаций входных файлов, содержащих модули внешних процедур. Например, если внешние процедуры содержатся в файлах «A», «B» и «C», то командная строка будет иметь вид:

```
.R LINK
*PROG=PROG,A,B,C,PASCAL
```

Допускается размещать внешние процедуры в оверлейных сегментах программы, например:

```
.R LINK
*PROG=PROG,PASCAL//
*A/O:1
*B/O:1
*C/O:1//
```

Более подробная информация о правилах построения оверлейных программ приводится в главе «Редактор связей. Руководство оператора» (кн. 3) при описании системного компоновщика «LINK».

6. ЗАПУСК И ВЫПОЛНЕНИЕ ПРОГРАММЫ

Запуск программы, написанной на языке ПАСКАЛЬ, осуществляется обычными средствами системы ФОДОС-2, т. е. использованием команд монитора «R» или «RUN», которые описаны в главе «Командный язык системы» (кн. 2).

Например: .R PROG

При загрузке программы возможны ошибки (системные), которые описаны в главе «Сообщения системы» (кн. 6). Сообщения, возникающие при выполнении программы на языке ПАСКАЛЬ, будут описаны ниже.

7. СООБЩЕНИЯ

В данном разделе представлены тексты сообщений об ошибках, возникающих при выполнении программы на языке ПАСКАЛЬ. Тексты сообщений снабжены пояснениями причин ошибок и, следовательно, способами их устранения.

NO ROOM ON DEVICE OR FILE NOT FOUND

— данное сообщение связано с ошибками при выполнении процедуры RESET, если специфицированный файл не был найден, или при работе процедуры REWRITE возникло отсутствие возможности для создания нового файла, т. е. нет места в каталоге носителя и т. п.

NOT A VALID DEVICE

— в процедуре RESET или REWRITE было указано устройство, драйвер которого отсутствует в системных таблицах резидентного монитора ФОДОС-2.

END OF FILE ON DEVICE

— данная ошибка связана с выполнением процедуры READ или GET, если в тот момент значение функции EOF было TRUE.

I/O CHANNEL NOT OPEN

— ошибка связана с выполнением операции ввода/вывода с каналом, который не был предварительно открыт процедурами RESET или REWRITE.

NOT ENOUGH AVAILABLE MEMORY

— сообщение свидетельствует о том, что динамическая область памяти исчерпана (как правило, при выполнении процедуры NEW); другая причина данного сообщения — недостаток памяти для рекурсивного вызова процедуры.

INTEGER ERROR

— ошибка связана с выполнением арифметических операций с переменными типа INTEGER; как правило, — это переполнение при умножении или выполнении процедур TRUNC или ROUND.

FLOATING POINT ERROR

— сообщение об этой ошибке связано с неправильным выполнением арифметической операции над переменными типа REAL в процессе вычислений.

LOG OF NEGATIVE
EXP OVERFLOW
SQRT OF NEGATIVE

— данные ошибки возникают при передаче неправильных параметров в функции языка ПАСКАЛЬ: LOG, EXP или SQRT.

DEVIDE BY ZERO

— ошибка связана с попыткой выполнить деление на 0;

BAD SET EXPRESSION

— ошибка при вычислении выражения, связанного с множеством: анализируемый элемент не принадлежит данному множеству;

ARRAY BOUND ERROR

— при вычислении выражения или т. п. значение индекса приняло недопустимое значение;

TRAP TO 4

— произошло прерывание по вектору четыре, которое связано с обращением к несуществующей ячейке памяти или использованием недопустимого режима адресации в инструкции процессора; обычно эта ошибка появляется из-за ошибок в фрагментах, написанных на МАКРОАССЕМБЛЕРе;

BAD SUPPORT PACKAGE
RESERVED INSTRUCTION TRAP
MISSING SPECIAL FEATURE

— ошибка связана с использованием исполняющей системы ПАСКАЛЯ, не соответствующей типу центрального процессора, а также применением инструкций (в фрагментах на МАКРОАССЕМБЛЕРе), отсутствующих в наборе инструкций центрального процессора;

SUPPORT CONDITIONALS ERROR

— ошибка при выполнении логической операции;

NEW OF LENGTH 0

— ошибка связана с выполнением процедуры «NEW» при создании переменной нулевой длины;

FATAL I/O ERROR

— произошла неустранимая ошибка ввода/вывода;

TOO MANY FILES OPEN

— сделана попытка открыть слишком много файлов для ввода/вывода.

8. ТРАНСЛЯЦИЯ ВНЕШНИХ ПРОЦЕДУР

Внешними называются процедуры или функции, которые компилируются и ассемблируются отдельно от основной программы на языке ПАСКАЛЬ. Компоновщик используется для объединения внешних процедур и основной программы и получения программы в формате загрузки. Средства внеш-

них процедур позволяют облегчить процесс разработки и отладки программного обеспечения, поскольку внесение изменений в основную программу не требует повторной компиляции и ассемблирования внешних процедур. Отладка больших программ существенно упрощается путем создания полностью тестированных внешних процедур. В этом случае при отладке основной программы ее компилируют с ключом «D». Однако локальные переменные внешних процедур становятся недоступными отладчику. Размер отладочной информации в программе существенно уменьшается. В отдельных случаях, когда программа очень велика по размеру занимаемой оперативной памяти, единственная возможность проведения отладки — использование внешних процедур, прошедших этап тестирования.

Другими словами, когда над большим программным продуктом работает несколько программистов, полезно использовать совместно разработанную и отлаженную библиотеку внешних процедур.

Внешние процедуры и функции создаются аналогично обычным процедурам в языке ПАСКАЛЬ. Использование ключа «E» при трансляции, приводит к объявлению всех процедур и функций на первом уровне вложенности глобальными. Первые 6 знаков в имени процедуры или функции объявляются в качестве глобальной точки входа (метки), которая будет использована компоновщиком при объединении внешних модулей и основной программы. Специальная форма комментариев (*¤E+*) может также использоваться для определения процедуры в качестве внешней. Программа, которая использует внешнюю процедуру, должна включать заголовок этой процедуры, за которым следует ключевое слово «EXTERNAL». Надо соблюдать осторожность при задании типов и порядка следования параметров процедуры, поскольку компилятор лишен возможности проверить эту информацию.

Компилятор не отводит место для размещения переменных на первом уровне (глобальных). Вместо этого компилятор присваивает им соответствующие адреса с тем, чтобы они совпадали при выполнении с глобальными переменными основной программы. Вся ответственность на совпадение описаний глобальных переменных во внешних процедурах и основной программы возлагается на программиста. Пользоваться этой возможностью следует очень осторожно, поскольку она может служить источником различных непредсказуемых ошибок (и результатов).

Например:

глобальные переменные (файл GLOBAL.PAS):

```
VAR                 (* глобальные переменные *)
    I: INTEGER;
    R: REAL;
    C: CHAR;
```

Основная программа **(файл MAIN.PAS):**

```
(* внешняя процедура *)
PROCEDURE EXPROC(X:INTEGER); EXTERNAL;
BEGIN               (* Начало MAIN *)
    I:=1; R:=3.1415926; C:='X';
    EXPROC(5);      (* вызов внешней процедуры *)
END.                (* Конец MAIN *)
```

внешняя процедура (файл EXTERN.PAS):

```
(*$E+*)
PROCEDURE EXPROC(Y:INTEGER); (* внешняя процеду-
ра *)
BEGIN               (* начало внешней процедуры *)
    WRITELN('I= ',I:2);
    WRITELN('R= ',R);
    WRITELN('C= ',C);
    WRITELN('PROCEDURE ARGUMENT= ',Y:3);
END;                (* конец процедуры *)
```

Ниже представлена последовательность команд для выполнения программы:

```
.R PASCAL
*MAIN=GLOBAL,MAIN
.R PASCAL
*EXTERN=GLOBAL,EXTERN
.R MACRO
*MAIN=MAIN
*EXTERN=EXTERN
*^C
.R LINK
*PROG=MAIN,EXTERN,PASCAL
*^C
.R PROG
```

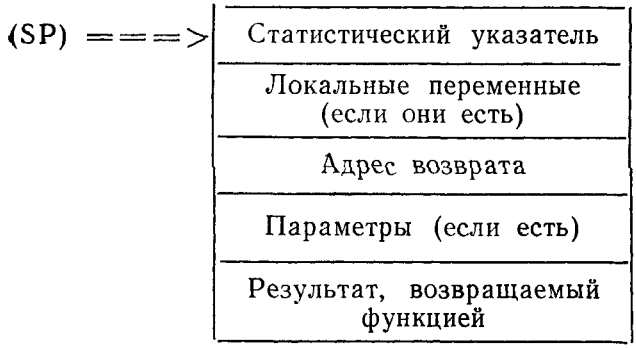
9. ОСОБЕННОСТИ ВЫПОЛНЕНИЯ ПРОГРАММЫ НА ЯЗЫКЕ ПАСКАЛЬ

Для того, чтобы эффективно использовать вставки на МАКРОАССЕМБЛЕРе в программе на языке ПАСКАЛЬ, необходимо понимать некоторые особенности его реализации. Наиболее важные детали будут описаны ниже, хотя рекомен-

дуется (и это является весьма полезным) заняться изучением текста на АССЕМБЛЕРе, который генерирует компилятор ПАСКАЛЬ.

При выполнении программа на языке ПАСКАЛЬ использует две области данных: глобальная область и стек. В глобальной области содержатся все переменные, определенные на уровне ноль, т. е. в начале программы. Область стека используется для хранения локальных переменных для процедур и адресов передачи (возврата) управления. Пятый регистр R5 всегда содержит адрес начала глобальной области, а регистр шестой (SP) указывает на начало программного стека. Содержимое этих регистров не должно меняться во фрагменте на АССЕМБЛЕРе (или должно сохраняться, а затем восстанавливаться).

При вызове процедуры исполняющая система сохраняет в стеке адрес возврата и отводит в стеке место для параметров и локальных переменных. Каждый очередной вызов процедуры (возможно рекурсивный) приводит к созданию в стеке очередной области данных, как показано ниже:



Статический указатель содержит адрес области стека, выделенной при последнем вызове процедуры. Этот указатель используется для вычисления адресов переменных промежуточных уровней (переменных, которые не являются ни локальными, ни глобальными). Все локальные переменные для текущей процедуры могут использоваться путем индексации относительно текущего указателя стека — SP. Доступ к глобальным переменным осуществляется индексацией регистра R5. Доступ к промежуточной переменной может осуществляться путем «движения» по стеку с использованием статических указателей до тех пор, пока не будет найдена требуемая область стека. После этого индексация осуществляется

относительно найденной базы. Для того, чтобы сделать доступ к переменным более простым для программиста на МАКРОАССЕМБЛЕРе, компилятор определяет имя переменной в качестве смещения относительно соответствующего указателя (R5 или SP).

Аналогично, точки входа в процедуры и функции компилятор заменяет на одноименные метки. Следует помнить, что компилятор не проверяет правильность обращений к переменным во фрагментах на МАКРОАССЕМБЛЕРе.

Пример:

```
FUNCTION SWAPBYTES (I:INTEGER):INTEGER;
BEGIN
        (*¤C
                SWAB    I(SP)
        *)
        SWAPBYTES := I;
END;
```

Если необходимо всю процедуру написать на МАКРОАССЕМБЛЕРе, то это можно осуществить, просто заключив весь фрагмент в заголовок процедуры, т. е. указать начало и конец процедуры по правилам ПАСКАЛЯ. В теле процедуры фрагменты на ПАСКАЛЕ и МАКРОАССЕМБЛЕРе могут чередоваться, например:

```
PROCEDURE EXAMPLE;
BEGIN
    (*¤C
            ; фрагмент на МАКРОАССЕМБЛЕРе
    *)
WRITE (' текст сообщения');(* фрагмент на ПАСКАЛЕ *)
    (*¤C
            ; еще фрагмент на МАКРОАССЕМБЛЕРе
    *)
END;
```

ПРИМЕЧАНИЕ. Следует помнить, что функции сохраняют содержимое регистров R0—R4 (включая регистры-аккумуляторы процессора с плавающей запятой), а процедуры — не сохраняют.

10. ПРОГРАММА IMP

Программа IMP написана на языке ПАСКАЛЬ и предназначена для выполнения некоторой оптимизации программ, создаваемых компилятором ПАСКАЛЬ в операционной системе ФОДОС-2.

Как указывалось выше, компилятор транслирует программу с языка ПАСКАЛЬ на язык МАКРОАССЕМБЛЕР. Программа IMP обрабатывает текст программы на МАКРОАССЕМБЛЕРе, выполняет отдельные виды оптимизации и создает более эффективную программу тоже на языке МАКРОАССЕМБЛЕР.

Оптимизации подвергаются инструкции переходов (JMP, BR и т. п.). IMP выполняет замену инструкции JMP на BR, если это возможно. При этом экономится 1 слово памяти. Например:

```
        JMP     L20
```

заменяется на:

```
        BR      L20
```

В некоторых случаях выполняются более сложные виды оптимизации:

```
        BGT     L30
        JMP     L50
L30:    ...
```

Заменяется на:

```
        BLE     L50
```

В этом случае экономится 2 слова памяти.

10.1. Вызов и загрузка

Вызов программы IMP осуществляется командами монитора R или RUN. После загрузки программа запрашивает имя файла, в котором находится текст программы, которую нужно оптимизировать. После этого IMP запрашивает спецификацию выходного файла для оптимизированной версии программы.

После ввода спецификаций файлов IMP начинает работать. В конце IMP распечатывает статистику о процессе оптимизации, т. е. число слов, на которое сократился размер программы, и процентное уменьшение размера программы.

Пример:

```
.R IMP
INPUT FILE? DK:PROG.MAC
OUTPUT FILE? DK:PROG1.MAC
IMPROVEMENT 50 WORDS 8.7%
```

ПРИМЕЧАНИЕ. По умолчанию тип входного и выходного файлов «MAC».

Программа IMP может использоваться для оптимизации как полных программ на языке ПАСКАЛЬ, так и отдельных частей, которые оформлены по правилам внешних процедур.

Текст программы на МАКРОАССЕМБЛЕРе не должен содержать макрокоманд. Регистры должны обозначаться толь-

ко как %0, %1,..., %6, %7. В тексте программы можно использовать комментарии.

Процедура оптимизации выполняется сравнительно медленно. Поэтому оптимизировать программу следует только после того, как она полностью отлажена.

10.2. Сообщения программы IMP

Ниже приводятся тексты сообщений, выдаваемых IMP с пояснениями.

NO <> IN LINE
NNNN LINE

— в строке с номером «NNNN» отсутствуют скобки «<» и «>».

ERROR AT BRJMP NODE NNNN

— в строке с номером «NNNN» обнаружена ошибка, возможно, связанная с использованием ключа «С» и вставленным в текст на ПАСКАЛЕ фрагмента на МАКРОАССЕМБЛЕРе.

ERROR AT JMP NODE NNNN

— в строке с номером «NNNN» обнаружена ошибка, возможно, связанная с использованием ключа «С» и вставленным в текст на ПАСКАЛЕ фрагмента на макроассемблере.

11. ПРОГРАММА FORM

Программа FORM написана на языке ПАСКАЛЬ и предназначена для выполнения форматирования исходного текста различных пользовательских программ на ПАСКАЛЕ, а также для создания и распечатки таблиц перекрестных ссылок.

ПРИМЕЧАНИЕ. При форматировании программы производится проверка ее блочной структуры.

11.1. Вызов и загрузка

Вызов программы FORM осуществляется командами монитора R или RUN. После загрузки программа выводит на терминал «*» и ожидает командной строки в формате CSI.

Командная строка для FORM может содержать до 2-х выходных и один входной файл.

Спецификация входного файла определяет исходный текст программы, которая должна быть подвергнута форматированию. Тип файла по умолчанию — «.PAS».

Спецификация первого выходного файла задает файл, в который будет помещен отформатированный текст исходной программы на ПАСКАЛЕ. Эта спецификация не является обязательной. Тип файла по умолчанию — «.PAS».

Спецификация второго выходного файла задает файл, в который будет помещена таблица перекрестных ссылок программы и ее листинг. Спецификация данного выходного файла не является обязательной. Тип файла по умолчанию — «.LST».

Пример:

```
.R FORM
*PROG1,LP:=PROG
NO ERROR IN BLOCKSTRUCTURE
.
```

Сформатированный текст программы на ПАСКАЛЕ можно использовать для последующей работы, т. е. транслировать компилятором, редактировать, отлаживать и т. д.

11.2. Сообщения программы FORM

Сообщения, выдаваемые программой FORM на терминал:

LINE NNNN TOO LONG
— строка исходной программы с номером NNNNN содержит более 147 знаков, что является недопустимым;

MISSING POINT AT PROGRAM END
— в конце программы на языке ПАСКАЛЬ отсутствует «.»;

COMMAND STRING ERROR
— командная строка для FORM содержит ошибку;

ERROR IN BLOCKSTRUCTURE
— блочная структура программы на ПАСКАЛЕ содержит ошибку;

NO ERROR IN BLOCKSTRUCTURE
— при форматировании программы ошибок в блочной структуре не обнаружено; данное сообщение свидетельствует о нормальном завершении работы программы FORM.

Следующие сообщения появляются во втором выходном файле, создаваемом программой FORM (в листинге таблиц перекрестных ссылок):

NEXT LINE TOO LONG
— следующая строка листинга содержит более 147 знаков;

MISSING 'END' OR 'UNTIL' NUMBER NNNN
— пропущено «END» или «UNTIL» с номером NNNN;

MISSING 'THEN' NUMBER NNNN
— пропущена операторная скобка «THEN» с номером NNNN;

MISSING 'OF' TO 'CASE' NUMBER NNNN
— пропущен «OF» в операторе «CASE» с номером NNNN;
ONLY ONE 'EXIT' ALLOWED
— в данном контексте допустим только один оператор «EXIT»;
MISSING 'EXIT' IN 'LOOP' NNNN
— пропущено «EXIT» в операторе цикла с номером NNNN.

Ниже перечисляются нормальные заголовки, появляющиеся в листинге сформатированной программы:
CROSS REFERENCE LISTING OF VARIABLES
— таблица перекрестных ссылок переменных;
LIST OF PROCEDURES
— список процедур;
X IS CALLED BY Y
— процедура «X» вызывается процедурой «Y»:
X CALLS UP Y
— процедура «X» вызывает процедуру «Y»;
INDENTED PROCEDURE LIST
— порядковый список процедур.

12. СРЕДСТВА ОТЛАДКИ ПРОГРАММ

Отладка программ на языке ПАСКАЛЬ осуществляется специальным модулем, входящим в состав исполняющей системы ПАСКАЛЬ. Отладчик позволяет полностью управлять выполнением программы, распечатывать и модифицировать значения переменных и т. п.

Включение отладчика в состав выполняемой программы осуществляется автоматически при использовании ключа «D» в процессе компиляции. Данный ключ приводит к включению в состав генерируемой программы дополнительной информации о размещении операторов и именах переменных. Использование ключа «S» позволяет существенно расширить объем информации, доступной в процессе отладки. Однако, в этом случае нужно обязательно получить файл с листингом отлаживаемой программы. Тогда при выполнении точки останова или возникновении ошибки отладчик будет распечатывать соответствующую исходную строку программы на языке ПАСКАЛЬ, в которой произошел останов.

При запуске отлаживаемой программы отладчик распечатывает свой идентификатор на терминале. Если при компиляции использовался ключ «S», то отладчик запросит спецификацию файла, содержащего листинг отлаживаемой про-

граммы. Файл с листингом должен размещаться на внешнем запоминающем устройстве каталоговой организацией и создаваться одновременно с компиляцией отлаживаемой программы.

Когда отладчик готов к приему команды, то он распечатывает знак «]» в левом углу терминала. После этого можно вводить команды отладчику. Поскольку модуль отладчика также написан на языке ПАСКАЛЬ, то он считывает команды (вводит команды) обычными операторами ввода, которые имеются в ПАСКАЛЕ. Поэтому, если введенная команда содержит ошибку, отладчик распечатает соответствующее сообщение и повторит запрос команды.

Все примеры работы с отладчиком в данном разделе демонстрируются на программе DBGTST (которая находится в файле DBGTST.PAS). Компиляция этой программы осуществляется командами:

```
.R PASCAL
*DBGTST,DBGTST=DBGTST/S/D
```

При компиляции создается файл с листингом программы DBGTST.LST. Распечатка листинга имеет следующий вид:

```
DBGTST     P  A  S  C  A  L          13-12-82          PAGE 1
           ДВК  "ЭЛЕКТРОНИКА  НЦ 8020/2,7,4
LINE  STMT LEVEL  NEST   SOURCE STATEMENT
  1                     PROGRAM DBGTST;
  2
  3                     VAR     A,B:INTEGER;
  4
  5                        PROCEDURE  SUM(C,D,E:INTEGER);
  6
  7                          VAR          I:INTEGER;
  8
  9                             FUNCTION ADD(P,Q:INTEGER)
 10                                          :INTEGER;
 11                             VAR       X,Y:INTEGER;
 12
 13                           BEGIN (* START OF ADD *)
 14     1      3      1        X:=P;
 15     2      3      1        Y:=Q;
 16     3      3      1        IF Y=0 THEN ADD:=X
 17     5      3      2         ELSE ADD:=ADD(0,X)+ADD(Y,0);
 18     6      3      1       END; (* END OF ADD *)
 19
 20                           BEGIN (* START OF SUM *)
 21     1      2      1          I:=ADD(C,D);
 22     2      2      1          I:=ADD(1,E);
 23     3      2      1       END; (* END OF SUM *)
 24
 25                        BEGIN (* START OF MAIN *)
```

```
26      1       1       1         A:=1;
27      2       1       1         B:=2;
28      3       1       1         SUM(3,4,5);
29      4       1       1     END. (* END OF MAIN *)
```

Из текста программы можно видеть, что она содержит глобальную процедуру «SUM» и две переменные «A» и «B». Термин «глобальная» используется в данном случае потому, что переменные «A» и «B» и процедура «SUM» могут быть доступны на протяжении всей программы. Функция «ADD» является локальной, т. е. внутренней для процедуры «SUM». Поэтому функция «ADD» может вызываться только внутри процедуры «SUM», но не из основной программы. Аналогично, переменная «I» является локальной для процедуры «SUM», а переменные «X» и «Y» — локальными для функции «ADD».

Остальная часть данного раздела посвящена описанию команд отладчика.

ПРИМЕЧАНИЕ. Данная версия отладчика ПАСКАЛЬ допускает работу только под управлением BJ- или FB-мониторов.

12.1. Точка останова (;B)

Одной из важнейших возможностей отладчика является способность прерывать выполнение отлаживаемой программы. Если с помощью команды установить точку останова на требуемом операторе программы, то, когда выполнение программы дойдет до указанного оператора, управление будет передано отладчику. После этого отладчику можно давать команды, с помощью которых можно, например, просматривать значения переменных и, в случае необходимости, их модифицировать.

Для того, чтобы установить точку останова, необходимо напечатать имя процедуры или функции, затем запятую, за которой следует номер оператора (где следует установить точку останова) и «;B». В каждой процедуре и функции операторы нумеруются последовательно, начиная с 1-го. Номера операторов распечатываются в листинге программы в колонке, помеченной «STMT».

Если несколько операторов размещаются на одной строке, то в колонке «STMT» листинга распечатывается номер первого оператора в этой строке.

ПРИМЕЧАНИЕ. Следует помнить, что в каждый момент времени может быть только одна точка останова.

Примеры:

```
] MAIN,3;B
```

```
] SUM,2;B
] ADD,1;B
```

После того, как установлена точка останова на операторе, отладчик будет получать управление всякий раз перед выполнением этого оператора. Следует учесть, что отладчик не проверяет допустимость номера оператора в команде, определяющей точку останова. Поэтому, если, например, установить точку останова на операторе «FOO,99», который не существует, то управление отладчику никогда не попадет. Однако это дает возможность опытному программисту определять точки останова в оверлейных процедурах, которые не находятся в оперативной памяти в момент определения точки останова.

Отмена точки останова осуществляется командой «0;B» или «;B».

Пример:

```
PASCAL DEBUGGER V2.2
LISTING FILE NAME? DBGTST
] SUM,1;B
] ,G
BREAKPOINT AT SUM,1 I:=ADD(C,D);
] ;P
PROGRAM TERMINATION AT MAIN,4 END. (* END OF MAIN *)
]
```

12.2. Запуск (;G) и продолжение (;P) программы

В предыдущем примере команды «;G» и «;P» использовались для запуска или продолжения выполнения программы. Если точка останова определена, то в команде «;P» можно задать число раз, которое будет пропущена точка останова (т. е. управление будет передано отладчику только после того, как оператор выполнится указанное число раз). Это осуществляется с помощью числового аргумента команды:

```
] ADD,1;B
] ;G
BREAKPOINT AT ADD,1
] 3;P
BREAKPOINT AT ADD,1
]
```

В показанном примере, после начальной точки останова, отладчик получит управление после того, как функция «ADD» будет выполнена 3 раза, т. е. на 4-ый раз.

При выполнении программы в пошаговом режиме команда «N;P» приведет к выполнению «N» операторов. Только после этого управление будет передано отладчику.

12.3. Пошаговый режим выполнения (;S)

Команды «N;S», где N не равно нулю, переводит отладчик в пошаговый режим выполнения программы. При этом управление будет передаваться отладчику после выполнения очередного оператора, как будто там была определена точка останова. Команда продолжения «;P» приводит к выполнению одного очередного оператора программы. Если использовать команду «N;P», то без остановки будет выполнены «N» операторов. Выход из пошагового режима задается командой «0;S» или просто «;S».

Пример использования пошагового режима:

```
PASCAL DEBUGGER V2.2
LISTING FILE NAME? DBGTST
] 1;S
] ;G
BREAKPOINT AT MAIN,1 A:=1;
] ;P
BREAKPOINT AT MAIN,2 B:=2;
] ;P
BREAKPOINT AT MAIN,3 SUM(3,4,5);
] 13;P
BREAKPOINT AT ADD,4 IF Y=0 THEN ADD:=X
]
```

12.4. Трассировка выполняемых операторов (;T)

Если отладчик находится в режиме трассировки, то перед выполнением очередного оператора распечатываются его координаты, т. е. имя процедуры и номер этого оператора. Если использовался ключ «S» при компиляции, то распечатывается также исходная строка, содержащая оператор. Таким образом трассировка позволяет контролировать (наблюдать) процесс выполнения программы. Режим трассировки устанавливается командой «N;T», где N не равно нулю. Завершение режима трассировки (выход из этого режима) осуществляется по команде «0;T» или «;T».

ПРИМЕЧАНИЕ. Следует помнить, что, если в одной строке находится несколько операторов, то строка будет распечатана только один раз:

```
PASCAL DEBUGGER V2.2
LISTING FILE NAME? DBGTST
] SUM,2;B
] ;G
```

```
BREAKPOINT AT SUM,2 I:=ADD(I,E);
| 2;T
| ;P
ADD,1 X:=P;
ADD,2 Y:=Q;
ADD,3 IF Y=0 THEN ADD:=X
ADD,5 ELSE ADD:=ADD(0,X)+ADD(Y,0);
ADD,1 X:=P;
ADD,2 Y:=Q;
ADD,3 IF Y=0 THEN ADD:=X
ADD,5 ELSE ADD:=ADD(0,X)+ADD(Y,0);
ADD,1 X:=P;
ADD,2 Y:=Q;
ADD,3 IF Y=0 THEN ADD:=X
ADD,6 END; (* END OF ADD *)
ADD,1 X:=P;
ADD,2 Y:=Q;
ADD,6 END; (* END OF ADD *)
ADD,6 END; (* END OF ADD *)
ADD,1 X:=P;
ADD,2 Y:=Q;
ADD,3 IF Y=0 THEN ADD:=X
ADD,6 END; (* END OF ADD *)
ADD,6 END; (* END OF ADD *)
SUM,3 END; (* END OF SUM *)
MAIN,4 END. (* END OF MAIN *)
PROGRAM TERMINATION AT MAIN,4 END. (* END OF MAIN *)
|
```

12.5. Трассировка выполняемых процедур (;C)

В большинстве случаев трассировка всей программы может потребовать много времени. Для решения этой проблемы отладчик обеспечивает более короткую форму трассировки — трассировку вызовов процедур и функций.

Данный режим трассировки задается командой «N;C», где N не равно нулю. Завершение данного режима трассировки осуществляется командой «0;C» или «;C».

В данном режиме трассировки отладчик распечатывает информацию о входе в процедуру или функцию, а также при их соответствующем завершении:

```
PASCAL DEBUGGER V2.2
LISTING FILE NAME? DBGTST
1,C
1,G
```

```
ENTERING SUM FROM MAIN,3 SUM(3,4,5);
ENTERING ADD FROM SUM,1 I:=ADD(C,D);
ENTERING ADD FROM ADD,5 ELSE ADD:=ADD(0,X)+ADD(Y,0);
ENTERING ADD FROM ADD,5 ELSE ADD:=ADD(0,X)+ADD(Y,0);
LEAVING  ADD
ENTERING ADD FROM ADD,5 ELSE ADD:=ADD(0,X)+ADD(Y,0);
LEAVING  ADD
LEAVING  ADD
ENTERING ADD FROM ADD,5 ELSE ADD:=ADD(0,X)+ADD(Y,0);
LEAVING  ADD
LEAVING  ADD
ENTERING ADD FROM SUM,2 I:=ADD(I,E);
ENTERING ADD FROM ADD,5 ELSE ADD:=ADD(0,X)+ADD(Y,0);
ENTERING ADD FROM ADD,5 ELSE ADD:=ADD(0,X)+ADD(Y,0);
LEAVING  ADD
ENTERING ADD FROM ADD,5 ELSE ADD:=ADD(0,X)+ADD(Y,0);
LEAVING  ADD
LEAVING  ADD
ENTERING ADD FROM ADD,5 ELSE ADD:=ADD(0,X)+ADD(Y,0);
LEAVING  ADD
LEAVING  ADD
LEAVING  SUM
PROGRAM TERMINATION AT MAIN,4 END. (* END OF MAIN *)
]
```

12.6. Слежение за переменной (;W)

Команда слежения используется для «наблюдения» за значением переменной. Если указанная переменная изменяет свое значение, то отладчик получает управление, как и в точке останова. Для установки наблюдения за переменной необходимо напечатать имя переменной, а затем «;W». Чтобы снять наблюдение, используется команда «—1;W».

Перед выполнением очередного оператора отладчик проверяет, изменилось ли значение наблюдаемой переменной. Если значение изменилось, то отладчик печатает об этом сообщение и переходит в состояние ожидания команды.

ПРИМЕЧАНИЕ. Следует помнить, что отладчик выполняет останов после оператора, который привел к модификации значения переменной. Для того, чтобы распечатать тот оператор, который вызвал модификацию, необходимо использовать команду «¤L», описанную ниже.

После продолжения выполнения программы наблюдение за переменной будет продолжаться.

Особая осторожность требуется при слежении за локальными переменными. Установить слежение за локальной переменной можно только, когда управление будет передано соответствующей процедуре или функции. Решение проблемы заключается в том, чтобы установить точку останова на первом операторе процедуры. При достижении этой точки оста-

нова можно установить слежение за одной из локальных переменных этой процедуры.

Когда процедура или функция завершается, то локальная переменная перестает быть доступной, и отладчик снимает наблюдение за ней и распечатывает соответствующее сообщение:

```
PASCAL DEBUGGER V2.2
LISTING FILE NAME? DBGTST
| SUM,1;B
| ;G
BREAKPOINT AT SUM,1 I:=ADD(C,D);
| I/
0
] I;W
| ;P
WATCHED VALUE CHANGED.
BREAKPOINT AT SUM,2 I:=ADD(I,E);
| I/
'
| ;P
WATCHED VALUE CHANGED.
BREAKPOINT AT SUM,3 END; (* END OF SUM *)
| I/
12
| MAIN,4;B
] ;P
WATCH TERMINATED. VALUE DIDN'T CHANGE.
BREAKPOINT AT MAIN,4 END. (* END OF MAIN *)
]
```

12.7. Распечатка содержимого переменных

Отладчик может распечатывать содержимое переменных в четырех форматах: INTEGER, BYTE, REAL, CHAR. Для распечатки переменной следует ввести ее имя и команду, определяющую формат распечатки:

/ — INTEGER в десятичном виде;

\ — BYTE в десятичном виде;

% — REAL в формате с плавающей запятой;

' — один знак в символьном виде (CHAR).

Формат распечатки INTEGER обычно используется при распечатке значений переменных, объявленных как INTEGER.

Формат BYTE бывает полезным при выводе скалярных типов или переменных типа BOOLEAN. Символьная информация также может распечатываться форматом BYTE. Формат REAL используется для вывода переменных типа REAL, а формат CHAR — переменных типа CHAR.

Следует помнить, что отладчик не имеет информации о типах переменных. Поэтому можно распечатывать одну и ту же переменную по различным форматам, однако результаты при этом будут разные, например:

```
PASCAL DEBUGGER V2.2
LISTING FILE NAME? DBGTST
] MAIN,3;B
] ;G
BREAKPOINT AT MAIN,3 SUM(3,4,5);
] A/
1
] B/
2
] I/
CAN'T FIND VARIABLE CALLED «I»
2
] SUM,2;B
] ;P
BREAKPOINT AT SUM,2 I:=ADD(I,E);
] I/
7
]
```

Для удобства распечатки массивов можно использовать счетчик повторения после знака, определяющего формат распечатки. Например, для распечатки знакового массива «X», состоящего из 30-ти знаков, необходима команда «X'30». Если этот же массив имеет тип REAL, то его можно распечатать по команде «X/30». При распечатке элементов целых массивов одновременно выводятся и их адреса в виде «.+N», где N — это смещение в байтах от начального адреса массива. Адрес элемента выводится перед распечаткой значения элемента.

При указании адреса переменной, хотя обычно используется один идентификатор, можно применять простые выражения. Например, для распечатки 10-го элемента массива целых чисел следует воспользоваться командой «X+20». Очевидно, что десятый элемент целочисленного массива находится, начиная с 20-го байта. После того, как была распечатано значение переменной, ее адрес можно обозначать «.». Напри-

мер, если была распечатана переменная «X+20/» и нужно распечатать следующий элемент массива «X+22/», то можно воспользоваться командой «.+2/». Из текущего адреса можно также вычитать значения, т. е. допустима конструкция типа «.—4».

При использовании переменных-указателей можно проследить любой уровень ссылки использованием оператора «^» (косвенная адресация для распечатки переменных-указателей). Например, если переменная «X» указывает на целое, то это число можно распечатать командой «X^/». Оператор «.» идентичен оператору «+» и может применяться при распечатке сложных записей. Конструкция «X.4» полностью эквивалентна «X+4», что означает третье слово записи, на которую указывает переменная «X» (все смещения задаются в байтах).

Имеющиеся операторы дают возможность распечатывать очень сложные структуры данных, например, «X.4^.2.100^.4».

Для того, чтобы узнать адрес переменной или записи (вместо ее значения), следует использовать оператор «=». Например, «X=» приведет к распечатке адреса переменной «X», а не ее значения. Однако при работе с абсолютными адресами следует соблюдать осторожность. Локальные переменные функций или процедур размещаются в стеке. Поэтому их адреса меняются (или могут меняться) от вызова к вызову. Также не следует пытаться распечатать значения переменных, про которые отладчик «говорит», что он найти их не может («CAN'T FIND VARIABLE»).

12.8. Изменение содержимого переменных

Отладчик позволяет изменять значения переменных, используя оператор присваивания «:=». Для того, чтобы присвоить переменной «A» значение «1», можно использовать оператор «A:=1». Оператор «X:='строка'» присвоит знаковому массиву строку знаков. Поскольку отладчик не располагает информацией о типах переменных, то в операторе «:=» нужно указывать тип. Например, чтобы присвоить переменной «X» типа REAL значение 1.234, нужно ввести команду «X:=R1.234».

Примеры:

```
] A:=1
] FILENAME:='TEST.DAT'
] CORRELATION:=R.09332123456E—2
```

12.9. Распечатка списка имен переменных

Если отладка программы выполняется без листинга или часто используются локальные процедуры и функции, в кото-

рых определены локальные переменные, то бывает полезным распечатать список переменных, которые в данный момент являются доступными. Распечатку имен переменных можно выполнить командой «¤V». Список распечатывается, начиная с имени текущей локальной процедуры (функции), за которым следуют имена переменных, определенных в данной процедуре (функции). Далее распечатываются процедуры, находящиеся на уровнях вложенности, имеющих меньшие номера, а также их локальные переменные. В конце списка выводятся имена глобальных переменных. Список всех переменных может быть достаточно большим, однако, используя команду <CTRL/O>, можно прекратить распечатку.

Примеры:

```
PASCAL DEBUGGER V2.2
LISTING FILE NAME? DBGTST
[ ¤V
MAIN
    A
    B
] SUM,1;B
] ;G
BREAKPOINT AT SUM,1 I:=ADD(C,D);
] ¤V
SUM
    C
    D
    E
    I
MAIN
    A
    B
] ADD,1;B
] ;P
BREAKPOINT AT ADD,1 X:=P;
] ¤V
ADD
    P
    Q
    X
    Y
SUM
    C
    D
    E
```

```
    I
MAIN
    A
    B
]
```

12.10. Распечатка списка вызванных процедур

Если выполнение программы прервано, например, по точке останова, то обычно бывает полезно понять, каким путем выполнение программы привело к точке останова. Много важной информации можно получить в этом случае по команде «¤S», которая распечатывает стек вызовов процедур и функций. Распечатка содержит имя процедуры или функции и номер оператора, в котором данная процедура была вызвана. Текущая процедура распечатывается первой, затем выводится имя процедуры, которая вызвала данную и т. д.

Пример:

```
PASCAL DEBUGGER V2.2
LISTING FILE NAME? DBGTST
] ADD,1;B
] ;G
BREAKPOINT AT ADD,1 X:=P;
] ¤S
ADD,1 X:=P;
SUM,1 I:=ADD(C,D);
MAIN,3 SUM(3,4,5);
] 2;P
BREAKPOINT AT ADD,1 X:=P;
] ¤S
ADD,1 X:=P;
ADD,5 ELSE ADD:=ADD(0,X)+ADD(Y,0);
ADD,5 ELSE ADD:=ADD(0,X)+ADD(Y,0);
SUM,1 I:=ADD(C,D);
MAIN,3 SUM(3,4,5);
]
```

12.11. Распечатка последних выполненных операторов

Отладчик всегда запоминает информацию о последних 10-ти выполненных операторах. Если необходимо понять, как произошел переход в точку останова, то можно воспользоваться командой «¤L», которая распечатает номера последних 10-ти операторов, которые были выполнены, например:

```
PASCAL DEBUGGER V2.2
LISTING FILE NAME? DBGTST
```

```
] ADD,3;B
] ;G
BREAKPOINT AT ADD,3 IF Y=0 THEN ADD:=X
] ;P
BREAKPOINT AT ADD,3 IF Y=0 THEN ADD:=X
] ¤L
MAIN,2 B:=2;
MAIN,3 SUM(3,4,5);
SUM,1 I:=ADD(C,D);
ADD,1 X:=P;
ADD,2 Y:=Q;
ADD,3 IF Y=0 THEN ADD:=X
ADD,5 ELSE ADD:=ADD(0,X)+ADD(Y,0);
ADD,1 X:=P;
ADD,2 Y:=Q;
ADD,3 IF Y=0 THEN ADD:=X
]
```

13. КОНТРОЛЬНО-ДЕМОНСТРАЦИОННАЯ ЗАДАЧА

Состав контрольно-демонстрационной задачи:
DEM.PAS — программа получения суммы чисел. Результат вычисления выводится на экран.

Порядок выполнения задачи следующий:

```
.AS MY1: DK:
.RUN PASCAL
*DEM=DEM
.MACRO DEM
.R LINK
DEM=DEM,PASCAL
.RUN DEM
```

На экране появится сообщение:

ВВЕДИТЕ 5 ЧИСЕЛ ЧЕРЕЗ ПРОБЕЛЫ

Ввести пять чисел, например: 1 2 3 4 5

На экран будет выведено:

```
1  2  3  4  5
СУММА ЧИСЕЛ =  15.00
.
```

Ниже приводится исходный текст задачи на языке ПАСКАЛЬ.

```
PROGRAMM SEC (INPUT,OUTPUT);
SONST N=5;
VAR
      A    : REAL;
      SUM  : REAL
      I    : INTEGER;
BEGIN
      SUM:=0;
      WRITELN('введите ',N:1,'чисел через пробелы');
      FOR I:=1 TO N DO
       BEGIN
        READ(A);
        SUM:= SUM + A;
       END;
      WRITELN('-----------------------------------------');
      IF((SUM/SUM)=1) THEN
       WRITELN('сумма чисел =' , SUM:6:2 )
      ELSE WRITE('ошибка в программе!');
END.
```

ПРИЛОЖЕНИЕ

СПИСОК КОМАНД ОТЛАДЧИКА

¤L — вывод списка 10-ти последних операторов, которые были выполнены.

¤S — распечатка списка (стека) вызванных процедур и функций.

¤V — распечатка списка имен переменных, доступных в данной точке программы.

PROC, STMT;B — установить точку останова на операторе «STMT» в процедуре с именем «PROC».

0;B — отменить точку останова.

;P — продолжить выполнение программы после точки останова.

N;P — пропустить точку останова «N» раз или выполнить «N» операторов в пошаговом режиме.

;G — начать выполнение программы (аналогично «;P»).

1;T — установить режим трассировки выполняемых операторов.

0;T — отменить режим трассировки выполняемых операторов.

1;C — установить режим трассировки вызовов процедур и функций.

0;C — отменить режим трассировки вызовов процедур и функций.

1;S — установить пошаговый режим выполнения программы.

0;S — отменить пошаговый режим выполнения программы, т. е. перейти в обычный режим.

VAR;W — установить слежение за изменением значения переменной «VAR».

—1;W — отменить слежение за значением переменной.

VAR/ — распечатать значение переменной типа INTEGER.

VAR/N — распечатать «N» значений целочисленных переменных (массива).

VAR\ — распечатать значение переменной в десятичном виде (как байта).

VAR\N — распечатать «N» последующих байтов в десятичном виде.

VAR'% — распечатать значение переменной типа REAL.

VAR'%N — распечатать «N» значений переменных с плавающей запятой.

VAR' — распечатать значение переменной типа CHAR в — символьном виде.

VAR'N — распечатать «N» последующих значений переменных в символьном виде.

VAR= — распечатать абсолютный адрес переменной.

VAR:=NUMBER — присвоить целочисленное значение (NUMBER) переменной «VAR».

VAR:='TEXT' — присвоить символьную строку переменной.

VAR:=RNUMBER — присвоить значение с плавающей запятой (NUMBER) переменной.

«+», «.» — операторы сложения.

«—» — оператор вычитания.

«^» — оператор ссылки (косвенная адресация).

БЕЙСИК

ОПИСАНИЕ ЯЗЫКА

1. ОБЩИЕ СВЕДЕНИЯ И СПОСОБ ОПИСАНИЯ ЯЗЫКА

1.1. Назначение языка

Язык БЕЙСИК является одним из простых в изучении языков программирования.

Программа на языке БЕЙСИК представляет собой набор операторов, объединенных в логические блоки.

Диалоговый режим интерпретатора БЕЙСИК облегчает отладку программ, обмен память — внешнее устройство (и наоборот).

1.2. Условные обозначения

В данном документе используются следующие условные обозначения:

1) [ ] — квадратные скобки	Элементы внутри скобок необязательны;
2) Элементы из прописных латинских букв и специальных символов	Ключевые слова. Должны указываться точно такими, какие они есть в формате. Пример: LET RUN # ;
3) Элементы из набора латинских букв	Следуют за ключевым словом. Необходимо заменить данные элементы согласно описанию, даваемому в тексте.

В табл. 1 приведены элементы из набора латинских букв, часто встречающихся в описании формата операторов.

Таблица 1

Мнемоника	Обозначение	Смысловая нагрузка
1	2	3
N, NUM	Номера строк	Номера строк
EXP	Выражение	Допустимое в БЕЙСИК выражение. Может быть числовым или строковым
STR	Строка	Строковое выражение
LIST	Список элементов	Элементы из констант, переменных, массивов
VAR	Переменная	Вещественная, целая или строковая переменная
CONST	Константа	Вещественная, целая или строковая константа

2. ЭЛЕМЕНТЫ И ОСНОВНЫЕ КОНСТРУКЦИИ ЯЗЫКА

2.1. Набор символов языка БЕЙСИК

Все используемые символы рассматриваются как алфавит языка БЕЙСИК. Алфавит включает:

1) прописные (латинские) буквы от A до Z;
2) буквы кириллицы от А до Я;
3) цифры от 0 до 9;
4) *специальные символы:*

Пробел
» Кавычки
Номер
$ Знак денежной единицы
% Процент
& Коммерческое «И»
' Апостроф
(Круглая скобка левая
) Круглая скобка правая
+ Плюс
, Запятая
. Точка
/ Дробная черта
; Точка с запятой
< Меньше
= Равно
> Больше
? Вопросительный знак

@ Коммерческое «ЭТ»
\ Обратная дробная черта
^ Стрелка вверх

5) непечатаемые символы.

Буквы кириллицы могут использоваться как часть текстовой константы или буквенно-цифрового литерала. В комментариях допускается использование любых печатных символов.

2.2. Номера строк

Каждая строка программы на языке БЕЙСИК должна начинаться номером. Номера строк необходимы для:

— указания интерпретатору БЕЙСИК порядка выполнения программы;
— изменения обычного порядка выполнения операторов для обеспечения ссылок при условной и безусловной передаче управления;
— модификации отдельных строк и отладки программ.

Номер — положительное целое число в диапазоне от 1 до 32767.

Рекомендуется использовать номера строк с шагом 10. Это позволяет вставлять дополнительные операторы между строками.

Пример:

```
10 A=5
20 B=10
30 C=A+B
40 END
```

В примере присваиваются значения переменным «А» и «В», они суммируются, результат помещается в «С».

2.3. Операторы

Оператор языка БЕЙСИК в программе следует за номером строки. Оператор указывает интерпретатору, что должно быть выполнено, и как следует воспринимать данные (если они есть), следующие за этим оператором.

Операторы делятся на две группы:

1) выполняемые операторы — определяют действия программы, указывая интерпретатору БЕЙСИК, какую операцию нужно выполнить (PRINT, GO TO, READ);

2) невыполняемые операторы — описывают характер и упорядочение данных, позволяют вводить в программу примечания и сообщения описательного характера (DATA, REM).

В строку можно записать несколько операторов, каждый из которых (за исключением последнего) отделяется от последующего с помощью обратной дробной черты. При этом

номер строки ставится только перед первым оператором.

Запись заканчивается нажатием клавиши <ВК>.

Пример:

```
10 LET A=B*C/D \ PRINT A
```

2.4. Пробелы и табуляция

В языке БЕЙСИК разрешается использование пробелов и знаков табуляции для удобства чтения программ.

Пример:

Использование пробелов.

Запись 10 LET A=B*2+1 читается лучше, чем

```
10LETA=B*2+1 или
10 L E T A = B * 2 + 1
```

Пример:

Использование табуляции.

```
10 FOR N=1 TO 3
20      FOR I=1 TO 5
30          FOR J=1 TO 10
40          A(I,J)=N/(I+J-1)+A(I,J)
50          NEXT J
60      NEXT I
70 NEXT N
```

2.5. Выполнение операторов в программном и непосредственном режимах

БЕЙСИК выполняет операторы в двух режимах: программном и непосредственном. В программном режиме каждой строке предшествует номер, указывающий последовательность ее выполнения в программе. В непосредственном режиме номер строки отсутствует; оператор выполняется сразу же после нажатия клавиши <ВК>.

Сообщение READY указывает на готовность системы принять информацию в непосредственном режиме. Строки можно набирать в любом порядке. БЕЙСИК размещает их в порядке возрастания номеров строк.

Для замены строки следует набрать новую строку под тем же номером.

Пример:

Старая строка	Новая строка
30 A=B+C	30 A=B*C

Для удаления строки следует набрать ее номер и нажать клавишу <ВК>. Удаление набираемого символа производится клавишей <ЗБ>. Отказ от набираемой строки производится нажатием СУ/U (описание приведено в п. 7.3.5).

Операторы БЕЙСИК, используемые в непосредственном режиме как команды, набираются без номера строки и называются операторами непосредственного режима.

Пример:

Строку подпрограммы 10 PRINT 4+5 БЕЙСИК поместит в память для более позднего выполнения. Но если набрать ее без номера, БЕЙСИК выполнит строку и выдаст сообщение READY.

```
PRINT 4+5
9
READY
```

Разрешается запись нескольких операторов непосредственного режима в одной строке. Операторы разделяются символом «\».

Пример:

```
A=5\B=20\C= .3729\PRINT A*B, SIN(C)
100          .364318
READY
```

Непосредственный режим обычно используется при отладке программ.

Например, оператор PRINT в непосредственном режиме позволяет узнать текущее значение переменных в программе.

3. ЭЛЕМЕНТЫ ДАННЫХ ЯЗЫКА

3.1. Константы

В языке БЕЙСИК используются три типа констант:

1) вещественные константы;
2) целые константы;
3) строковые константы.

3.1.1. Вещественная константа — это набор из одной или нескольких десятичных цифр. Константа может принимать как положительные, так и отрицательные значения.

Отсутствие точки в константе предполагает наличие ее за последней цифрой справа.

Пример:

184 равносильно 184.

Пример:

Допустимые вещественные константы.

```
 5          42861
74          —125
 6.           .95
```

БЕЙСИК разрешает оперировать с вещественными константами в диапазоне от 10^(—38) до 10^(+38).

Использование в программе констант вне указанного диапазона вызывает сообщение о неустранимой ошибке.

При работе с очень большими и очень малыми числами БЕЙСИК разрешает использование экспоненциального представления, то есть число (в диапазоне от 1 до 10), умноженное на десять в степени:

[—]X.XXXXXE[—]NN

где

(—) — знак числа;
X — цифра от 0 до 9;
. — десятичная точка;
E — используемый формат;
NN — порядок.

Числовые константы могут иметь вид:

.73012E—02
9.99
—1
—8.3067

Числа с плавающей запятой занимают в памяти либо два, либо четыре 16-разрядных слова. Значение, вычисленное с одинарной точностью, занимает два слова (8 десятичных цифр), а вычисленное с двойной точностью (15 десятичных цифр) — четыре слова.

При вычислениях с двойной точностью необходимо учитывать:

— константы, переменные и массивы требуют вдвое больше памяти, чем те же элементы с одинарной точностью;
— арифметические операции и функции с числами двойной точности выполняются медленнее, чем те же операции над числами с одинарной точностью.

Оба формата позволяют представлять числа в диапазоне от 10^(—38) до 10^(+38).

БЕЙСИК выполняет вывод результатов при вычислениях с плавающей запятой:

— с помощью оператора PRINT (печатается шесть цифр);
— с помощью оператора PRINT USING и функции STR$ (печатаются числа в заданном формате).

3.1.2. Целая константа — это набор десятичных цифр, оканчивающихся знаком процента «%».

Диапазон использования целых констант от —32768% до +32767%. Операции с целыми константами вне этого диапазона вызывают сообщение о неустранимой ошибке.

3.1.3. Строковая константа (литерал) — это набор буквенно-цифровых и/или специальных символов. Строковые кон-

станты ограничиваются с двух сторон одиночными или двойными кавычками.

БЕЙСИК печатает каждый символ внутри кавычек точно так, как он представлен в исходной программе.

Начальный и конечный ограничители должны быть одного типа. Возможна комбинация двойных и одинарных кавычек.

Пример:

PRINT 'пункт «строковые константы»'

Результат работы оператора:

пункт «строковые константы»

3.2. Переменная — это величина, значение которой может изменяться в процессе выполнения программы.

БЕЙСИК допускает три типа переменных:

1) вещественные переменные;

2) целые переменные;

3) строковые переменные.

3.2.1. Вещественная переменная обозначается одной буквой или буквой, за которой следует одна цифра.

В начале программы, написанной на языке БЕЙСИК, необходимо присвоить всем переменным нулевое значение.

Целочисленное значение, присвоенное вещественной переменной, БЕЙСИК печатает как целое, но во внутреннем представлении рассматривает его как число с плавающей запятой.

3.2.2. Целые переменные

Идентификатор целой переменной состоит из одной буквы или из одной буквы и одной цифры и заканчивается знаком процента «%».

Пример:

Целые переменные:

A% C8%

B1% D%

Значение целых переменных должно находиться в диапазоне от —32768 до +32767.

Если вещественная константа присваивается целой переменной, то БЕЙСИК отбрасывает дробную часть значения.

3.2.3. Строковые переменные

Идентификатор строковой переменной — это буква, за которой может следовать одна цифра. Знак «$» должен быть последним символом в идентификаторе строковой переменной.

Пример:

Допустимые имена	Недопустимые имена
B1$	B1
A$	A
S2$	2S$
I$	$I
C6$	C6

Длина символьной строки, соответствующей строковой переменной, может изменяться от 0 (пустая строка) до 255.

ПРИМЕЧАНИЕ. Вещественная переменная, целая переменная и строковая переменная, имена которых начинаются с одних и тех же буквенно-цифровых символов, представляют три различных переменных.

Пример:

A5 — вещественная переменная.
A5% — целая переменная.
A5$ — строковая переменная.

3.3. Массивы

Имя массива БЕЙСИК — переменная, за которой следуют один или два индекса (в диапазоне от 0 до 32767), заключенные в скобки.

Пример:

одномерный массив A(I) из шести элементов и его распределение в памяти:

A(0) A(1) A(2) A(3) A(4) A(5)

Пример:

двухмерный массив M1(I,J) из 9 элементов и его распределение в памяти:

M1(0,0) хранится в ячейке L
M1(0,1) хранится в ячейке L+1
M1(0,2) хранится в ячейке L+2
M1(1,0) хранится в ячейке L+3
M1(1,1) хранится в ячейке L+4
.
.
.
M1(2,2) хранится в ячейке L+8

Если пользователь не зарезервировал память для массива (оператором DIM), БЕЙСИК выделяет место в памяти для одномерного массива из 11 элементов, для двухмерного массива из 121 элемента.

В программе можно использовать переменную и массив с одним и тем же именем.

Пример:

A — переменная,
A(5) — массив.

Недопустимо использование одного и того же имени для одномерного и двухмерного массивов одновременно. Использование массива взамен большого числа переменных приводит к экономии памяти.

Пример:

Программа с массивом

```
5 DIM A(8)
10 FOR I=0 TO 8
20 LET A(I)=I
30 NEXT I
```

Программа с отдельными переменными

```
5 A0=0
10 A1=1
15 A2=2
20 A3=3
25 A4=4
30 A5=5
35 A6=6
40 A7=7
45 A8=8
```

4. ВЫРАЖЕНИЯ

Выражение — это константы, переменные, индексные переменные, функции или любые комбинации их, образованные с помощью:

— арифметических операций;
— операций отношения;
— операций над строками.

4.1. Арифметические выражения

Для образования арифметических выражений используются следующие арифметические операции:

∧ — возведение в степень;
∗ — умножение;
/ — деление;
+ — сложение;
— — вычитание.

Выполнение арифметических операций над арифметическими выражениями одного типа дает результат того же типа.

Пример:

A%+B% = целому выражению.
C1∗D = вещественному выражению.

Сочетание целой величины с вещественной дает вещественный результат.

Пример:

A∗B% = вещественному выражению.
8.8∗8% = 70.4

Два знака арифметических операций нельзя располагать рядом. Исключение составляет унарный минус.

Пример:

A∗—B — правильно,
A∗(—B) — правильно.

БЕЙСИК вычисляет значение выражения согласно приоритету операций:

1) возведение в степень (∧) — наивысший;
2) унарный минус (—);
3) умножение, деление (∗,/);
4) сложение, вычитание (+,—) — низший.

При использовании скобок БЕЙСИК вычисляет первым самое внутреннее выражение, затем внешнее по отношению к предыдущему и так далее.

Пример:

B=(20∗(16+(8∧2)))

БЕЙСИК вычисляет первым (8∧2), затем (16+64) и, наконец, (20∗80).

БЕЙСИК вычисляет выражение в скобках в первую очередь, даже если операция в скобках имеет более низкий приоритет, чем операция вне скобок.

Операции, размещенные в одной строке и имеющие равный приоритет, выполняются последовательно слева направо.

Пример:

A∧B∧C соответствует (A∧B)∧C.

4.2. Строковые выражения состоят из строковых констант, строковых переменных, строковых функций и/или любой возможной комбинации, разделяемой знаками операций.

В качестве знаков операций над строковыми выражениями БЕЙСИК использует знак «+» (плюс) и знак «&» (амперсанд). Знаки операций служат для объединения строк.

A$+B$ и A$&
B$ означают присоединение строки B$ к концу строки A$. Результирующая строка не должна превышать 255 символов.

Пример:

```
10 D$="ПРИ"&"СОЕДИНЕНИЕ"
20 PRINT D$
30 END
```

Результат, выводимый на терминал:
ПРИСОЕДИНЕНИЕ

4.3. Операции отношения служат для установления отношений между двумя величинами или двумя строками. Сравнение и передача управления осуществляется, в основном, операторами IF THEN (IF GOTO). Допустимо сравнение целого и вещественного выражений. Знаки операций отношения приведены в табл. 2.

Таблица 2

Операция	Пример	Комментарии
1	2	3
=	A=B	A равно B
	A$=B$	Строки A$ и B$ равны
<	A<B	A меньше B
	A$<B$	Символьная строка A$ меньше символьной строки B$
>	A>B	A больше B
	A$>B$	Символьная строка A$ больше символьной строки B$
<=,=<	A<=B	A меньше или равно B
	A$<=B$	Символьная строка A$ либо эквивалентна символьной строке B$, либо меньше
>=,=>	A>=B	A больше или равно B
	A$>=B$	Символьная строка A$ либо эквивалентна символьной строке B$, либо больше
<>,><	A<>B	A не равно B
	A$<>B$	Символьные строки A$ и B$ не эквивалентны друг другу

Сравнение строк проводится посимвольно слева направо, в порядке расположения символов в таблице кодов КОИ—7.

Пример:

```
10 A$="ABCD"
20 B$="ABCF"
30 IF A$<B$ GOTO 50
40 PRINT B$\GOTO 60
50 PRINT A$
60 END
```

В строке 30 происходит сравнение символьных строк A$ и B$ и передача управления строке 50, если условие выполняется. В противном случае, выполнение программы продолжится со строки 40.

Сравнивая символьные строки разной длины, БЕЙСИК дополняет короткую строку пробелами.

5. ОПЕРАТОРЫ И ФУНКЦИИ ЯЗЫКА БЕЙСИК

5.1. Операторы

5.1.1. Оператор REM. Оператор REMARK используется для ввода примечаний и комментариев в программу пользователя.

Формат оператора REM: [N] REM COMMENT
где N — номер строки;
COMMENT — текст примечаний и комментариев.

Примечания и комментарии печатаются при получении листинга программы пользователя и не влияют на выполнение программы.

Необходимо учитывать при написании больших программ, что оператор REM (с текстом) занимает место в памяти.

5.1.2. Оператор DIM. Оператор DIM служит для резервирования места в памяти для числовых или строковых массивов.

Формат оператора DIM: [N] DIM LIST
где N — номер строки;
LIST — список массивов, разделенных запятыми.

Пример:

```
10 DIM A(36),B(4,6),C%(8,18),D$(20)
```

Двумерный массив размещается в памяти по строкам, то есть второй индекс изменяется быстрее первого.

Пример:

Оператор DIM(3,4) определяет следующее размещение массива в памяти:

```
0.0 0.1 0.2 0.3 0.4 1.0 1.1 1.2 ... 3.3 3.4
```

Это соответствует матрице:

```
0.0  0.1  0.2  0.3  0.4
1.0  1.1  1.2  1.3  1.4
2.0  2.1  2.2  2.3  2.4
3.0  3.1  3.2  3.3  3.4
```

Пример:

Определение размерности строковых массивов.

```
10 DIM T$(4)
20 FOR I=0 TO 4
30 INPUT T$(I)
40 NEXT I
50 END
```

Указывая в операторе DIM максимальное значение индексов, не обязательно заполнять значениями все поле отведенной памяти.

Оператор DIM является невыполняемым оператором. Его можно поместить в любом месте программы и в любой части многооператорной строки.

5.1.3. Оператор LET. Оператор LET присваивает значение переменной.

Формат оператора LET: [N] [LET] VAR = EXP
где N — номер строки;
LET — необязательное имя оператора;
VAR — переменная, принимающая новое значение;
EXP — выражение, определяющее новое значение.

Переменная и выражение должны быть оба числовыми или оба строковыми.

Пример:

```
10 LET A = 482.5 или 10 A= 482.5
```

БЕЙСИК вычисляет значение любой формулы.

Пример:

```
10 B(I) = (A(0)+A(I))/2
```

Оператор LET можно поместить в любой части многооператорной строки.

Пример:

```
10 DIM A(7)\I=40\PRINT I
```

5.1.4. Оператор INPUT. Оператор INPUT позволяет вводить и анализировать данные в процессе выполнения программы.

Формат оператора INPUT: [N] INPUT VAR1[,VAR2,VAR3,...]
где N — номер строки;
VAR1,VAR2,VAR3,... — элементы данных, которым присваиваются вводимые значения.

Встретив оператор INPUT, БЕЙСИК печатает знак «?» и ждет ввода значений для каждой переменной. Значения разделяются запятыми. При нажатии клавиши <ВК> производится ввод данных и анализ их на соответствие типа переменных вводимым значениям. БЕЙСИК вводит значения слева направо до ограничителя строки «ВК».

Пример:

```
10 INPUT C1,C$
RUNNH
?50
?ЛЕТ
READY
```

Число вводимых данных должно соответствовать числу переменных в операторе INPUT. Если введено недостаточное количество данных, БЕЙСИК печатает знак «?». Избыток вводимых значений вызовет предупреждение об ошибке. Лишние значения игнорируются.

Строковую переменную (или массив) допускается вводить без ограничивающих кавычек. Однако, если в тексте имеется символ «запятая», то наличие кавычек обязательно. В этом случае игнорируются начальные и конечные пробелы.

С оператором INPUT используется оператор PRINT для информации о количестве переменных и их типе (числовая или строковая).

Пример:

```
10 PRINT «введите 3 целых числа A,B,C и одну символьную строку J»
20 INPUT A%,B%,C%,J$
30 END
```

Дополнительная информация об операторах INPUT и LINPUT приведена в пункте 6.3.1.

5.1.5. Оператор LINPUT. Оператор LINPUT соответствует оператору INPUT. Но оператор LINPUT используется только для строковых данных.

Формат оператора LINPUT:

[N] LINPUT STR—VAR1[,STR—VAR2,...]

где N — номер строки;

STR—VAR1,STR—VAR2,... — строковые переменные.

Оператор LINPUT принимает строку входной информации (с терминала), включая начальные, конечные, промежуточные пробелы, символы пунктуации и кавычки. Если разделитель «,» встречается внутри текста вводимой переменной, то текст ограничивается разделителями «или'.

5.1.6. Операторы READ, DATA, RESTORE. Операторы READ и DATA используются для организации блока данных, который считывается интерпретатором БЕЙСИК во время выполнения программы.

Формат оператора READ:

[N] READ VAR1[,VAR2,VAR3,...]

где N — номер строки;

VAR1,VAR2,VAR3,... — переменные, которым присваиваются значения из списка оператора DATA.

Пример:

```
10 READ A,B%,C$,D(5)
```

Формат оператора DATA:

[N] DATA CONST1[,CONST2,CONST3,...]

где N — номер строки;

CONST1,CONST2,CONST3,... — вещественная, целая или строковая (в кавычках или без) константа.

Перед выполнением программы БЕЙСИК просматривает все операторы DATA в порядке их появления и создает блок данных. Каждый раз, когда в программе встречается оператор READ, блок данных выдает последовательно соответствующее значение для переменных этого оператора в том порядке, в котором они заданы в блоке данных.

После выполнения оператора READ положение последних считанных данных запоминается (с помощью указателя). Следующий оператор READ начинает выбирать данные с той позиции, которая была установлена предыдущим оператором READ.

Пример:

Использование операторов READ и DATA:

```
10 READ A;B%,C$,C1$
20 DATA 3.5,—6,«режим», работы
```

БЕЙСИК присваивает значения в следующем порядке:

A=3.5
B%= —6
C$=режим
C1$=работы

Оператор READ допускается помещать в любом месте многооператорной строки.

Оператор DATA должен быть либо единственным, либо последним оператором строки.

БЕЙСИК допускает чтение числовой константы в строковую переменную.

Пример:

```
10 READ A$
20 DATA 15.15
```

Попытка считать строковую константу в числовую переменную вызовет сообщение об ошибке.

При выполнении оператора READ указатель считывания данных перемещается. БЕЙСИК осуществляет повторное считывание одних и тех же данных с помощью оператора RESTORE.

Формат оператора RESTORE:

[N] RESTORE

где N — номер строки.

Пример:

```
10 READ A,B,C
20 RESTORE
30 READ D,E,F
```

```
40 DATA 2,4.5,6,8,10
50 END
```
Оператор READ в строке 10 читает первые три значения из оператора DATA в строке 40.

A=2
B=4.5
C=6

Затем оператор RESTORE (строка 20) перемещает указатель данных в начало строки 40 так, что второй оператор READ в строке 30 читает первые три значения.

D=2
E=4.5
F=6

5.1.7. Оператор PRINT. Оператор PRINT выводит данные на терминал в процессе выполнения программы.

Формат оператора PRINT: [N] PRINT [LIST]
где N — номер строки;
LIST — список элементов, представленных в виде констант, переменных, строковых или числовых выражений, или функцией TAB и разделяемых запятой или точкой с запятой.

Оператор PRINT без списка аргументов выводит строку пробелов.

Если элементом списка является выражение, БЕЙСИК вычисляет его и печатает результат.

Пример:

```
10 A1=25\A2=35
20 PRINT «A1+A2= »;A1+A2
```
Результат, выводимый на терминал: A1+A2= 60

В строке 20 строковая константа отделяется от выражения точкой с запятой. Это позволяет печатать результат рядом с символьной строкой.

БЕЙСИК выполняет операцию возврата каретки и перевода строки после каждого оператора PRINT.

Строка символов, выводимая на терминал, состоит из 5 зон по 14 позиций в каждой зоне.

Когда элементы в списке оператора PRINT разделены запятыми, каждый последующий элемент печатается в следующей свободной зоне.

Если последняя зона в строке заполнена, БЕЙСИК продолжает печать с первой зоны следующей строки.

Наличие в списке оператора PRINT двух рядом стоящих запятых вызывает пропуск одной зоны.

Точка с запятой в качестве разделителя элементов списка

оператора PRINT запрещает продвижение печатаемого символа.

5.1.7.1. Формат вывода чисел и символьных строк

БЕЙСИК печатает числа и строки по определенному формату.

Строки печатаются в том виде, в каком представлено строковое выражение в операторе PRINT (ограничивающие кавычки не печатаются. Если строковое выражение не заключено в кавычки, то начальные и конечные пробелы не печатаются).

Пример:

PRINT 'ответ «Да» или «Нет»'
Результат работы оператора PRINT:
ответ «Да» или «Нет»

Перед отрицательным числом ставится знак минус, а перед положительным — пробел.

При печати целых чисел знак процента «%» не печатается.

Пример:

```
10 PRINT  —4
20 PRINT 15;;;30%
```

Сообщение, выводимое на терминал:

```
—4
  15   30
```

БЕЙСИК печатает результаты вычислений в виде десятичного числа (целого или с плавающей запятой), если эти результаты находятся в интервале (0.01;999999). В остальных случаях БЕЙСИК использует экспоненциальное представление.

Пример:

Вводимое значение	Значение, печатаемое БЕЙСИК
.000789	7.8900E—04
.03	.03
888889	888889
1000000	1.00000E+06

5.1.7.2. Функция TAB

Формат функции TAB: [N] PRINT TAB(EXP)

где N — номер строки;

EXP — задаваемый номер позиции печати. Принимает значения от нуля до номера самой правой позиции строки терминала.

Функция TAB позиционирует положение символа строки при печати. Оператор PRINT начинает печать с позиции, ука-

занной аргументом функции TAB. Если указана позиция, находящаяся левее текущей, БЕЙСИК игнорирует требование функции **TAB**.

Пример:

```
10 A=123\B=45\C=6
20 PRINT TAB(5);A,TAB(15);B;TAB(23);C
```

Числа располагаются следующим образом:

```
123         45          6
```

БЕЙСИК игнорирует TAB(10), так как запятая требует печати в зоне, которая находится за позицией, указанной в TAB.

5.1.8. Оператор GO TO. Оператор GO TO вызывает непосредственный переход к указанной строке с нарушением естественного порядка выполнения операторов программы.

Формат оператора GO TO: [N] GO TO LINE NUMBER

где N — номер строки;

LINE NUMBER — номер строки, к которой осуществляется переход

Пример:

```
10 A=9\GO TO 30
20 A=A*2
30 PRINT A,A*A
40 END
```

Операторы выполняются в следующей последовательности:

Присвоение переменной «А» значения 9 и переход к строке 30 (строка 10);

Выполняется оператор PRINT (строка 30);

Завершение программы в строке 40.

Строка 20 не выполняется.

Оператор GO TO должен быть либо единственным оператором строки, либо последним оператором многооператорной строки.

5.1.9. Оператор ON GOTO (ON THEN). Оператор ON GOTO (ON THEN) позволяет осуществлять переход к одной из указанных строк в зависимости от значения числового выражения.

Формат оператора ON GOTO (ON THEN):

[N] ON EXP GOTO LINE NUM1[,LINE NUM2,...]

[N] ON EXP THEN LINE NUM1[,LINE NUM2,...]

где N — номер строки;

EXP — любое допустимое арифметическое выражение;

LINE NUM1,LINE NUM2,... — номера строк перехода.

Ключевые слова GOTO и THEN взаимозаменяемы.

При выполнении оператора ON GOTO (ON THEN) вы-

числяется прежде всего значение числового выражения. Целая часть его используется в качестве указателя на один из перечисленных номеров строк в списке. Если результирующее значение равно 1, передается управление строке, номер которой расположен первым в списке. Если значение равно 2, управление передается строке, номер которой записан вторым в списке и т. д. Если значение выражения меньше 1 или больше количества номеров строк в списке, БЕЙСИК печатает сообщение об ошибке.

Пример:

200 ON A GOTO 50,20,100,300

A=1 — переход к строке 50,
A=2 — переход к строке 20,
A=3 — переход к строке 100,
A=4 — переход к строке 300,
A<1 или A>4 — печатается сообщение об ошибке.

5.1.10. Операторы IF THEN и IF GOTO. Операторы IF THEN и IF GOTO передают управление в зависимости от истинности (или ложности) выражения отношения.

Формат оператора IF THEN:

[N] IF REL — EXP THEN LINE NUMBER
или [N] IF REL —EXP THEN STATEMENT

где N — номер строки;
REL—EXP — проверяемое условие; выражение отношения может быть как арифметическим, так и строковым;
LINE NUMBER — номер строки, выполняемой в случае истинности условия;
STATEMENT — оператор, подлежащий выполнению. Может быть оператор IF THEN;

Формат оператора IF GOTO:

[N] IF REL—EXP GOTO LINE NUMBER

где N — номер строки;
REL—EXP — проверяемое условие; выражение отношения может быть как арифметическим, так и строковым;
LINE NUMBER — номер строки, выполняемой в случае истинности условия.

Если в операторе IF THEN(IF GOTO) после ключевого слова THEN(GOTO) следует номер строки и условие истинно, управление передается этой строке.

Пример:

```
10 INPUT C
20 IF C=0 THEN 30\PRINT C\GOTO 40
```

```
30 PRINT «C= »;C
40 END
```

Если в операторе IF THEN после ключевого слова THEN следует оператор и условие истинно, то выполняется оператор, следующий за THEN, и операторы, следующие за оператором IF THEN в этой строке (если они есть). Если условие ложно, то управление передается строке, следующей за строкой с оператором IF THEN.

Пример:

```
10 INPUT N
20 IF N=0 THEN PRINT «Истина »;\PRINT N\GOTO 40
30 PRINT «N= »;N\GOTO 50
40 PRINT «Ветка N=0»
50 END
```

Пример:

```
10 INPUT A1,A2,A3
20 IF A1>A2 THEN IF A2<A3 THEN PRINT A2;\GOTO 40
30 PRINT «Сравнения нет»
40 END
```

В последнем примере строка 30 выполняется, если A1 меньше или равно A2, или A2 больше или равно A3.

5.1.11. Оператор FOR и NEXT. Операторы FOR и NEXT позволяют организовать цикл так, что БЕЙСИК автоматически проверяет условие при каждом проходе.

Формат оператора FOR:

[N] FOR VAR=EXP1 TO EXP2 [STEP EXP3]

где N — номер строки;
VAR — управляющая переменная (индекс цикла);
EXP1 — начальное значение индекса, любое числовое выражение;
EXP2 — конечное значение индекса, любое числовое выражение;
EXP3 — приращение величины индекса (шаг), может быть положительное или отрицательное числовое выражение (по умолчанию равен 1).

Операторы FOR и NEXT используются только в паре. Оператор FOR определяет начало цикла, оператор NEXT — конец цикла.

Формат оператора NEXT: [N] NEXT VAR

где N— номер строки;
VAR — переменная, должна совпадать с индексом цикла, указанным в операторе **FOR.**

Пример:

```
20 FOR I=3 TO 30 STEP 3
```

```
30 A(I)=I\PRINT A(I)
40 NEXT I
```

Заданное начальное значение I равно 3, и БЕЙСИК производит проверку: превосходит ли значение I конечное значение 30 или нет. Цикл выполняется, если I по значению меньше или равно 30.

Рекомендации по работе с операторами FOR и NEXT:

1) если начальное значение индексной переменной больше конечного значения (при положительном шаге), цикл не выполняется;

2) передавать управление внуть цикла недопустимо.

Пример:

```
10 J=5
20 FOR J=1 TO J*4
30 NEXT J
```

БЕЙСИК вычисляет в строке 20 величину J*4 до присвоения переменной J значения 1. Во избежании ошибок рекомендуется использовать последовательность:

```
20 FOR J=1 TO 5*4
30 NEXT J
```

3) для строгого соблюдения количества циклов рекомендуется использовать целые значения индексов. Это связано с двоичным представлением чисел. БЕЙСИК допускает и такое использование:

```
10 FOR I=1.5 TO 7.7 STEP 1.32
```

4) циклы, задаваемые с помощью оператора FOR, допускается вкладывать друг в друга. Перекрытие циклов недопустимо. Вложенный цикл должен иметь собственные операторы FOR и NEXT. Внутренний цикл должен заканчиваться до окончания внешнего.

Пример:

Построение вложенных циклов

Допустимое

```
10 FOR I%=1% TO 5%
20 FOR J= 0 TO 4
30 NEXT J
40 NEXT I%
5  FOR A%=1% TO 5%
10 FOR B=1 TO 10
15 NEXT B
20 FOR C%=1% TO 5%
30 FOR D=5 TO 50 STEP 5
40 NEXT D
50 NEXT C%
60 NEXT A%
```

Недопустимое

```
10 FOR I=1 TO 10
20 FOR J=2 TO 11
30 NEXT I
40 NEXT J
```

5.1.12. Операторы END и STOP. Операторы STOP и END используются для останова и завершения выполнения программы.

Формат оператора END: [N] END
где N — номер строки.

Оператор END является последним в программе. При отсутствии операторов END и STOP оператор, выполняемый последним, завершает работу программы и закрывает все файлы.

Формат оператора STOP: [N] STOP
где N — номер строки.
Оператор вызывает останов программы и вывод сообщения:
STOP AT LINE N (останов на строке N)
READY
где N — номер строки с оператором STOP.

БЕЙСИК переходит в режим редактирования. Останов программы позволяет пользователю распечатать значения переменных, изменить их значения, то есть оператор STOP является удобным средством для отладки программ. Место и количество используемых операторов STOP в программе на языке БЕЙСИК не ограничено. Продолжить выполнение программы можно с помощью оператора GO TO в диалоговом режиме, указав номер строки, с которой необходимо продолжить программу.

Оператор STOP приостанавливает выполнение программы, но не закрывает файлы.

5.1.13. Операторы GOSUB и RETURN. Операторы GOSUB и RETURN осуществляют связь программы с подпрограммой.

Формат оператора GOSUB: [N[GOSUB LINE NUMBER
где N — необязательный номер строки;
LINE NUMBER — номер строки, точка входа в подпрограмму.

Встретив в программе оператор GOSUB, БЕЙСИК передает управление строке подпрограммы, заданной в операторе GOSUB. Оператор выполняет подпрограмму (с заданной строки) пока не встретится оператор выхода из подпрограммы (оператор RETURN).

Формат оператора RETURN: [N] RETURN
где N — номер строки.

БЕЙСИК организует таблицу адресов возврата. Всякий раз, когда выполняется GOSUB, БЕЙСИК помещает в таблицу адрес строки, следующий за оператором GOSUB. Таблица вмещает не более 20 адресов строк.

Пример:
Использование операторов GOSUB и RETURN.

```
10 PRINT «Введите коэффициенты. Сначала A, потом B,C»
20 PRINT «При A=0 программа идет на END»
30 INPUT A
40 IF A=0 THEN 32767
50 INPUT B,C
60 IF A<>1 THEN 350
70 GOSUB 200
80 GOTO 10
200 PRINT «Уравнение вида X^2+B*X+C=0»
210 D1=(B/2)^2—C
220 IF D1<>0 THEN 250
230 PRINT «Имеет одно решение X= »;—B/2·
240 RETURN
250 IF D1<0 THEN 280
260 PRINT «Имеет два решения X1= »;—B/2+SQR(D1);»
265 PRINT « X2= »;—B/2—SQR(D1)
270 RETURN
280 PRINT «Имеет мнимые корни X1= »;
285 PRINT —B/2;«+»;SQR(—D1);«*I;»;
290 PRINT " X2= ";—B/2;"—";SQR(—D1);"*I"
300 RETURN
350 GOSUB 400
360 GOTO 10
400 PRINT «Уравнение вида A*X^2+B*X+C=0»
410 D=B*B—4*A*C
420 IF D<>0 THEN 450
430 PRINT «Имеет одно решение X= »;—B/(2*A)
440 RETURN
450 IF D<0 THEN 490
460 PRINT «Имеет два решения X1=»;(—B+SQR(D))/(2*A);
470 PRINT "; X2= ";(—B—SQR(D))/(2*A)
480 RETURN
490 PRINT «Имеет мнимые корни X1=»;
500 PRINT —B/(2*A);"+";SQR(—D)/(2*A);"*I";
510 PRINT " X2= ";—B/(2*A);"—";SQR(—D)/(2*A);"*I"
520 RETURN
32767 END
```

5.1.14. Оператор ON GOSUB. Оператор ON GOSUB используется для условной передачи управления одной из не-

скольких подпрограмм или к одной из нескольких входных точек подпрограмм.

Формат оператора ON GOSUB:

[N] ON EXP GOSUB LINE NUM1[,LINE NUM2,...]

где

EXP — любое выражение, допустимое в БЕЙСИК (кроме выражения отношения);

LINE NUM1, LINE NUM2,... — список номеров строк перехода.

Оператор ON GOSUB аналогичен оператору ON GOTO (см. пункт 5.1.9.).

Пример:

50 ON A+B GOSUB 200,100,20

A+B=1 — управление передается строке 200,

A+B=2 — управление передается строке 100,

A+B=3 — управление передается строке 20,

A+B<1 или A+B>3 — БЕЙСИК печатает сообщение об ошибке.

Оператор ON GOSUB позволяет передавать управление в любую точку подпрограммы.

5.2. Функции

5.2.1. Допустимые типы функций

БЕЙСИК содержит набор математических (приведены в табл. 3) и строковых функций (включая функции текущей даты и времени) и позволяет пользователю создавать собственные функции.

Для обращения к функции необходимо набрать имя функции и список аргументов, заключенные в скобки (EXP). Количество и тип аргументов указывается в описании. Функция используется в выражениях так же, как константы и переменные.

Функция вычисляет результат и возвращает его значение. Интерпретатор продолжает вычислять значение выражения, как если бы вместо функции указали результат ее действия.

В тригонометрических функциях аргумент (EXP) задается в радианах. Пользователю предлагается использовать следующую формулу преобразования:

$$\text{Значение в радианах} = \frac{\text{Значение в градусах} * \text{PI}}{180}$$

Для арктангенса значение угла находится в интервале (—PI/2;PI/2).

Таблица 3

Функция	Значение
1	2
SGN(EXP)	Функция знака. Результат функции: если EXP — положительный аргумент, то (+1); если EXP — отрицательный аргумент, то (—1); если EXP — нулевой аргумент, то 0.
ABS(EXP)	Функция «абсолютная величина». Определяет абсолютное значение аргумента EXP.
INT(EXP)	Целочисленная функция. Определяет целую часть EXP, округляя результат.
SIN(EXP)	Функция синуса. Вычисляет синус аргумента EXP.
COS(EXP)	Функция косинуса. Вычисляет косинус аргумента EXP.
ATN(EXP)	Функция арктангенса. Вычисляет арктангенс аргумента EXP.
SQR(EXP)	Функция квадратного корня. Вычисляет квадратный корень положительного аргумента EXP.
EXP(EXP)	Экспоненциальная функция. Вычисляет показательную функцию (E∧X, где E=2,71828 ..., X<=87).
LOG(EXP)	Функция натурального логарифма. Вычисляет натуральный логарифм аргумента EXP (аргумент должен быть положительным).
LOG10(EXP)	Функция десятичного логарифма. Вычисляет десятичный логарифм аргумента EXP (аргумент должен быть положительным).
PI	Возвращает постоянное значение, равное 3,1415927. Может использоваться как числовая константа.
RND(EXP)	Функция случайных чисел. Генерирует псевдослучайное число или совокупность чисел в интервале (0;1). Значение аргумента EXP игнорируется. Функция генерирует один и тот же список чисел при многократном выполнении.

Пример:

Функция нахождения целой части числа.

Функцию INT можно использовать для округления до любого заданного десятичного разряда.

```
10 INPUT M
20 IF M=—9999 THEN 100
30 INPUT P
40 A=INT(M*10∧P+0.5)/(10∧P)
50 PRINT M,P,A
100 END
```

В строке 10 вводится искомое число. В строке 30 вводится необходимая точность (количество знаков после запятой). В строке 40 вычисляется точность округления (P) дробной части числа.

Пример:

Тригонометрические функции.

Программа преобразует значение угла в градусах в значение угла в радианах (строка 30). Затем вычисляет и печатает (строка 60) синус, косинус и тангенс угла. А также печатает значение арктангенса величины, полученной в строке 50.

```
10 INPUT G
20 IF G=—9999 THEN 100
30 R=G*PI/180
40 IF ABS(COS(R))<.01 THEN 100
50 T=SIN(R)/COS(R)
60 PRINT R,SIN(R),COS(R),T,ATN(T)
100 END
```

Анализ программы на число —9999 (строка 20) позволяет закончить или продолжить работу программы. В строке 40 косинус аргумента проверяется на ноль, так как при вычислении тангенса деление на машинный ноль вызовет сообщение об ошибке.

Пример:

Функция натурального логарифма.

Программа вычисляет и печатает логарифмы (строка 40) по любому основанию.

```
10 INPUT B
20 INPUT X
30 IF X=—9999 THEN 100
40 PRINT B,X,LOG(X),LOG(X)/LOG(B)
50 GOTO 10
100 END
```

5.2.2. Оператор RANDOMIZE

Формат оператора RANDOMIZE: [N] RANDOMIZE

где N — номер строки.

Оператор RANDOMIZE помещается перед первым использованием функции случайных чисел (функция RND) в программе. При выполнении функции RND оператор RANDOMIZE изменяет начальное значение случайного числа таким образом, что та же самая программа, выполняемая второй раз, дает другие результаты. **Пример:**

Использование функции RND.

1) Без оператора RANDOMIZE.

```
10 PRINT RND,RND,RND
```

Результаты, выводимые на терминал, после неоднократного выполнения строки 10:

.0407319	.528293	.0803172
.0407319	.528293	.0803172
.	.	.
.	.	.
.	.	.

2) С оператором RANDOMIZE.

```
5 RANDOMIZE
10 PRINT RND,RND,RND
```

Результаты, выводимые на терминал, после неоднократного выполнения строк 5,10:

.219321	.0640597	.410471
.44509	.741367	.442393
.668662	.412022	.454539
.	.	.
.	.	.
.	.	.

Для генерации случайных чисел, находящихся в открытом интервале (A,B), используется следующее выражение:
(B—A)*RND+A
где A и B — числа задаваемого интервала.

Пример:

```
10 FOR I=1 TO 5
20 PRINT (7—5)*RND+5
30 NEXT I
```

Результаты, выводимые на терминал:

```
5.08146
6.05659
6.60634
5.12878
5.31561
```

Оператор RANDOMIZE не рекомендуется использовать при тестировании и отладке программы.

5.2.3. Строковые функции. Строковые функции — это функции, которые обрабатывают строковый или числовой аргумент.

Если имя функции оканчивается символом ($), то в результате ее вызова создается строка. Если символ ($) отсутствует в имени функции, результатом будет целое десятичное значение.

5.2.3.1. Функция LEN. Функция LEN определяет длину строки, то есть выдает количество символов в символьной строке.

Формат функции LEN: LEN(STR)
где STR — строковая константа.

Пример:

```
10 A$="ABCDEFGS"
20 PRINT LEN(A$)
```

Результат, выводимый на терминал:

```
8
```

5.2.3.2. Функция TRM$. Функция TRM$ возвращает заданную строку, исключая конечные пробелы.

Формат функции TRM$: TRM$(STR)
где STR — символьная строка.

Пример:

```
10 A$="ФУНК "
20 B$="ЦИЯ"
30 PRINT TRM$ (A$)+B$
```

Результат, выводимый на терминал:

```
ФУНКЦИЯ
```

5.2.3.3. Функция POS. Функция осуществляет поиск подстроки в строке.

Формат функции POS: POS(STR1,STR2,EXP)
где
STR1 — строка, в которой осуществляется поиск;
STR2 — подстрока;
EXP — позиция символа (десятичное число), с которого начинается поиск.

Обнаружив подстроку в строке, POS возвращает позицию первого символа подстроки, и возвращает значение, равное 0, если подстроки в строке нет.

Пример:

```
10 W$="пндвтрсрдчтвптнсбтвск"
20 PRINT "введите день недели:пнд,втр,срд,чтв,";
21 PRINT "птн,сбт,вск"
25 PRINT "при D$='END' программа выходит на конец"
30 INPUT D$
40 IF D$="END" THEN 1000
50 IF LEN(D$) <>3 THEN 100
60 D=(POS(W$, D$,1)+2)/3
70 IF D<>INT(D) THEN 100
80 PRINT D$;;D;"-й день недели"
90 GOTO 20
```

```
100 PRINT "введите еще раз"/GOTO 30
1000 END
```

В этой программе функция POS используется для установления соответствия наименования дня недели его порядковому номеру.

Предполагаемые ситуации при использовании функции POS:

1) подстрока (STR2) пустая, а строка (STR1) не пустая, функция возвращает длину строки плюс 1, если в EXP номер позиции больший, чем исходная строка; и возвращает значение выражения EXP, если EXP меньше, чем количество символов в искомой строке;

2) строка (STR1) пустая, а подстрока (STR2) не пустая, функция возвращает 0;

3) значение выражения (EXP) меньше 1, функция начинает поиск с первого символа;

4) значение выражения (EXP) больше длины строки (STR1), а подстрока (STR2) не пустая, функция возвращает 0.

5.2.3.4. Функция SEG$. Функция выделяет подстроку в пределах символьной строки.

Формат функции SEG$:

SEG$(STR,EXP1,EXP2)

где STR — строка символов;
EXP1 — позиция первого копируемого символа;
EXP2 — позиция последнего копируемого символа.

Пример:

```
10 PRINT SEG("ПОДСТРОКА",4,9)
```

Результат, выводимый на терминал:

СТРОКА

Предполагаемые ситуации при работе с функцией SEG$:

Задаваемое значение EXP1 меньше 1, БЕЙСИК считает его равным 1;

EXP1 больше EXP2 или длины строки, функция возвращает нуль-строку (пустая строка);

EXP2 больше длины строки, функция возвращает всю строку;

EXP1 равно EXP2, функция возвращает один символ.

5.2.3.5. Функции даты (DAT$) и времени (CLK$). Функция даты DATS возвращает текущую дату в виде:

NN—MMM—YY

где NN — две цифры месяца;
MMM — три буквы месяца;
YY — две последние цифры года.

Функция даты имеет вид: DAT$

Функция времени CLK$ возвращает текущее время в виде:
TT—MM—SS
где TT — часы;
MM — минуты;
SS — секунды.

Функция времени имеет вид: CLK$

5.2.3.6. Функции преобразования

БЕЙСИК содержит набор строковых функций (приведены в табл. 4) преобразования символов в соответствующий код КОИ—7 и наоборот.

Таблица 4

Функция	Значение
1	2
ASC(STR)	Возвращает значение символа в коде КОИ-7. Строка STR должна содержать один символ и не должна быть пустой. Пример: PRINT ASC(«X») 88
CHR$(EXP)	Возвращает строку из одного символа. Значение EXP может иметь вид (N+128), где N — десятичное значение кода КОИ-7 в пределах от 0 до 127. Однако символы вида CHR$(N) и CHR$ (N+128), не эквивалентны, хотя и являются одним и тем же символом в таблице КОИ-7.
VAL(STR)	Возвращает числовое значение, заданное строкой. Строка должна иметь вид числовой константы. Пример: PRINT VAL(«15.3E-2») 0.153
STR$(EXP)	Возвращает значение выражения в виде строки без промежуточных, начальных и конечных пробелов. Пример: PRINT STR$(« 2 + 3 ») 5
BIN(STR)	Возвращает целое десятичное значение двоичного числа, представленного строкой. STR может быть нулем, единицей и пробелом. Пробел игнорируется при вводе. Пример: 10 PRINT BIN('100 101 001') 297
OCT(STR)	Возвращает целое десятичное значение восьмеричного числа, представленного строкой. STR содержит цифры 0—7 и пробел. Пробел игнорируется при вводе. Пример: 10 PRINT OCT(«451») 297

5.2.4. Функции, определяемые пользователем.

Чтобы избежать неоднократного повторения одной и той же последовательности операторов или одних и тех же математических формул в различных точках программы, БЕЙСИК позволяет вводить функции, составленные на языке БЕЙСИК самим пользователем, и определять их.

Имена функций начинаются буквами FN. Третья буква — любая латинская, за которой следует знак "%" или знак "$". Затем в круглых скобках следует список переменных (количество от 1 до 5) и через знак "=" располагается вычисляемое выражение.

Функции могут определяться в любом месте программы с помощью оператора DEF.

Формат оператора DEF:

$$\text{DEF FNL}\left[\begin{matrix}\$\\ \%\end{matrix}\right](\text{LIST})=\text{EXP}$$

где L — любая латинская буква;
% — функция возвращает целое значение;
$ — функция возвращает строку;
LIST — список переменных, которые могут быть целыми, вещественными или строковыми;
EXP — вычисляется при каждом использовании функции и может использовать переменные, которые не входят в LIST.

Если знаки "%" и "$" отсутствуют, функция возвращает вещественное значение.

Тип выражения должен соответствовать типу функции. Если EXP вещественное выражение, а имя функции целого типа (или наоборот), то БЕЙСИК приводит значение выражения к типу, заданному именем функции.

При обращении к функции интерпретатор вычисляет значение выражения в операторе DEF, заменяя фиктивные переменные функции в операторе DEF соответствующими значениями из списка выражений в обращении к функции.

Пример:

```
10 DEF FNA(X,Y)=X^2+Y^2
20 INPUT A,B
30 PRINT FNA(A,B)
```

ПРИМЕЧАНИЯ:

1) Тип и количество используемых переменных должны соответствовать переменным из LIST оператора DEF.

2) Аргумент функции может отсутствовать.

Пример:

```
10 DEF FNA=X^2
20 R=FNA( )
```

3) Одна и та же функция определяется оператором DEF один раз.

4) Оператор DEF используется только в программном режиме.

6. ВВОД — ВЫВОД ДАННЫХ

БЕЙСИК позволяет вести обработку файлов данных двух типов: с последовательным доступом и с прямым доступом.

Перед обращением к файлу данных его необходимо открыть, то есть связать файл с номером канала. Для этого используется оператор OPEN.

При завершении работы с файлом его необходимо закрыть, то есть отключить от канала. Для этого используется оператор CLOSE.

6.1. Оператор OPEN. Оператор OPEN связывает номер канала ввода — вывода с именем определенного файла.

Формат оператора OPEN:

```
[N] OPEN STR FOR INPUT AS FILE [#]EXP1
```

или `[N] OPEN STR FOR OUTPUT AS FILE [#]EXP1`

где N — номер строки;

STR — спецификация файла;

EXP1 — номер канала ввода — вывода; константа в диапазоне (1—12).

Оператор FOR INPUT открывает существующий файл для считывания информации в память.

Оператор FOR OUTPUT создает новый файл. Если на устройстве существует файл с тем же именем, он уничтожается.

К концу оператора могут добавляться следующие параметры: [DOUBLE BUF] [,FILESIZE EXP1]

где

DOUBLE BUF — дополнительный оператор; определяет наличие места в памяти для организации второго буфера;

FILESIZE EXP1 — определяет область в блоках для выходного файла на диске.

6.2. Оператор CLOSE. Оператор CLOSE закрывает определенный логический файл.

Формат оператора CLOSE:

```
CLOSE [ [#]EXP1,[#]EXP2,[#]EXP3,... ]
```

где

EXP1,EXP2,... — логические номера, связанные с открываемыми файлами.

Оператор CLOSE без определенной спецификации закрывает все открытые файлы. После закрытия выходные файлы становятся постоянными.

Операторы CHAIN, END или выполнение строки с наибольшим номером закрывает файл.

Пример:

```
10 B = 10
20 CLOSE #3, B—2, B+1, B
```

При выполнении строки 20 закрываются файлы, связанные с логическими номерами 3, 8, 11, 10.

6.3. Файл с последовательным доступом открыт либо для ввода, либо для вывода операторами OPEN FOR INPUT и OPEN FOR OUTPUT соответственно, но не одновременно. Если файл открыт для ввода, он считается оператором INPUT # и LINPUT #. Если он открыт для вывода, то будет считываться оператором PRINT #.

Оператор OPEN без FOR INPUT или FOR OUTPUT открывает файл для считывания из него, если файл существует, и для формирования нового, если он отсутствует.

6.3.1. Операторы INPUT # и LINPUT #. Оператор INPUT # считывает файл, открытый оператором OPEN FOR INPUT.

Формат оператора INPUT #:

N INPUT #EXP,VAR1[,VAR2,VAR3,...]

где N — номер строки;
EXP — номер канала, связанного с файлом;
VAR1,VAR2,... — элементы данных, которым присваиваются вводимые значения.

Оператор LINPUT # считывает символьную строку из файла.

Формат оператора LINPUT #:

N LINPUT #EXP,STR—VAR1[,STR—VAR2,STR—VAR3,...]

где N — номер строки;
EXP — номер канала, связанного с файлом;
STR—VAR1,STR—VAR2,... — строковые переменные.

При считывании файла данных необходимо придерживаться тех же синтаксических правил, как если бы ввод выполнялся с терминала. Числовые данные должны быть разделены запятыми или управляющим символом <ВК>.

6.3.2. Оператор PRINT #. Оператор PRINT # помещает данные в определенный файл.

Формат оператора PRINT #: N PRINT #EXP[,LIST]

где N — номер строки;

EXP — номер канала, связанного с файлом;
LIST — список элементов, представленных в виде числовых и строковых выражений, или функцией TAB, и разделяемых запятой или точкой с запятой.

Оператор PRINT # без списка аргументов не помещает строку пробелов в файл.

ПРИМЕЧАНИЕ. В операторах INPUT #, LINPUT #, PRINT # значение EXP не должно быть равным нулю; после EXP в качестве разделителя можно ставить символ ":".

6.3.3. Оператор PRINT USING. Оператор PRINT USING печатает результаты работы программы в установленном формате.

Формат оператора PRINT USING:

PRINT [#EXP,] USING STR,LIST

где

EXP — номер канала ввода — вывода, связанного с файлом;
STR — строковая константа (формат); управляет печатью списка LIST;
LIST — список выводимых элементов.

В форматной строке STR могут быть обычные символы и символы управления форматом печати. Обычные символы печатаются в идентичной форме. Символы управления форматом печати определены двумя типами полей: числовыми и строковыми.

Пример:

```
40 PRINT USING "формат: ###.#", 15.1
```

Сообщение, выводимое на терминал:

```
Формат: 15.1
```

6.3.3.1. Числовые поля

Поле чисел указывается в форматной строке символом # и выравнивается по правому краю. Расположение десятичной точки определяется символом '.'. Производится округление (не усечение) чисел, если это необходимо.

Пример:

```
30 PRINT USING "###.##", 12.345
40 PRINT USING "##.###", 10.55
50 PRINT USING "######", 142
60 PRINT USING ".###",0.444
70 PRINT USING "задание:##",2
```

Сообщение, выводимое на терминал:

```
12.35
10.550
  142
.444
```

Задание: 2

Если числовое поле в форматной строке меньше выводимого по этому формату элемента, БЕЙСИК печатает знак % и выводит значение элемента списка, не принимая во внимание поле формата. Если в поле числа указано больше знакомест, чем имеется в числе, после последней значащей цифры проставляются нули.

Пример:

```
40 PRINT USING "##", 100
```

Сообщение, выводимое на терминал: % 100

Для более гибкого управления печатью чисел используются специальные символы:

1) символ "**".

Если числовое поле в форматной строке начинается со звездочек **, любые неиспользованные знакоместа в поле формата заполняются звездочками. Отрицательные числа не могут выводиться с помощью заполнения звездочками, если знак не выводится после числа.

Пример:

```
10 A = 27.95 \ B = 107.50 \ C = 1007.50
20 PRINT USING "**##.##",A, B, C
```

Сообщение, выводимое на терминал:

```
**27.95
*107.50
1007.50
```

2) Символ возведения в степень "^".

Если число представляется в экспоненциальном формате, то за полем числа в форматной строке следует последовательность символов ^^^^ (четыре).

Пример:

```
10 PRINT USING "##.#^^^^",1
20 PRINT USING "##.#^^^^",300
```

Сообщение, выводимое на терминал:

```
10.0E-01
30.0E+01
```

Символы "^^^^" нельзя использовать в поле числа с начальными символами ** и $$ и с конечным знаком минус.

3) Конечный знак минус.

Если поле числа в форматной строке завершается знаком минус, знак выводимого числа печатается позади числа.

Пример:

```
10 PRINT USING "###.#-",5
20 PRINT USING "###.#-", -10.5
```

Сообщение, выводимое на терминал:

```
 5.0
10.5—
```

4) Символ "$".

Если поле числа начинается с символов $$, перед первой цифрой числа выводится символ $. Символы $ $ резервируют два знакоместа: первое — для символа $. Действие второго символа $ эквивалентно указанию дополнительного символа # в форматной строке. Отрицательные числа не могут выводиться с помощью плавающего символа $, если знак не выводится после числа.

Пример:

```
10 PRINT USING "$$##.#—",—5.35
20 PRINT USING "$$##.#", 305
30 PRINT USING "$$##.#", 10001.1
```

Сообщение, выводимое на терминал:

```
 $5.4—
$305.0
% 1001.1
```

5) Символ ",".

Если в поле числа, слева от десятичной точки (в любой позиции), поместить символ ",", то запятая включается через каждые три цифры слева от десятичной точки. Символ "," справа от десятичной точки считается печатным символом.

Пример:

```
10 PRINT USING "#,###.#",5625.34
20 PRINT USING "###.#,#",46.375
30 PRINT USING "#####,#",17580
```

Сообщение, выводимое на терминал:

```
5,625.3
 46.4,
17,580
```

6.3.3.2. Строковые поля. Строковые поля при печати могут быть выравнены слева или справа, или расположены по центру строкового поля форматной строки. Если символов в строке больше, чем резервируется позиций, строка усекается.

Символы управления печатью для строковых полей следующие:

1) Апостроф «'».

Резервирует место для одного символа; является признаком начала строкового поля, за которым следует одна из букв: L, R, C и E (употребление этих букв описано ниже).

Пример:

```
10 PRINT USING «'», «НЕТ»
```

Сообщение, выводимое на терминал:

H

2) Символ «L».

Выравнивает строку влево, резервирует место для одного символа.

Пример:

```
10 PRINT USING «'LL»,«A»
20 PRINT USING «'LLLLL»,«ABCDEKLMN»
30 PRINT USING «'LLL»,«ДА»
```

Сообщение, выводимое на терминал:

```
A
ABCDEK
ДА
```

3) Символ «R».

Выравнивает строку вправо, резервирует место для одного символа.

Пример:

```
10 PRINT USING «'RRRRR»,«A»
20 PRINT USING «'RRRRR»,«ABCDEKLMN»
30 PRINT USING «'RRRRR»,«ДА»
```

Сообщение, выводимое на терминал:

```
    A
ABCDEK
   ДА
```

4) Символ «C».

Печатает строку по центру строкового поля форматной строки, резервирует место для одного символа.

Пример:

```
10 PRINT USING «'CCCCC»,«A»
20 PRINT USING «'CCCCC»,«AB»
30 PRINT USING «'CCCCC»,«ABC»
40 PRINT USING «'CCCCC»,«ABCD»
50 PRINT USING «'CCCCC»,«ABCDE»
```

Сообщение, выводимое на терминал:

```
  A
  AB
 ABC
 ABCD
ABCDE
```

5) Символ «E».

Выравнивает строку влево, расширяет поле (если необходимо напечатать всю строку), резервирует место для 1 символа. Если в строковом поле зарезервировано меньше симво-

лов, чем в символьной строке, БЕЙСИК расширяет поле и печатает всю строку.

Пример:

Использование символов управления печатью для строковых полей.

```
40 READ A$
50 F$=«::'CCCC::'EEEE::'LLLL::'RRRR::»
60 IF A$="" GOTO 100
70 PRINT USING F$, A$, A$, A$, A$
80 GOTO 40
90 DATA«ABCD»,«ABCDEFG»,«A»
95 DATA«AB»,«»
100 END
```

Сообщение, выводимое на терминал:

```
::ABCD ::ABCD ::ABCD :: ABCD::
::ABCDE::ABCDEFG::ABCDE::ABCDE::
::     A ::A         ::A      ::      A::
::    AB ::AB        ::AB     ::     AB::
```

6.3.3.3. Общий вид форматной строки

В форматной строке оператора PRINT USING может быть несколько полей (числовых и строковых) одновременно.

Пример:

```
30 A% = 13\B%=12\A$=«MATEM.»\B$ = «CTPOK.»
40 F$ = «ФУНКЦИЙ:### 'RRRRR AND ## 'RRRR»
50 PRINT USING F$,A%,A$,B%,B$
```

Сообщение, выводимое на терминал:

```
ФУНКЦИЙ: 13 MATEM. И 12 CTPOK.
```

Пример:

```
40 OPEN «LP:» FOR OUTPUT AS FILE #1
50 A$=«ABC»\ B$ = «12»
60 PRINT #1, USING «ИМЕЕТСЯ:'RRRR>'LL»,A$,B$
```

В файл по каналу 1 печатается:

```
ИМЕЕТСЯ: ABC>12
```

6.3.3.4. Ошибки при работе с оператором PRINT USING

При работе с оператором PRINT USING могут возникнуть два вида ошибок: неустранимые и устранимые.

Фатальная ошибка прекращает работу оператора и БЕЙСИК выдает сообщение: ?PRINT USING ERROR(?PRU).

Условия возникновения неустранимой ошибки:

1) недопустимый формат;

2) неверное построение полей в форматной строке;

3) несоответствие типа поля элементу списка;

4) печать отрицательного числа в поле с начальными символами ** и $$ в случае, когда конечный знак минус не задан;

5) элементы списка разделены символами, отличными от запятой или точки с запятой.

При возникновении устранимой ошибки, работа оператора PRINT USING продолжается, но результат выполнения может оказаться неверным.

Условия возникновения устранимой ошибки:

1) число знакомест элемента списка больше числа знакомест, указанного в поле формата;

2) поле формата содержит недопустимую комбинацию символов;

3) обычные символы представляют собой одно из допустимых полей форматной строки.

6.3.4. Оператор IF END #. Оператор IF END # передает управление на строку с соответствующим номером либо оператору при обнаружении признака конца файла с последовательным доступом.

Формат оператора IF END # THEN:

N IF END [#]EXP THEN LINE NUMBER

или N IF END [#]EXP THEN STATEMENT

где N — номер строки;

EXP — номер канала, связанного с файлом.

LINE NUMBER — номер строки, выполняемой в случае истинности условия;

STATEMENT — оператор, подлежащий выполнению.

Формат оператора IF END # GOTO:

N IF END [#]EXP GOTO LINE NUMBER

где N — номер строки;

EXP — номер канала, связанного с файлом;

LINE NUMBER — номер строки, выполняемой в случае истинности условия.

Признак конца файла с последовательным доступом обнаруживается, если в файле больше нет данных.

6.3.5. Оператор RESTORE #. Оператор RESTORE # устанавливает указатель считывания файла в начальную позицию.

Формат оператора RESTORE #: N RESTORE #EXP,

где N — номер строки;

EXP — номер канала, связанного с файлом.

6.3.6. Использование файлов данных с последовательным доступом

Пример:

```
10 OPEN "FIL1" FOR OUTPUT AS FILE #1
20 READ A$, B, C%
30 IF A$ = "" THEN GOTO 60
```

```
40 PRINT #1, A$;«,»;B;«,»;C%
50 GOTO 20
60 CLOSE #1
110 IF END #4 GOTO 160
120 INPUT #4,A$,B,C%
130 PRINT A$,B,C%
140 GOTO 110
160 PRINT «конец файла»
170 DATA «К1»,5.2,5,«К2»,6.2,6
180 DATA «К3»,7.2,7,«К4»,8.2,8
190 DATA "",0,0
200 CLOSE #4
210 END
```

6.4. Файлы с прямым доступом. Оператор DIM #

Оператор OPEN без FOR INPUT и FOR OUTPUT аналогичен оператору OPEN FOR INPUT.

Оператор DIM #, используемый с оператором OPEN, определяет файл данных как файл с прямым доступом.

Формат оператора DIM #:

N DIM #EXP1,VAR(EXP2[,EXP3])[=EXP4]

где N — номер строки;
EXP1 — номер канала, связанного с файлом;
VAR — переменная, определяющая массив;
EXP2,EXP3 — индексы массива VAR;
EXP4 — максимальная длина символьных строк строкового массива.

Пример:

```
10 DIM #2%,AB$(10) = 128
20 OPEN «NAME» FOR INPUT AS FILE #2
```

Пример:

```
10 DIM #1%,A(10,10)
20 OPEN «FILE» AS FILE #1
```

6.5. Оператор NAME TO. Оператор NAME TO изменяет имя файла.

Формат оператора: NAME STR1 TO STR2

где STR1 — спецификация файла, подлежащего изменению;
STR2 — новая спецификация файла.

Если в STR1 указано устройство, то в STR2 должно быть указано это же устройство.

6.6. Оператор KILL. Оператор KILL удаляет определенный файл.

Формат оператора: [N] KILL STR

где N — номер строки;
STR — спецификация удаляемого файла.

Пример:

Использование файла с прямым доступом.

```
10 DIM #3%,V(20)
20 OPEN «RANDOM» AS FILE #3%
30 FOR I% = 0% TO 10%
40 PRINT V%(I%)
50 NEXT I%
60 CLOSE #3%
70 END
```

6.7. Файлы программ БЕЙСИК

6.7.1. Оператор CHAIN. Оператор CHAIN сегментирует программу, сохраненную в файле на диске.

Формат оператора CHAIN: [N] CHAIN STR [LINE EXP]
где N — номер строки;
STR — спецификация вызываемого файла (сегмента программы);
EXP — номер строки, с которой начинается выполнение сегмента программы.

При выполнении оператора CHAIN все открытые файлы закрываются, загружается новый сегмент программы и выполнение продолжается.

Переменные и массивы теряют свои значения, если они не указаны в списке оператора COMMON (описание оператора приведено ниже).

Если EXP равен нулю, выполнение начинается с оператора с наименьшим номером.

При работе оператора могут возникнуть следующие ошибки:

?FILE NOT FOUND (?FNF)
(файл не обнаружен);
?SYNTAX ERROR (?SYN)
(недопустимый номер строки, указанный
в операторе CHAIN);
?UNDEFINED LINE NUMBER? (?ULN)
(несуществующий номер строки, указанный
в операторе CHAIN);

6.7.2. Оператор COMMON. Оператор COMMON передает данные в памяти между сегментами программ.

Формат оператора COMMON: N COMMON LIST
где N — номер строки;
LIST — список переменных и массивов.

Необходимо соблюдать следующие правила:

1) последовательность списка в операторе COMMON должна быть одинакова у всех вызываемых сегментов программ;

2) массивы, указанные в операторе COMMON, не должны определяться в операторе DIM;

3) число элементов списка не должно превышать 255.

При работе оператора COMMON могут возникнуть следующие ошибки:

?TOO MANY ITEMS IN COMMON (?TIC)
(число элементов списка превышает 255);
?ILLEGAL DIMENSION (?IDM)
(массив, указанный в списке COMMON,
также определен в операторе DIM);

Пример:

Использование операторов CHAIN и COMMON.

Сегмент 1.

```
10 COMMON A(20)
20 FOR I = 1 TO 10
30 A(I) = I
40 NEXT I
50 CHAIN «SEG2»
```

Сегмент 2.

```
10 COMMON A(20)
20 FOR I = 1 TO 10
30 PRINT A(I)
40 NEXT I
50 END
```

6.7.3. Оператор OVERLAY. Оператор OVERLAY объединяет программу в памяти с программой, хранимой в определенном файле на устройстве файловой структуры.

Формат оператора OVERLAY: N OVERLAY STR [LINE EXP]

где N — номер строки;
STR — спецификация файла (сегмента программы);
EXP — номер строки, с которой начинается выполнение программы.

Характеристика оператора OVERLAY:

1) строки вызываемой программы заменяют строки программы в памяти, если их номера совпадают;

2) все массивы и переменные сохраняют свои значения;

3) все открытые файлы остаются открытыми;

4) программа продолжается с указанной строки (если используется оператор LINE) или со строки, следующей за той, которая вызвала оператор OVERLAY.

5) вызываемая программа не должна содержать операторы DIM, DEF.

Пример:

Программа находится в памяти (головная программа).

```
10 DIM A(20)
12 T=5
15 OPEN «LP:» FOR OUTPUT AS FILE #1
20 FOR I = 1 TO 5
25 A(I) =I
30 T = T+A(I)
35 PRINT #1,T
40 NEXT I
50 CLOSE
90 OVERLAY «OVL» LINE 10
100 END
```

Сегмент «OVL».

```
18 PRINT #1\PRINT #1
20 FOR I=1 TO 10
25 A(I) =I*I
```

6.7.4. Оператор CALL. Оператор CALL вызывает программу, написанную на языке АССЕМБЛЕР.

Формат оператора CALL: [N] CALL NAME [(LIST)]

где N — номер строки;

NAME — имя вызываемой программы (строковая константа);

LIST — список аргументов для вызываемой программы.

БЕЙСИК возвращает результат выполнения оператора через аргументы, за исключением элементов файлов с прямым доступом.

Пример:

```
50 CALL «FIL1»(A%, A$, C(9))
```

7. СРЕДСТВА ОТЛАДКИ. КОМАНДА ЗАПУСКА ПРОГРАММЫ

Интерпретатор БЕЙСИК работает с командами трех типов:

— команды редактирования;

— команды работы с файлами программ;

— команды клавиатуры.

7.1. Команды редактирования

7.1.1. Команды LIST и LISTNH

Команда LIST распечатывает на терминал программу, находящуюся в памяти.

Формат команды LIST: LIST [LIN.NUM] [—LIN.NUM]

где LIN.NUM—LIN.NUM — номера строк.

Способы использования команды:

LIST	— распечатка всей программы;
LIST LIN.NUM	— распечатка указанной строки;
LIST —LIN.NUM	— распечатка программы от начала до указанной строки;
LIST LIN.NUM—LIN.NUM	— распечатка программы между указанными строками;
LIST LIN.NUM—	— распечатка программы от указанной строки до конца программы.

Команда LISTNH аналогична команде LIST, но при распечатке программы не выводится заголовок.

7.1.2. Команды RUN и RUNNH. Запуск программ в памяти

Команда RUN запускает программу для выполнения.

Формат команды RUN: RUN [STR]

где STR — спецификация файла

Если задана спецификация файла, то сначала производится чтение файла с внешнего носителя, а затем ее запуск.

Если спецификация файла не задана, то загрузка с внешнего носителя не производится. В этом случае предполагается, что программа находится в оперативной памяти. По этой команде система выводит заголовок программы, который состоит из имени программы, текущей даты и времени системы.

Команда RUNNH аналогична команде RUN. Она отличается тем, что не выводит заголовка программы.

7.1.3. Команда DEL. Команда DEL исключает одну или более строк текущей программы.

Формат команды DEL: DEL [LIN.NUM][—LIN.NUM]

где LIN.NUM—LIN.NUM — номера строк.

Способы использования команды:

DEL	— исключает всю программу;
DEL LIN.NUM	— исключает строку с указанным номером;
DEL —LIN.NUM	— исключает часть программы от начала программы до указанной строки;
DEL LIN.NUM—LIN.NUM	— исключает часть программы между указанными строками;
DEL LIN.NUM—	— исключает часть программы от указанной строки до конца программы.

Для исключения одной строки следует набрать номер этой строки.

7.1.4. Команда NEW. Команда NEW очищает содержимое памяти и присваивает имя программе, которая будет загружаться.

Формат команды NEW: NEW [NAME]
где NAME — имя новой программы.

Если набрана команда NEW, БЕЙСИК печатает:
NEW FILE NAME— —

Пользователь должен указать имя программы, загружаемой в память, или нажать клавишу <ВК>. В этом случае имя загружаемой программы будет NONAME.

7.1.5. Команда SCR. Команда SCR очищает содержимое памяти и присваивает имя NONAME программе, которая будет загружаться.

Формат команды SCR: SCR

7.1.6. Команда CLEAR. Команда очищает содержимое массивов и строковых буферов пользователя. Имя программы не изменяется.

Формат команды CLEAR: CLEAR

7.2. Команды для работы с файлами программ

7.2.1. Команда SAVE. Команда SAVE сохраняет программу, находящуюся в памяти, в коде КОИ—7.

Формат команды SAVE: SAVE [STR]
где STR — спецификация файла

Если спецификация файла отсутствует, команда сохраняет программу на диске с текущим именем.

Пример:
Получение листинга текущей программы.
SAVE LP:

Пример:
Вывод программы, находящейся в памяти, на перфоленту.
SAVE PC:

7.2.2. Команда REPLACE. Команда REPLACE аналогична команде SAVE. Но команда REPLACE заменяет файл, ранее созданный командой SAVE на устройстве файловой структуры. Если файла, который заменяется, нет на устройстве файловой структуры, возникает ошибка.

Формат команды REPLACE: REPLACE [STR]
где STR — спецификация файла.

7.2.3. Команда OLD. Команда OLD загружает программу, сохраненную командой SAVE, в память.

Формат команды OLD: OLD [STR]
где STR — спецификация файла.

Если спецификация файла отсутствует, БЕЙСИК выдает сообщение: OLD FILE NAME— —
Пользователь должен указать имя файла, вызываемого в память, или нажать клавишу <ВК>. В этом случае БЕЙСИК загружает программу с именем NONAME.

7.2.4. Команда APPEND. Команда APPEND загружает программу, сохраненную командой SAVE, и объединяет ее с текущей программой.

Формат команды APPEND: APPEND [STR]
где STR — спецификация файла.

Если обе программы содержат идентичный номер строки, строка в памяти заменяется строкой добавленной программы.

При отсутствии спецификации файла, БЕЙСИК выдает сообщение: OLD LINE NAME— —

7.2.5. Команда RUN. Запуск программы из файла на устройстве файловой структуры

Команда RUN загружает в память программу из указанного файла и выполняет ее.

Формат команды RUN: RUN STR
где STR — спецификация файла.

7.2.6. Команда UNSAVE. Команда UNSAVE удаляет файл с внешнего запоминающего устройства.

Формат команды UNSAVE: UNSAVE STR
где STR — спецификация файла.

7.2.7. Команда RENAME. Команда RENAME изменяет имя программы, находящейся в памяти.

Формат команды RENAME: RENAME [FILNAME]
где FILNAME — имя файла.

При отсутствии спецификации файла БЕЙСИК выдает сообщение: NEW FILE NAME— —

Пользователь должен указать имя файла или нажать клавишу <ВК>. В этом случае новое имя программы будет NONAME.

7.2.8. Команда COMPILE. Команда COMPILE компилирует программу, находящуюся на устройстве файловой структуры, и сохраняет ее на этом устройстве.

Формат команды COMPILE: COMPILE STR
где STR — спецификация исходного файла.

Если спецификация файла отсутствует, команда COMPILE компилирует программу, находящуюся в памяти, и сохраняет ее на системном устройстве файловой структуры.

7.2.9. Команда SUB. Команда SUB редактирует строку текущей программы.

Формат команды SUB: SUB LIN.NUMXSTR1XSTR2[X[N]]
где LIN.NUM — номер строки для редактирования;
X — символ-ограничитель;
STR1 — последовательность заменяемых символов;
STR2 — последовательность вновь вводимых в строку символов;
N — число; N-ое появление STR1 в редактируемой строке.

Ограничитель может быть любым символом (за исключением пробела, табуляции и цифр), допустимым в языке БЕЙСИК.

Ограничитель не должен встречаться в STR1 и STR2.

Пример:

```
10 A = B\C$ = 0          — редактируемая строка;
SUB 10@0@0@2             <BK>
10 A = B\C# = O          — отредактированная строка.
```

Пример:

```
50 PRINT «неверный символ»; — редактируемая строка;
SUB 50@E@и @3
50 PRINT «неверный символ»; — строка отредактирована.
```

7.2.10. Команда RESEQ. Команда RESEQ перенумеровывает строки программы, находящиеся в памяти.

Формат команды RESEQ:
RESEQ [LIN.NUM1],[LIN.NUM2][—LIN.NUM3][,EXP]
где LIN.NUM1 — новый номер строки, взамен старого;
LIN.NUM2—LIN.NUM3 — номера строк, с которых начинается перенумерация;
EXP — шаг приращения номеров строк.

Пример:

```
110 FOR I = 1 TO 5
120 INPUT C
130 A = A + C
140 IF C=0 THEN 160
145 PRINT A
150 NEXT I
160 END
RESEQ 10, 110—160, 10
10 FOR I = 1 TO 5
20 INPUT C
30 A = A+C
40 IF C = 0 THEN 70
50 PRINT A
60 NEXT I
70 END
```

Способы использования команды RESEQ:
— если LIN.NUM1 отсутствует, он принимается равным ближайшему номеру перед LIN.NUM2 плюс приращение.

Пример:

```
10 INPUT A
110 PRINT A
120 END
RESEQ 110—120, 10
10 INPUT A
20 PRINT A
30 END
```

— если LIN.NUM2 отсутствует, нумерация строк производится с первой по LIN.NUM3;
— если LIN.NUM3 отсутствует, нумерация строк производится с LIN.NUM2 до конца программы;
— если EXP отсутствует, приращение считается равным 10.

7.2.11. Команда LENGTH. Команда LENGTH позволяет пользователю определить длину программы, находящейся в памяти.

Формат команды LENGTH: LENGTH

В ответ на набранную команду БЕЙСИК печатает:
XXXX WORDS USED,YYYY FREE
где XXXX — количество слов, занимаемых программой;
YYYY — количество свободных слов в буфере пользователя.

7.3. Команды с клавиатуры (ключевые команды). Ключевые команды вызываются при одновременном нажатии клавиши <СУ> и буквы, которая определяет действие, подлежащее выполнению.

7.3.1. Команда СУ/С. Команда СУ/С вызывает прерывание выполнения команды или программы, БЕЙСИК печатает сообщение: STOP
Команда СУ/С отменяет действие команд СУ/О и СУ/S.

7.3.2. Команда СУ/О. Команда СУ/О прекращает вывод данных на терминал, но не прерывает выполнение программы. Вывод данных на терминал возобновится, если команда СУ/О будет набрана повторно.

7.3.3. Команда СУ/S. Команда СУ/S прекращает вывод данных на терминал и приостанавливает выполнение программы до тех пор, пока не будет набрана команда СУ/Q.

7.3.4. Команда СУ/Q. Команда СУ/Q позволяет продолжать вывод данных на терминал после того, как он был прекращен по команде СУ/S.

7.3.5. Команда СУ/U. Команда СУ/U удаляет набираемую

строку на терминале. Строка удаляется в том случае, если не набран символ «ВК».

7.3.6. Команда RUBOUT. Команда RUBOUT уничтожает последний печатаемый символ на терминале.

Пример:

Если предположить, что напечатано 15 FOR N = 7, а требуется напечатать: 15 FOR N = 1 TO 100, то после нажатия клавиши RUBOUT строка будет иметь вид:

15 FOR N = 7\1 TO 100

ПРИЛОЖЕНИЕ 1

ТАБЛИЦА СИМВОЛЬНЫХ КОДОВ КОИ-7

Таблица

Б	О	О	О	О	О	О	О	О	1	1	1	1	1	1	1
Б	О	О	О	О	1	1	1	1	О	О	О	О	1	1	1
Б	О	О	1	1	О	О	1	1	О	О	1	1	О	О	1
Б	О	1	О	1	О	1	О	1	О	1	О	1	О	1	О

Б4 Б3 Б2 Б1	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
О О О О	ПУС				@	P	'	P	ПУС		SP	О	@	P	Ю	П
О О О 1		ПФ1			a	я	A	Я		ПФ1	!	1	a	я	A	Я
О О 1 О		ПФ2			б	р	Б	Р		ПФ2	"	2	б	р	Б	Р
О О 1 1		ПФ3			ц	с	Ц	С		ПФ3	#	3	ц	с	Ц	С
О 1 О О		ПФ4			д	т	Д	Т		ПФ4	$	4	д	т	Д	Т
О 1 О 1					E	U	Е	У			%	5	E	U	Е	У
О 1 1 О					F	V	Ф	Ж			&	6	F	V	Ф	Ж
О 1 1 1	эъ				G	W	Г	В	ЭВ				G	W	Г	В
1 О О О	ВШ	АН			H	X	Х	Ь	ВШ	АН	(	8	H	X	Х	Ь
1 О О 1	ГТ				I	Y	И	Ы	ГТ		)	9	I	Y	И	Ы
1 О 1 О	ПС				J	Z	Й	З	ПС		*	:	J	Z	Й	З
1 О 1 1	ВТ	АР2			K	[	К	Ш	ВТ	АР2	+	;	K	[	К	Ш
1 1 О О	ПФ				L	\	Л	Э	ПФ		'	<	L	\	Л	Э
1 1 О 1	ВК				M	]	М	Щ	ВК		-	=	M	]	М	Щ
1 1 1 О	ВЫХ				N	^	Н	Ч	ВЫХ		.	>	N	^	Н	Ч
1 1 1 1	ВХ				O	_	О	ЗБ	ВХ		/	?	O	_	О	

ОПЕРАТОРЫ ЯЗЫКА БЕЙСИК

CALL	— вызывает программу, написанную на языке АССЕМБЛЕРа.
CHAIN	— сегментирует программу, сохраненную в файле на устройстве файловой структуры.
CLOSE	— закрывает определенный логический файл.
COMMON	— передает данные в памяти между сегментами программ.
DATA	— организует блок данных для считывания оператором READ.
DEF	— определяет функцию пользователя FNL.
DIM	— резервирует место в памяти для числовых или строковых данных.
DIM #	— определяет файл данных как файл с прямым доступом.
END	— определяет физический конец программы.
FOR	— указывает начало цикла и определяет его параметры.
GOSUB	— осуществляет переход к первому оператору подпрограммы.
GO TO	— осуществляет переход к указанной строке.
IF THEN, IF GO TO	— передает управление в зависимости от истинности или ложности выражения отношения.
IF END # THEN, IF END # GOTO	— передает управление при обнаружении признака конца файла с последовательным доступом.
INPUT	— вводит данные с терминала в процессе выполнения программы.
INPUT #	— считывает файл, открытый оператором OPEN FOR INPUT.
KILL	— удаляет определенный логический файл.
LINPUT	— вводит строковые данные с терминала.
LINPUT #	— считывает строковые данные из файла, открытого оператором OPEN FOR INPUT.

NAME TO	— изменяет имя файла.
NEXT	— указывает конец цикла, организованного оператором FOR.
ON GOSUB	— передает управление одной из нескольких подпрограмм.
ON GOTO, ON THEN	— передает управление одной из указанных строк в зависимости от значения числового выражения.
OPEN	— связывает номер канала ввода — вывода с именем определенного файла.
OVERLAY	— объединяет программу в памяти с программой, хранимой в определенном файле на устройстве файловой структуры.
PRINT	— выводит данные на терминал.
PRINT #	— помещает данные в определенный логический файл.
PRINT[#]USING	— печатает результаты работы программы в установленном формате.
RANDOMIZE	— изменяет начальное значение случайного числа при выполнении функции RND.
READ	— считывает блок данных, организованных оператором DATA.
REM	— используется для ввода примечаний и комментариев в программу пользователя.
RESET	— эквивалентен оператору RESTORE.
RESTORE	— устанавливает указатель считывания данных, организованных оператором DATA, в начальную позицию.
RESTORE #	— устанавливает указатель считывания файла в начальную позицию.
RETURN	— возвращает управление оператору, следующему за последним выполненным оператором GOSUB.
STOP	— производит останов программы. Работа программы может быть продолжена оператором GO TO.

ФУНКЦИИ ЯЗЫКА БЕЙСИК

Математические функции

ABS(EXP) — функция «Абсолютная величина». Определяет абсолютное значение аргумента EXP.

ATN(EXP) — функция арктангенса. Вычисляет арктангенс аргумента EXP.

COS(EXP) — функция косинуса. Вычисляет косинус аргумента EXP.

EXP(EXP) — экспоненциальная функция. Вычисляет показательную функцию (E∧X, где E = 2.71828..., X <= 87).

INT(EXP) — целочисленная функция. Определяет целую часть EXP, округляя результат.

LOG(EXP) — функция натурального логарифма. Вычисляет натуральный логарифм аргумента (EXP>0).

LOG10(EXP) — функция десятичного логарифма. Вычисляет десятичный логарифм аргумента (EXP>0).

PI — присваивает постоянное значение, равное 3.1415927. Может использоваться как числовая константа.

RND(EXP) — функция случайных чисел. Генерирует последовательность случайных чисел в интервале (0,1).

SGN(EXP) — функция знака. Результат выполнения функции: +1 — если EXP>0
0 — если EXP=0

SIN(EXP) — функция синуса. Вычисляет синус аргумента EXP.

SQR(EXP) — функция квадратного корня. Вычисляет квадратный корень аргумента EXP (EXP>=0).

TAB — функция табуляции. Организует выходной формат печати.

Строковые функции

ASC(STR) — возвращает значение символа в коде КОИ—7.

BIN(STR)	— возвращает целое десятичное значение двоичного числа, представленного строкой.
CHR$(EXP)	— возвращает строку из одного символа.
CLK$	— возвращает текущее время.
DAT$	— возвращает текущую дату.
LEN(STR)	— возвращает количество символов в строке.
OCT(STR)	— возвращает целое десятичное значение восьмеричного числа, представленного строкой.
POS(STR1,STR2, EXP)	— осуществляет поиск подстроки в строке и возвращает позицию первого элемента подстроки.
SEG$(STR,EXP1, EXP2)	— выделяет подстроку в строке.
STR$(EXP)	— возвращает выражение в виде строки без промежуточных, начальных и конечных пробелов.
TRM$(STR)	— возвращает заданную строку без конечных пробелов.
VAL(STR)	— возвращает десятичное значение числа, заданного в виде символьной строки.

ПРИЛОЖЕНИЕ 4

КОМАНДЫ ИНТЕРПРЕТАТОРА БЕЙСИК

APPEND	— соединяет программу, сохраненную в файле, с текущей программой в памяти.
CLEAR	— очищает числовые и строковые переменные, массивы.
COMPILE	— компилирует программу и сохраняет ее на устройстве файловой структуры.
DEL	— исключает строки текущей программы.
LENGTH	— определяет длину программы и объем свободной памяти в буфере пользователя.
LIST	— печатает на терминал строки программы, заданные их номерами.
NEW	— очищает буфер пользователя и присваивает имя программе, которая будет загружаться.
OLD	— загружает программу из указанного файла, предварительно очищая буфер пользователя.

RENAME — изменяет имя текущей программы на заданное.

REPLACE — заменяет файл на устройстве файловой структуры текущей программой.

RESEQ — перенумеровывает строки программы.

RUN — загружает и запускает программу из указанного файла. Запускает текущую программу, если спецификация файла не указана.

SAVE — сохраняет программу, находящуюся в памяти, на одном из внешних носителей.

SCR — очищает буфер пользователя.

SUB — редактирует строку текущей программы.

UNSAVE — удаляет заданный файл с устройства файловой структуры.

БЕЙСИК

РУКОВОДСТВО ПРОГРАММИСТА

1. НАЗНАЧЕНИЕ ПРОГРАММЫ

Интерпретатор БЕЙСИК предназначен для проверки, анализа и выполнения операторов языка БЕЙСИК.

Интерпретатор работает под управлением операционной системы ФОДОС-2; объем оперативной памяти, занимаемой интерпретатором, не более 14К слов.

2. ХАРАКТЕРИСТИКИ ПРОГРАММЫ

Функции интерпретатора БЕЙСИК можно сократить или расширить введением новых модулей при генерации системы.

Функции, реализуемые включаемыми модулями, являются необязательными для интерпретатора БЕЙСИК и отсутствуют в минимальной конфигурации.

Средства расширения функций интерпретатора:
операторы: CALL, PRINT USING;
команды: SUB, RESEQ;
трансцендентные функции: SQR, SIN, COS, ATN, LOG, LOG10, EXP;
функции интерпретатора: SYS, RCTRLO, ABORT, TTYSET, CTRLC, RCTRLC, TAB, RND, ABS, SGN, BIN, OCT, LEN, ASC, CHR$, POS, SEG$, VAL, TRM$, STR$, PI, INT, DAT$, CLK$;
арифметика с двойной точностью;
расширенные сообщения об ошибках;
работа БЕЙСИК в оперативном и фоновом режимах;
средства оверлейности.

Процедура включения всех перечисленных средств расширения, кроме функций интерпретатора, описана в главе «БЕЙСИК. ОПИСАНИЕ ЯЗЫКА».

Функции интерпретатора включаются пользователем при запуске БЕЙСИК. Отказ от использования средств расширения позволяет экономить объем памяти и ускорить выполнение программ.

ПРИМЕЧАНИЕ. В данном документе предполагается, что команды, сообщения, данные или любой текст, вводимый пользователем с терминала, заканчиваются нажатием клавиши <ВК>.

3. ОБРАЩЕНИЕ К ПРОГРАММЕ

Интерпретатор языка БЕЙСИК может работать под управлением однозадачного SJ-, фоново/оперативного FB-мониторов, монитора с управлением памятью — XM.

При использовании FB- или XM-мониторов БЕЙСИК запускается как оперативное или фоновое задание.

3.1. Фоновый режим

Запуск интерпретатора БЕЙСИК в фоновом режиме и под управлением SJ-монитора допускается производить двумя способами:

1) набрать команду .BASIC;
2) набрать команду .R [DEV:] BASIC;

где DEV — имя устройства содержащего файл интерпретатора BASIC.

Пример:

.RUN MY1:BASIC

с устройства MY1: запускается версия BASIC.SAV.

При запуске БЕЙСИК могут возникнуть следующие ошибки:

?KMON—F—FILE NOT FOUND
?KMON—F—NOT ENOUGH MEMORY
или
?NOT ENOUGH MEMORY FOR BASIC

Пример:

Запуск интерпретатора БЕЙСИК в фоновом режиме.

.BASIC
BASIC / 00 V.02
OPTIONAL FUNCTIONS (ALL, NONE, OR INDIVIDUAL)?

Ответ «A» включает все средства расширения функций интерпретатора. Ответ «N» не включает средства расширения. Ответ «I» включает отдельные средства расширения.

Пример.

OPTIONAL FUNCTIONS (ALL, NONE, OR INDIVIDUAL)? I
SYS? N

RCTRLO? N
ABORT? N
TTYSET? N
CTRLC & RCTRLC? N
TAB? Y
RND? Y
ABS? Y
SGN? Y
BIN? Y
OCT? Y
LEN? Y
ASC? Y
CHR$? N
POS? N
SEG$? N
VAL? N
TRM$? Y
STR$? Y
PI? Y
INT? Y
DAT$? N
CLK$? N
READY

при ответе «Y» данная функция включается в интерпретатор БЕЙСИК, при ответе «N» — данная функция не включается в интерпретатор.

3.2. Оперативный режим

Запуск интерпретатора БЕЙСИК в оперативном режиме выполняется командой:

.FRUN [DEV:] N:M

где DEV — имя рабочей версии интерпретатора;
M — размер памяти для пользователя, не менее 1000 слов.
N — ключевое слово.

Пример:

Запуск интерпретатора БЕЙСИК в оперативном режиме.

```
.FRUN BASIC/N:3000

E>
BASIC / FODOS V.02
OPTIONAL FUNCTIONS (ALL, NONE, OR INDIVIDUAL)?
```

ФОДОС-2 печатает «.» и «F>». Это указывает на то, что следующее за «.» и «F>» сообщение печатается оперативным заданием. На вопрос

OPTIONAL FUNCTIONS (ALL, NONE, OR INDIVIDUAL)?

Набрать <СУ/F> и один из вариантов ответа (A, N или I).

3.3. Запуск интерпретатора БЕЙСИК командным файлом

Запуск интерпретатора БЕЙСИК командным файлом выполняется командой:

.@FILE

где FILE — спецификация командного файла.

Командный файл может быть создан любым редактором текста или с помощью команды монитора COPY.

Пример создания командного файла MINRUN.COM:

```
.COPY TT: MY1: MINRUN. COM
.R BASIC
.I
```

далее в каждой строке этого файла указываются ответы, согласно приведенного примера: Y,Y,N,Y,N,N,N,Y,Y,Y,N,N,N,N,N, Y,N,N,Y,Y,Y,N,Y.

Создание файла завершается управляющим кодом СУ/Z.

Пример:

Запуск интерпретатора БЕЙСИК командным файлом MINRUN.

```
.@MINRUN
.R BASIC
BASIC / FODOS V.02
OPTIONAL FUNCTIONS (ALL, NONE, OR INDIVIDUAL)? I
SYS? Y
RCTRLO? Y
ABORT? N
TTYSET? Y
CTRLC & RCTRLC? N
TAB? N
RND? N
ABS? Y
SGN? Y
BIN? Y
OCT? N
LEN? N
ASC? N
CHR$? N
POS? N
SEG$? Y
VAL? N
TRM$? N
STR$? Y
PI? Y
INT? Y
```

DAT$? N
CLK$? Y
READY

3.4. Останов программ БЕЙСИК

Команда <СУ/С>, набранная дважды, останавливает выполнение программы немедленно. Интерпретатор печатает сообщение:
STOP AT LINE XXXXX
где XXXXX — номер строки программы, на которой произошло прерывание.

Команда <СУ/С>, набранная один раз, продолжает выполнение программы до тех пор, пока БЕЙСИК не встретит оператор ввода данных. Выполнение программы прерывается и интерпретатор печатает сообщение: STOP AT LINE XXXXX. Команда <СУ/С> не возвращает управление монитору. Возврат управления монитору выполняется командой BYE.
При работе БЕЙСИК в оперативном режиме набрать команду:
UNLOAD FG
в ответ на сообщение «B>».

Пример:

```
BYE
B>
UNLOAD FG
```

4. ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

Формат спецификации файлов интерпретатора языка БЕЙСИК: [DEVICE:][FILE][.TYPE]
где DEVICE — имя устройства, на котором находится файл;
FILE — имя файла (от одного до шести символов);
TYPE — тип файла.

В табл. 1 перечислены имена устройств, используемых интерпретатором языка БЕЙСИК.

Таблица 1

Наименование устройства	Назначение
1	2
DX, DXN	Гибкий диск с одинарной плотностью записи информации диаметром 207 мм
MX, MXN	Гибкий диск с одинарной плотностью записи информации диаметром 133 мм

1	2
MY, MYN	Гибкий диск с удвоенной плотностью записи информации диаметром 133 мм
LP	Устройство печати
TT	Системный терминал
SY, SYN	Логическое имя устройства, с которого загружена система
DK, DKN	Логическое имя устройства используется по умолчанию

Если формат спецификации файла не указан полностью, БЕЙСИК определяет отсутствующие элементы по умолчанию, приведенные в табл. 2 и табл. 3.

Таблица 2

Оператор или команда	Имя файла по умолчанию
1	2
SAVE, REPLACE, COMPILE OLD, APPEND, CHAIN, OVERLAY UNSAVE, OPEN, KILL, NAME	Имя текущей программы NONAME Система печатает сообщение об ошибке ?IFS

Таблица 3

Оператор или команда	Тип файла по умолчанию	
	Одинарная точность	Двойная точность
1	2	3
OPEN, KILL, NAME	.DAT	.DAT
SAVE, UNSAVE REPLACE, APPEND	.BAS	.BAS
COMPILE	.BAC	.BAX
RUN, OLD	.BAC (.BAS, если не найден файл с типом .BAC)	.BAX (.BAS, если не найден файл с типом .BAX)

5. ДОПОЛНИТЕЛЬНЫЕ ФУНКЦИИ ИНТЕРПРЕТАТОРА БЕЙСИК

5.1. Функция TTYSET. Функция TTYSET устанавливает характеристики для терминала пользователя.

Формат функции:
[LET] VAR=TTYSET(255%,EXP)
где VAR — переменная;
EXP — выражение, устанавливающее длину вводимой строки для данного терминала на значение EXP минус единица.

В результате интерпретатор автоматически выводит последовательность «ВК» и «ПС», если EXP-1 символов были напечатаны, и следующий символ печатается на другой строке.

Пример:

```
A=TTYSET(255%,81%)
```

Если первый аргумент отличен от 255%, или значение EXP равно 1 или больше 255%, интерпретатор печатает сообщение об ошибке:

```
?ARGUMENT ERROR (?ARG)
```

5.2. Функция RCTRLO. Команда <СУ/О> прекращает вывод данных на терминал до тех пор, пока не будет набрана повторно.

Функция RCTRLO подавляет действие команды <СУ/О>.
Формат функции:
[LET] VAR=RCTRLO
где VAR — переменная. Содержит неопределенное значение после выполнения оператора.

Пример:

```
 10 OPEN «DATA» FOR OUTPUT AS FILE #1
20 FOR I=1 TO 100 \ PRINT #1,I \ NEXT I
30 CLOSE #1
40 OPEN «DATA» FOR INPUT AS FILE #3
50 PRINT «Данные в файле:»
60 IF END #3 THEN 110
70 INPUT #3,D
80 PRINT D
90 T=T+D
100 GO TO 60
110 A=RCTRLO
120 PRINT «Сумма = »;T
READY
RUNNH
Данные в файле:
1
2
3
4
<СУ/О>                    — вывод данных прекращается.
```

```
Сумма = 5050          — вывод данных возобновляется
                        после выполнения строки 110.
READY
```

5.3. Функции CTRLC & RCTRLC

Функция RCTRLC блокирует действие команды <СУ/С>.
Формат функции: [LET] VAR=RCTRLC
где VAR — переменная. Содержит неопределенное значение после выполнения оператора.

Функция CTRLC разрешает действие команды <СУ/С>.
Формат функции: [LET] VAR=CTRLC
где VAR — переменная. Содержит неопределенное значение после выполнения оператора.

Пример:

```
1000 REM прерывание запрещено
1010 A=RCTRLC
1015 FOR I=1 TO 1000 \ S=S+I \ NEXT I
1020 PRINT «Нет прерывания»
1100 REM прерывание разрешено
1110 A=CTRLC
1120 PRINT «Есть прерывание»
1130 FOR I=1 TO 1000 \ S=S+I \ NEXT I
32767 END
READY
RUNNH
<СУ/С><СУ/С>
Нет прерывания
Есть прерывание
<СУ/С><СУ/С>
STOP AT LINE 1130
READY
```

5.4. Функция ABORT. Функция ABORT завершает выполнение программ и, в зависимости от значения аргумента, удаляет или оставляет программу в памяти.

Формат функции:
[LET] VAR=ABORT(EXP)
где VAR — переменная. Содержит неопределенное значение после выполнения оператора;
EXP — выражение. Принимает значение 0 или 1.
0 — программа, после завершения, остается в памяти.
1 — программа, после завершения, стирается из памяти, оставляя заголовок NONAME.

Пример:

```
10 PRINT «123»
20 A=ABORT(1)
```

```
30 PRINT «456»
RUNNH
123
READY
LIST
NONAME 21—JUL—87 14:53:30
READY
```

5.5. Функция SYS

Формат функции: [LET] VAR=SYS(EXP1,EXP2)

где VAR — переменная;

EXP1 — определяет выполняемую функцию;

EXP2 — произвольный аргумент.

В табл. 4 перечислены допустимые значения EXP1.

Таблица 4

EXP1	Назначение
1	2
1	Обрабатывается вводимый с терминала символ и возвращается его семибитное значение в коде КОИ-7 переменной VAR.
4	Возвращается управление монитору.
6	При блокировании действия <СУ/С> функцией RCTRLC интерпретатор анализирует, была ли набрана комбинация <СУ/С> во время выполнения программы. Переменной VAR возвращается значение ноль, если комбинация <СУ/С> не набрана, и 1, если комбинация <СУ/С> набрана.
7	Действие функции SYS(7) зависит от значения произвольного аргумента EXP2. Символы передаются на терминал без изменения, если EXP2 равно 0. Символы нижнего регистра преобразуются на эквивалентные символы верхнего регистра, если EXP2=1.

5.6. Примеры использования функции SYS

Пример 1.

Функция SYS(1).

```
20 PRINT «Напечатайте символ:»;SMR(A)
10 A=SYS(1)
40 PRINT «Значение »;CHR$(A);«В коде КОИ—7 равно »;
50 PRINT A
RUNNH
Напечатайте символ: Z
Значение Z в коде КОИ—7 равно 90
READY
```

Пример 2.

Функция SYS(4).

```
10 PRINT «До свидания»
20 A=SYS(4)
RUNNH
До свидания
```

Пример 3.

Функция SYS(6)

```
10 A=RCTRLC \ REM запрещение СУ/С
30 B=SYS(6) \ REM проверка запрещения СУ/С
40 IF B=1 THEN 100
50 PRINT «Программа выполняется»
60 GO TO 30
100 PRINT «Программа завершена»
110 A=CTRLC \ REM разрешение СУ/С
120 A=ABORT(1)
RUNNH
Программа выполняется
      .
      .
      .
Программа выполняется
<СУ/С><СУ/С>
Программа завершена
READY
STOP
READY
```

Пример 4.

Функция SYS(7,EXP2).

```
10 REM проверка функции SYS(7,EXP2)
20 PRINT «Y,N »; \ REM разрешить нижний регистр
30 INPUT A$
40 IF A$=«Y» THEN 100
50 IF A$<>«N» THEN 20
60 A=SYS(7,1) \ REM нижний регистр запрещен
70 GO TO 32767
100 A=SYS(7,0) \ REM нижний регистр разрешен
32767 END
RUNNH
Y,N ? Y
READY
Нижний регистр разрешен
```

6. ВКЛЮЧЕНИЕ ПРОГРАММ НА ЯЗЫКЕ АССЕМБЛЕР

БЕЙСИК позволяет включать программы, написанные на языке АССЕМБЛЕР (ПЯА), для расширения возможностей интерпретатора.

В настоящем разделе описаны:

— включение программ на языке АССЕМБЛЕР в интерпретатор БЕЙСИК;

— пересылка аргументов с помощью оператора CALL;

— использование дополнительных программ интерпретатора в программах пользователя.

6.1. Включение программ пользователя в интерпретатор БЕЙСИК

Для включения программ на языке АССЕМБЛЕРа используется файл BSCLI.MAC, содержащий таблицу указателей точек входа программ в интерпретатор. Структура таблицы указателей приведена на рис. 1.

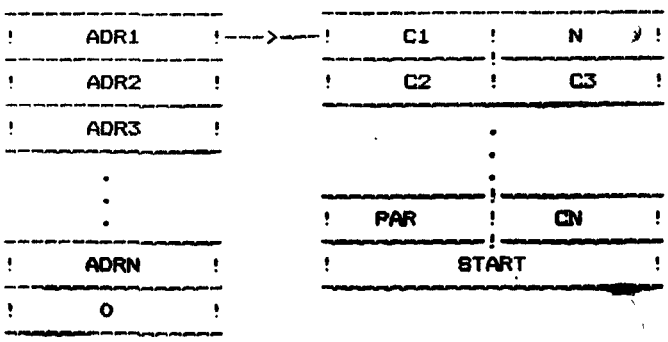


Рис. 1

C1 — первый символ имени программы.
CN — N-ый символ имени программы.
N — число символов в имени программы.
PAR — контрольный байт.
START — стартовый адрес программы.
ADR1 — точка входа 1-ой программы.
ADRN — точка входа N-ой программы.

Есть два варианта включения ПЯА в интерпретатор БЕЙСИК.

Вариант 1.

Имя программы пользователя определяется в самой программе. В этом случае таблица указателей файла BSCLI.MAC выглядит следующим образом:

```
                .
                .
                .
        .GLOBL INITNM, ADDNM, ROUTNM
FTABI:  .WORD FTBL
FTBL:   .WORD INITNM ;программа пользователя
        .WORD ADDNM
        .WORD ROUTNM
        .WORD 0
                .
                .
                .
        .END
```

где INITNM, ADDNM, ROUTNM — точки входа программ пользователя INITIT, ADDER, ROUTUS.

Пример.

Программа ADDER.

```
        ;Программа ADDER
        .GLOBL ADDNM
ADDNM:  .BYTE 5             ;число символов в имени
                            ;программы
        .ASCII «ADDER»
        .EVEN
        .WORD ADDST
ADDST:          .           ;начало программы
                .
                .
                .
```

Вариант 2.

Имя и адрес запуска программ пользователя определяется в конце таблицы указателей файла BSCLI.MAC.

```
                .
                .
        .GLOBL INITST, ADDST, ROUTST
FTABI:  .WORD FTBL
FTBL:   .WORD INITNM
        .WORD ADDNM
        .WORD ROUTNM
        .WORD 0
INITNM: .BYTE 6             ;количество символов
        .ASCII «INITIT»     ;в имени программы
```

```
        .EVEN
        .WORD INITST
ADDNM:  .BYTE 5
        .ASCII «ADDER»
        .EVEN
        .WORD ADDST
ROUTNM:.BYTE 6
        .ASCII «ROUTUS»
        .EVEN
        .WORD ROUTST
                .
                .
                .
        .END
```

Пример:

```
Программа ADDER.
        ; Программа ADDER
        .GLOBL ADDST
ADDST:      .                   ; начало программы
            .
            .
```

Таблица указателей файла BSCLI.MAC корректируется текстовым редактором. Таблица начинается меткой FTBL и оканчивается директивой «.WORD 0».

При написании программы ПЯА необходимо придерживаться требований, предъявляемых к составлению программ на языке АССЕМБЛЕР.

6.2. Пересылка аргументов оператором CALL

Оператор CALL вызывает программы пользователя, включенные в интерпретатор БЕЙСИК. При выполнении оператора CALL интерпретатор передает ПЯА значения аргументов и описание их типов, как показано на рис. 2.

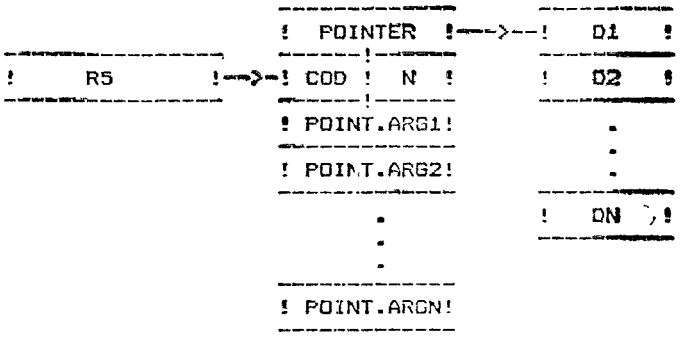


Рис. 2

D1 — дескриптор 1-го аргумента.
DN — дескриптор N-го аргумента.
POINTER — указатель списка дескрипторов.
COD — код 202.
N — количество аргументов.
POINT.ARG1 — указатель 1-го аргумента.
POINT.ARGN — указатель N-го аргумента.
R5 — регистр R5.

Регистр R5 содержит адрес слова, младший байт которого определяет количество аргументов в операторе CALL, старший байт — код 202.

В табл. 5 показаны значения битов слова дескриптора аргумента. Если младший бит слова дескриптора равен 0, то слово дескриптора является указателем дескриптора массива или строки. Бит 7-ой устанавливается в 1, если ПЯА не возвращает значение аргумента через оператор CALL. Бит 7-ой устанавливается в 0, если ПЯА возвращает значение аргумента.

Таблица 5

Биты	Значение	Тип аргумента
1	2	3
0	1	Числовая константа
	1	Ноль — аргумент
	0	Числовой массив
	0	Строковая константа
	0	Строковый массив
1—6 (тип данных)	11	Целое число
	20	Число с пл. запятой одинарной точности
	21	Число с пл. запятой двойной точности
	40	Строка
	77	Ноль — аргумент
7	0	Переменная
	1	Выражение
	1	Ноль — аргумент
8—12 (формат данных)	2	Целое число
	2	Строка
	4	Число с пл. запятой одинарной точности
	10	Число с пл. запятой двойной точности
	0	Ноль — аргумент
13—15	0	Элемент массива
	0	Ноль — аргумент
	1	Массив

6.2.1. Строки и строковые массивы

На рис. 3 приведен формат дескрипторов массивов и строковых аргументов.

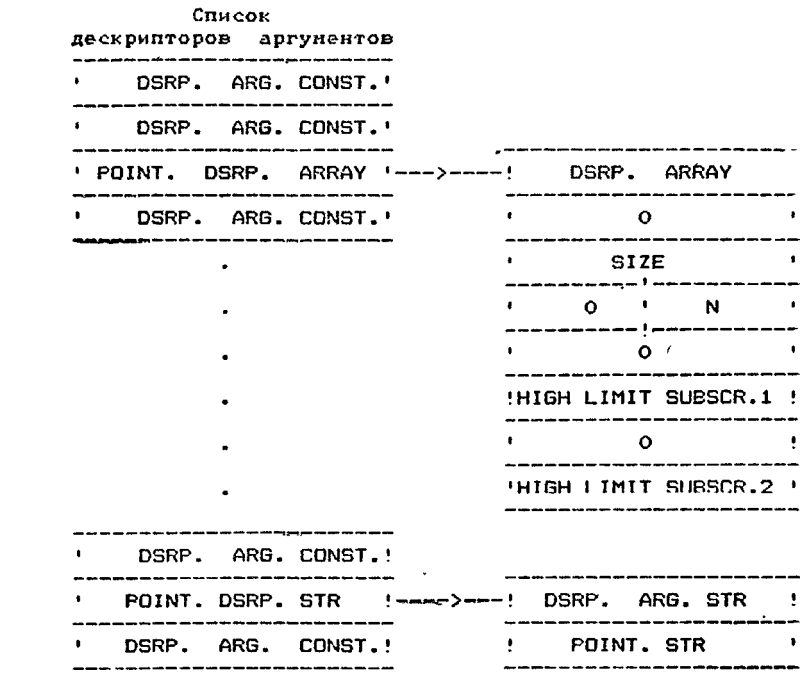


Рис. 3

DSRP. ARRAY — дескриптор массива.
SIZE — размер массива (в байтах).
N — количество индексов массива.
HIGH LIMIT SUBSCR.1 — верхняя граница 1-го индекса.
HIGH LIMIT SUBSCR.2 — верхняя граница 2-го индекса.
DSRP. ARG. STR — дескриптор строкового аргумента.
POINT. STR — указатель обращения к строке.
POINT. DSRP. ARRAY — указатель дескриптора массива.
POINT. DSRP. STR — указатель дескриптора строкового скаляра.
DSRP. ARG. CONST. — дескриптор аргумента (числовая константа).

Указатель обращения к строке строкового массива (POINT. STR) вычисляется по формуле:

POINT. STR=2*OFFSET+POINT.ARG

где OFFSET — смещение (позиция элемента относительно начала массива).

Указатель аргумента (POINT.ARG) для строкового массива указывает начало массива. Смещение элемента одномерного массива равно значению его индекса. Смещение элемента двумерного массива определяется по формуле:
OFFSET=SUMBSCR.1*(MAX SUBSCR.2+1)+SUBSCR.2

Пример.

Определение указателя обращения к строке на примере массивов A$(5) и B$(3,10). Указатели аргументов — A и B соответственно.

Элемент массива	Формула вычисления	Указатель обращения к строке
A$(0)	2*0+A	A
A$(5)	2*5+A	10+A
B$(3,5)	2*(3*11+5)+B	76+B
B$(0,1)	2*(0*11+1)+B	2+B
B$(2,10)	2*(2*11+10)+B	64+B

6.2.2. Программы доступа к строковому аргументу

В состав интерпретатора БЕЙСИК входят программы доступа к строкам, которые используются программами, написанными на языке АССЕМБЛЕРа:

1) $FIND
2) $ALC
3) $STORE
4) $DEALC

При написании ПЯА необходимо придерживаться определенной последовательности в использовании строковых аргументов:

1) обращение к строке с помощью функции $FIND, которая определяет длину строки и указатель 1-го символа;

2) создание временной строки в стеке функцией $ALC для записи промежуточных результатов работы;

3) чтение символов строкового аргумента, обработка их и запись во временную строку;

4) перезапись временной строки в исходный строковый аргумент функцией $STORE;

5) удаление из стека временной строки функцией $DEALC.

Регистр R5, перед выполнением перечисленных функций, должен содержать адрес слова, младший байт которого определяет число аргументов в операторе CALL, старший байт — код 202.

Программы доступа к строкам вызываются инструкцией:
JSR PC, имя программы

6.3. Использование внутренних программ интерпретатора БЕЙСИК

6.3.1. Программа обработки ошибок $ARGER

Программа печатает сообщение интерпретатора при возникновении неустранимой ошибки:

?ARGUMENT ERROR AT LINE XXXXX

или

?ARG AT LINE XXXXX

где XXXXX — номер строки с оператором CALL.

Обращение к программе:

JMP $ARGER

При использовании оператора CALL в непосредственном режиме сообщение не выдается. Интерпретатор печатает READY.

6.3.2. Программа $BOMB

Программа печатает сообщение пользователя при возникновении неустранимой ошибки:

?Сообщение об ошибке AT LINE XXXXX

где XXXXX — номер строки с оператором CALL.

Обращение к программе:

```
JSR R1, $BOMB
.ASCII «Сообщение»
.EVEN
```

При использовании оператора CALL в непосредственном режиме сообщение не выдается. Интерпретатор печатает READY.

6.3.3. Программа $MSG

Программа печатает сообщение пользователя и передает управление инструкции, следующей за директивой .EVEN.

Обращение к программе:

```
JSR R1, $MSG
.ASCII «Сообщение»
.BYTE 15,12,0
.EVEN
```

6.3.4. Программа $CHROT

Программа $CHROT печатает на терминале символ, засылаемый пользователем в младший байт регистра R0.

Обращение к программе:

JSR PC,$CHROT

6.3.5. Стандартные подпрограммы

В табл. 6 приведены стандартные подпрограммы интерпретатора БЕЙСИК.

Таблица 6

Операция, функция	Оператор	Обозначение	Подпрограммы одинарной точности	Подпрограммы двойной точности
1	2	3	4	5
Сложение	+	C=A+B	$ADR	$ADD
Вычитание	—	C=A—B	$SBR	$SBD
Умножение	*	C=A*B	$MLR	$MLD
		C%=A%*B%	$ML	$MLI
Деление	/	C=A/B	$DVR	$DVD
		C%=A%/B%	$DV	$DVI
Возведение в степень	∧	C=A∧B	XFF$	XDD$
		C=A∧B%	XFI$	XDI$
		C%=A%∧B%	XII$	XII$
Преобразование типа данных		B%=A	$RI	$DI
		B=A%	$IR	$ID
Функция усечения		B=SGN(A)*INT(ABS(A))	$INTR	$DINT
Синус		B=SIN(A)	SIN	DSIN
Косинус		B=COS(A)	COS	DCOS
Арктангенс		B=ATN(A)	ATAN	DATAN
Логарифм		B=LOG(A)	ALOG	DLOG
		B=LOG10(A)	ALOG10	DLOG10
Квадратный корень		B=SQR(A)	SQRT	DSQRT
Экспоненциальная функция		B=EXP(A)	EXP	DEXP

Стандартные подпрограммы со знаком денежной единицы «$» вызываются программой $POLSH. Обращение к программе
$POLSH выполняется инструкцией:
JSR R4,$POLSH,
за которой следует список вызываемых подпрограмм. Программа $UNPOL завершает список.

Пользователь определяет имена программ $POLSH, $UNPOL и программ списка как глобальные в ПЯА.

Аргументы между программой пользователя (ПЯА) и стандартными подпрограммами передаются через стек. При занесении аргументов в стек чисел с плавающей запятой двойной и одинарной точности первой заносится младшая часть числа (одно или два слова), затем старшая часть числа (одно или два слова). Аргументы заносятся в той же последовательности, в какой они приведены в обозначении подпрограмм (см. табл. 5).

После выполнения операций результат заносится на место аргумента A. После выполнения операций подпрограммами $R1, $DI результат заносится на место младшей части аргумента A.

Программа ПЯА создает список аргументов перед обращением к подпрограммам SIN, COS, ATAN, ALOG, ALOG10, SQRT, EXP (рис. 4 и рис. 5). Результат вычислений записывается в регистры R0 и R1 (одинарная точность) и R0,R1,R2, R3 (двойная точность).

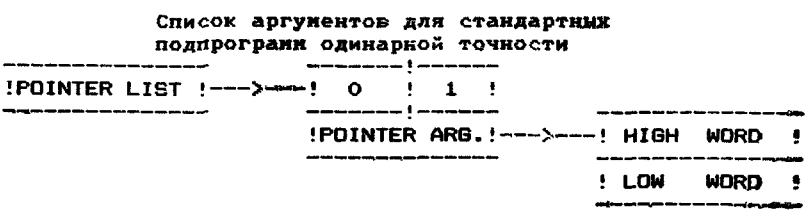


Рис. 4

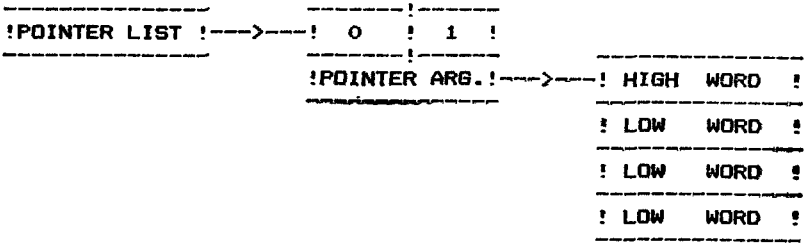


Рис. 5

POINTER LIST — указатель списка (регистр R5).
POINTER ARG. — указатель аргумента.
HIGH WORD — старшее слово аргумента.
LOW WORD — младшее слово аргумента.

Пример.

Составление программы на языке АССЕМБЛЕР.
Обращение к программе:
CALL HYPOT(A,B,C,C%)
Текст программы:

```
        .TITLE  HYPOT
        .PSECT  SUBRS,RO,I
        .GLOBL  HYPTAB
```

```
HYPTAB: .BYTE   5                       ;определяется имя
        .ASCII  'HYPOT'                 ;программы
        .EVEN
        .WORD   HYPOT
        .GLOBL  $MLR,XFI$, $ADR, SQRT,$RI
HYPOT:  CMP     (R5)+,#4
        BEQ     20$
10$:    JMP     $ARGER
20$:    CMPB    (R5)+,#202
        BNE     60$
        MOV     SP,R3
        SUB     #30.,R3
        CMP     R3,R4
        BHIS    30$
        JSR     R1,$BOMB
        .ASCIZ  'переполнение стека'
        .EVEN
30$:    MOV     -4(R5),R4               ;проверка соот-
        JSR     PC GETDSC               ;ветствия пересы-
        BIC     #160201,R3              ;лаемых аргумен-
        CMP     #2040,R3                ;тов
        BNE     10$
        JSR     PC,GETDSC
        BIC     #160201,R3
        CMP     #2040,R3
        BNE     10$
        JSR     PC,GETDSC
        BIC     #160001,R3
        CMP     #2040,R3
        BNE     10$
        JSR     PC,GETDSC
        BIC     #160001,R3
        CMP     #1022,R3
        BNE     10$
60$:    MOV     (R5)+,R3                ;засылка аргумен-
        MOV     2(R3),-(SP)             ;тов для работы
        MOV     (R3),-(SP)              ;программы
        MOV     2(R3),-(SP)             ;$MLR
        MOV     (R3),-(SP)
        JSR     R4,$POLSH
        $MLR
        $UNPOL
        MOV     (R5)+,R3                ;засылка аргу-
        MOV     2(R3),-(SP)             ;ментов для ра-
                                        ;боты программ
```

```
        MOV     (R3),—(SP)          ;XFI$, $ADR
        MOV     #2,—(SP)
        JSR     R4,$POLSH
        XFI$
        $ADR
        $UNPOL
        MOV     R5,—(SP)
        MOV     SP,R5
        TST     (R5)+
        MOV     R5,—(SP)
        MOV     #1,—(SP)
        MOV     SP,R5
        JSR     PC,SQRT
        CMP     (SP)+,(SP)+
        MOV     (SP)+,R5            ;засылка резуль-
        MOV     (R5)+,R3            ;тата
        MOV     R0,(R3)+            ;вычислений в
        MOV     R1,2(SP)            ;C,C%
        MOV     R0,(SP)
        JSR     R4,$POLSH
        $RI
        $UNPOL
        MOV     (SP)+,@(R5)+
        RTS     PC
GETDSC: MOV     (R4)+,R3
        BIT     #1,R3
        BNE     10$
        MOV     (R3),R3
10$:    RTS     PC
        .END
```

7. СООБЩЕНИЯ

Интерпретатор языка БЕЙСИК в процессе работы осуществляет синтаксическую проверку введенных операторов и команд, и в случае обнаружения ошибок выводит сообщения о них (приложения 1, 2).

ПРИЛОЖЕНИЕ 1

Сообщения об ошибках БЕЙСИК

?ARGUMENT ERROR (?ARG) — аргументы в вызове функции не соответствуют числу или типу аргументов, опре-

деленных для данной функции.

?ARRAYS TOO LARGE (?ATL) — мал объем памяти для массива, указанного в операторе DIM.

?BAD DATA READ (?BDR) — элементы данных, вводимых оператором DATA, не соответствуют типу данных в операторе READ.

?BAD DATA — RETYPE FROM ERROR (?BRT) — данные, вводимые оператором INPUT, не соответствуют типу данных, указанных в этом операторе.

?BAD LOG (?BLG) — аргумент функции LOG или LOG10 равен нулю или отрицательному значению.

?BUFFER STORAGE OVERFLOW (?BSO) — превышен объем программы пользователя.

?CANNOT DELETE FILE (?CDF) — в командах UNSAVE и KILL указан несуществующий файл.

?CHANNEL ALREADY OPEN (?CAO) — оператор OPEN открывает один из двенадцати каналов ввода-вывода, который был уже открыт.

?CHANNEL I/O ERROR (?CIE) — произошла ошибка ввода-вывода

?CHANNEL NOT OPEN (?CNO) — программа выполняет операцию ввода-вывода на одном из двенадцати каналов, который предварительно не был открыт.

?CHECKSUM ERROR IN COMPILED PROGRAM (?CCP) — ошибка, возникшая при работе оператора COMPILE.

?COMMON OUT OF ORDER (?COO) — нарушена последовательность аргументов в операторе COMMON.

?CONTROL VARIABLE OUT OF RANGE (?CVO) — неверно указан номер строки в операторах управления программой.

?DIVISION BY ZERO (?DVO) — деление на ноль.

?END NOT LAST (?ENL) — оператор END не последний в программе.

?ERROR CLOSING CHANNEL (?ECC) — ошибка, возникшая при работе операторе CLOSE.

?EXCESS INPUT IGNORED (?EII) — при работе оператора INPUT вводится больше данных, чем требуется.

?EXPONENTIATION ERROR (?ERR) — неверные аргументы для функции EXP.

?EXPRESSION TOO COMPLEX (?ETC) — вычисляемое выражение переполняет стек.

?FILE ALREADY EXISTS (?FAE) — оператор OPEN FOR INPUT открывает открытый файл.

?FILE NOT FOUND (?FNF) — требуемый файл не находится на определенном устройстве.

?FILE PRIVILEGE VIOLATION (?FPV) — попытка нарушить защиту файла.

?FLOATING OVERFLOW (?FOV) — переполнение в результате вычислений с плавающей запятой. БЕЙСИК присваивает значение 0 и продолжает выполнение.

?FLOATING UNDERFLOW (?FUN) — значение результата вычислений меньше, чем 10^—38. БЕЙСИК присваивает значение 0 и продолжает выполнение.

?FOR WITHOUT NEXT (?FWN) — программа содержит оператор FOR без соответствующего оператора NEXT.

?FUNCTION ALREADY DEFINED (?FAD) — функция пользователя FN определена повторно.

?ILLEGAL CHANNEL NUMBER (?ICN) — указанный номер канала не входит в диапазон 1—12.

?ILLEGAL DEF (?IDF) — недопустимый формат оператора DEF.

?ILLEGAL DIM (?IDM) — синтаксическая ошибка в операторах DIM или COMMON.

?ILLEGAL END OF FILE IN COMPILED PROGRAM (?IEF) — недопустимый конец файла скомпилированной программы. Ошибка, возникшая при работе оператора COMPILE.

?ILLEGAL FILE LENGTH (?IFL) — размер файла в операторе OPEN превосходит максимально возможную длину.

?ILLEGAL FILE SPECIFICATION (?IFS) — недопустимая спецификация файла.

?ILLEGAL IN IMMEDIATE MODE (?IIM) — обращение к операторам INPUT и INPUT# в непосредственном режиме.

?ILLEGAL I/O DIRECTION (?IID) — попытка записи в файл, открытый для чтения, и наоборот.

?INCONSISTENT NUMBER OF SUBSCRIPTS (?INS) — обращение к двухмерному массиву как к одномерному, и наоборот.

?INPUT STRING ERROR (?ISE) — в символьной строке, вводимой оператором INPUT, отсутствуют вторые кавычки.

?INTEGER OVERFLOW (?IOV) — переполнение в результате вычислений целых чисел.

?LINE TOO LONG (?LTL) — символьная строка больше допустимой в языке БЕЙСИК.

?LINE TOO LONG TO TRANSLATE (?TLT) — символьная строка больше объема памяти, отведенной для трансляции.

?MISSING SUBPROGRAM (?MSP) — в операторе CALL указывается имя несуществующей программы.

?NEGATIVE SQUARE ROOT (?NGS) — попытка извлечь квадратный корень из отрицательного числа. БЕЙСИК присваивает значение 0 и продолжает выполнение.

?NESTED FOR STATEMENTS WITH SAME CONTROL VARIABLE (?FSV) — имеется два или больше вложенных операторов FOR с одной и той же переменной цикла.

?NEXT WITHOUT FOR (?NWF) — программа содержит оператор NEXT без соответствующего оператора FOR.

?NOT A VALID DEVICE (?NVD) — неверное имя устройства в спецификации файла.

?NOT ENOUGN ROOM (?NER) — устройство, к которому производится обращение, не

имеет достаточной свободной области для обслуживания файла.

?NUMBERS AND STRINGS (?NSM) — попытка выполнить действия с числовыми и строковыми выражениями совместно.

?OUT OF DATA (?OOD) — недостаточно данных в операторе DATA для оператора READ.

?PRINT USING ERROR (?PRU) — неверный формат оператора PRINT USING.

?PROGRAM TOO BIG (?PTB) — введенная программа больше объема памяти пользователя.

?RESEQUENCE ERROR (?RES) — ошибка, возникшая при выполнении оператора RESEQ.

?RETURN WITHOUT GOSUB (?RWG) — попытка выполнить оператор RETURN до выполнения GOSUB.

?STRING STORAGE OVERFLOW (?SSO) — недостаточно памяти для хранения программы.

?STRING TOO LONG (?STL) — символьная строка превышает 255 символов.

?SUBSCRIPT OUT OF BOUNDS (?SOB) — значение вычисленного индекса вне границы, указанной в операторе DIM, или меньше нуля.

?SUBSTITUTE ERROR (?SUB) — неверный формат оператора SUB.

?SYNTAX ERROR (?SYN) — интерпретатор БЕЙСИК встретил нераспознаваемую команду или оператор.

?TO MANY GOSUBS (?TMG) — число вложенных GOSUB превысило 20.

?TO MANY ITEMS IN COMMON (?TIC) — число аргументов оператора COMMON превышает 255.

?UNDEFINED FUNCTIONS (?UFN) — не определена функция пользователя FN.

?UNDEFINED LINE NUMBER (?ULN) — неопределенный номер строки.

?UNDIMENSIONED ARRAY IN CALL (?UAC) — массив, указанный в операторе CALL, не определен в операторе DIM.

?USE REPLACE (?RPL) — попытка сохранить файл, уже имеющийся на устройстве файловой структуры, командой SAVE.

?VIRTUAL ARRAY CHANNEL ALREADY IN USE (?VSU) — один и тот же номер канала используется в двух или более операторах DIM.

ПРИЛОЖЕНИЕ 2

УСЛОВИЯ ВОЗНИКНОВЕНИЯ ОШИБОК

При использовании функций в программах, написанных на языке БЕЙСИК, могут возникнуть ошибки, описанные в таблице.

Таблица

Функция	Сообщение об ошибках	Условия возникновения ошибки
1	2	3
Все	?ARGUMENT ERROR	Аргументы в вызове функции не соответствуют типу аргументов, определенных для данной функции.
Все	?SYNTAX ERROR	Количество аргументов не соответствует количеству, определенному для данной функции, или разделителем аргументов является не специальный символ.
ASC(STR)	?ARGUMENT ERROR	STR — не односимвольная строка.
BIN(STR)	?ARGUMENT ERROR	STR — строка, содержащая символы, отличные от пробела, 0 или 1, или значение STR больше 2↑16.
CHR$(EXP)	?ARGUMENT ERROR	Значение EXP вне допустимого диапазона (0—32767).
EXP(EXP)	?EXPONENTIATION ERROR	Значение EXP больше 87.
FNL	?UNDEFINED FUNCTION	Функция FNL не определена оператором DEF.
LOG(EXP)	?BAD LOG	Значение EXP равно нулю или отрицательному значению. БЕЙСИК присваивает значение 0 и продолжает выполнение.

1	2	3
LOG10(EXP)	?BAD LOG	Значение EXP равно нулю или отрицательному значению. БЕЙСИК присваивает значение 0 и продолжает выполнение.
OCT(STR)	?ARGUMENT ERROR	STR — строка, содержащая символы, отличные от пробела, 0—7, или значение STR больше 2/\16.
PI	?SYNTAX ERROR	Неверный формат функции.
SQR(EXP)	?NEGATIVE SQUARE ROOT	Значение EXP отрицательное. БЕЙСИК присваивает значение 0 и продолжает выполнение.
TAB(EXP)	?ARGUMENT ERROR	Значение EXP вне допустимого диапазона (0—32767).
VAL(STR)	?ARGUMENT ERROR	STR — строка, включающая другие символы, кроме цифр.

СОДЕРЖАНИЕ

АССЕМБЛЕР. ОПИСАНИЕ ЯЗЫКА

АССЕМБЛЕР. РУКОВОДСТВО ПРОГРАММИСТА

РЕДАКТОР СВЯЗЕЙ. РУКОВОДСТВО ОПЕРАТОРА

БИБЛИОТЕКАРЬ. РУКОВОДСТВО ОПЕРАТОРА

ОТЛАДЧИК И ВИРТУАЛЬНЫЙ ОТЛАДЧИК. РУКОВОДСТВО ОПЕРАТОРА

ПАСКАЛЬ. ОПИСАНИЕ ЯЗЫКА

ПАСКАЛЬ. РУКОВОДСТВО ПРОГРАММИСТА

БЕЙСИК. ОПИСАНИЕ ЯЗЫКА

БЕЙСИК. РУКОВОДСТВО ПРОГРАММИСТА

Ответственный за выпуск М. Г. Бойкова
Редактор Т. А. Савельева
Корректор В. Н. Лыткина

Изд. № 30. Сдано в набор 09.02.90. Подписано в печать 18.05.90.
Формат 60×84¹/₁₆. Бум. тип. № 1. Гарнитура литературная.
Печать высокая. Объем 22,4 усл. печ. л. Тираж 20 000 экз. Заказ 399
Бесплатно.

Ленинградское отделение РППО «Союзбланкиздат».

Великолукская городская типография управления издательств,
полиграфии и книжной торговли Псковского облисполкома,
182100, г. Великие Луки, ул. Полиграфистов, 78/12.