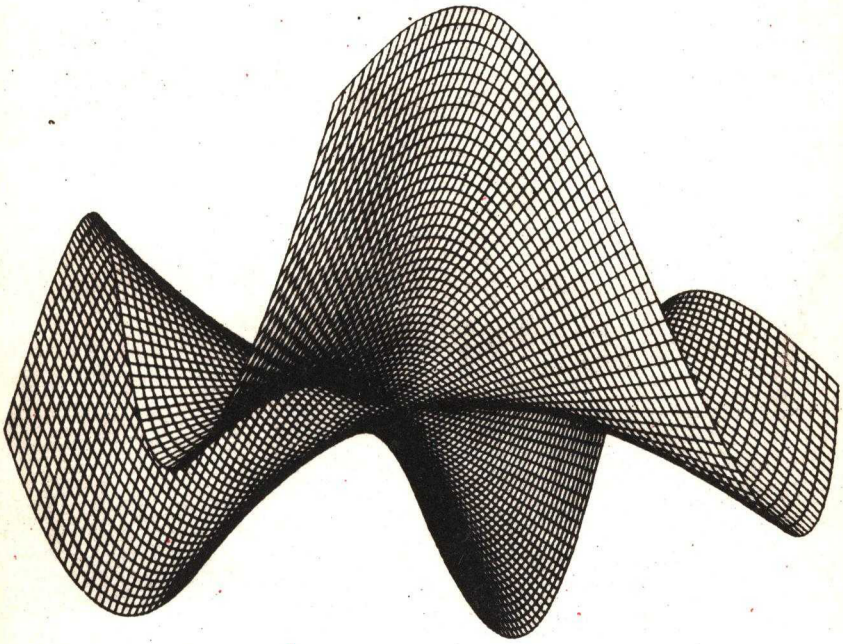
НАУЧНО-ПРОИЗВОДСТВЕННОЕ ОБЪЕДИНЕНИЕ
СИСТЕМ УПРАВЛЕНИЯ
MERASTER

УВК КАМАК – МЕРА 60
СМ 1633



ОПЕРАЦИОННАЯ СИСТЕМА RT 60 V04

СПРАВОЧНИК ПОЛЬЗОВАТЕЛЯ BASIC

ТОМ № 5, ЧАСТЬ 2

KATOWICE

СОДЕРЖАНИЕ стр.

ПЕРЕЧЕНЬ ТАБЛИЦ:

ПЕРЕЧЕНЬ РИСУНКОВ:

АННОТАЦИЯ

Настоящий справочник предназначен для пользователей, которым известен язык программирования BASIC (см. ОПИСАНИЕ ЯЗЫКА BASIC) и операционная система RT-60 (см. ВВЕДЕНИЕ В ОПЕРАЦИОННУЮ СИСТЕМУ RT-60 и СПРАВОЧНИК ПОЛЬЗОВАТЕЛЯ ОПЕРАЦИОННОЙ СИСТЕМЫ RT-60). Большинство свойств настоящей версии интерпретатора BASIC считается общими для более ранних вариантов BASIC (например для интерпретатора BASIC , работающего под управлением операционной системы SCR-60). Эти свойства подробно описаны в ОПИСАНИИ ЯЗЫКА BASIC.

В настоящем справочнике характеризуются те свойства интерпретатора BASIC, которые определенным образом зависят от самой операционной системы RT-60.

Эти свойства следующие:

- процедура запуска интерпретатора BASIC
- действие ключевой команды CTRL/C
- точность представления чисел
- формат сообщений об ошибках
- формат спецификации файлов
- функции параметров в операторе OPEN
- алгорифм управления файлами
- способ вытеснения файлов
- действие услуговых функций
- процедуры использования подпрограмм на языке макроассемблера
- процедуры завершения

Всем пользователям рекомендуется прочитать настоящее описание, за исключением раздела 4, который посвящен использованию подпрограмм на языке макроассемблера в программах BASIC.

От пользователя, включающего подпрограммы на языке макроасс-

емблера в BASIC , требуется опыт в программировании на языке MACRO , в компоновке и запуске программ.

Ниже представляются конвенции, нотации и символы, которые используются в последующих разделах документа.

Следующие символы (так называемые специальные ключи терминала) очень часто применяются в BASIC:

Символ	Способ ввода
CTRL/X	Одновременно нажать служебный ключ CTRL и клавишу с буквой, указанной за дробной (косой) чертой;
⟨CR⟩	Нажать клавишу CR;
⟨ESC⟩	Нажать клавишу ESC (ESCAPE или ALTMODE);
RUBOUT	Нажать клавишу RUBOUT (DEL) .

Дополнительно в справочнике используются некоторые символические обозначения, встречающиеся в спецификациях операторов, функций и команд, т.е. :

Обозначение	Описание
[ ]	Элементы, заключенные в квадратные скобки считаются необязательными; например: [LET] variable-expression
{ }	Выбор одного из двух или более возможных элементов; например: IF relational expression { THEN statement / THEN line number / GO TO line number }
. . .	Предшествующий элемент может повторяться указанным образом; например:

line number [CLOSE #] expr1,[#] expr2,...

элементы, определенные при помощи строчных букв, и специальные символы

Эти элементы необходимо специфицировать точно таким образом, в каком они специфицированы в формате; например:

LET
RUN
#

элементы, записываемые с помощью строчных букв называются ключевыми словами.

элементы, определенные при помощи прописных букв

Эти элементы заменяются согласно описанию, приведенному в тексте. Ниже представляются элементы, выражаемые с помощью прописных букв.

Элементы, представляемые с помощью прописных букв, в общем встречаются в описаниях формата. Значение каждого из этих элементов четко определено. Если в спецификации формата не указываются ограничения, накладываемые на элементы, предполагается, что они будут использоваться общепринятым образом. Отдельные элементы спецификации формата изложены в ОПИСАНИИ ЯЗЫКА BASIC.

Элемент	Условное обозначение	Описание
expression	expr	Любое допустимое выражение BASIC представляющее числовую величину, если это не определено иначе. Выражение может быть текстовым или числовым, например: (5 * SIN(X)) ^ Y

file specification	- - -	Спецификация файла(п.2.I.)
integer	**int**	Любая положительная константа целого типа или любая положительная константа, которая может принимать целый тип,если за ней будет следовать знак процента; например:5%,3%,2,7.
line number	- - -	Любой допустимый номер строки;например: I0,I00,32767.
string	- - -	Любое текстовое выражение ;например: **"ABC",C$+SEG$ (A$,3,4)**
variable	**var**	Переменная,тип которой может быть с плавающей запятой,целым или текствовым.

Если более чем один элемент,определённый при помощи прописных букв,выступает в спецификации формата,очередные слова будут пронумерованы;например:

CLOSE #expr1,#expr2,#expr3,.

Все строки,вводимые пользователем,должны быть закончены символом <CR>.

Поскольку пользователь получает отгенерированную стандартную версию интерпретатора **BASIC**,за специальной версией необходимо обратиться к поставщику системы **MERA-60**.Включение дополнительных свойств в **BASIC** во время генерации описывается лишь с целью пояснения способов расширения возможностей интерпретатора.

I. ПРОЦЕДУРЫ ЗАПУСКА BASIC

I.I. Необязательные элементы

BASIC обладает определенным числом необязательных элементов, которые можно включать в BASIC полностью или выбирая нужные. Как в ОПИСАНИИ ЯЗЫКА BASIC , так и в настоящем справочнике приведены все эти элементы. Исключив некоторые или все необязательные элементы, пользователь может уменьшить объем памяти, необходимой для выполнения программы, ускорить ее выполнение или одно и другое.

Необязательные элементы BASIC следующие:

операторы:

CALL

PRINT USING

команды:

SUB

RESEQ

функции:

SQR	SYS	ABS	SEG$
SIN	RCTRLO	SGN	VAL
COS	ABORT	BIN	TRM$
ATN	TTYSET	OCT	STR$
LOG	CTRLC	LEN	PI
LOG10	RCTRLC	ASC	INT
EXP	TAB	CHR$	DAT$
	RND	POS	CLK$

Другие элементы:

- арифметика с двойной точностью,
- длинная форма сообщений об ошибках,
- возведение в степень (т.е. выражение A^B)
- возможность запуска интерпретатора BASIC в качестве оперативной или фоновой задачи,
- свойства более экономного использования доступной области памяти и факторы, увеличивающие скорость выполнения программ.

Включение некоторых элементов производится во время компоновки самого интерпретатора BASIC. Элементы включаемые в процессе компоновки, следующее:

- все необязательные операторы,
- все необязательные команды,
- функции SQR, SIN, COS, ATN, EXP, LOG и LOG10,
- все другие необязательные свойства.

Во время запуска BASIC выбираются следующие необязательные функции:

SYS	ABS	SEG$
RCTRLO	SGN	VAL
ABORT	BIN	TRM$
TTYSET	OCT	STR$
CTRLC	LEN	PI
RCTRLC	ASC	INT
TAB	CHR$	DAT$
RND	POS	CLK$

Перед использованием интерпретатора BASIC необходимо произвести его компоновку вместе с необязательными элементами.

I.2. Запуск интерпретатора BASIC

BASIC может работать либо под управлением монитора SJ, либо под управлением монитора FB. Во втором случае BASIC можно запускать в качестве оперативной или фоновой задачи.

I.2.I. Запуск интерпретатора BASIC в системе SJ или в качестве фоновой задачи.

Перед запуском интерпретатора необходимо считать операционную систему и дополнительно можно ввести текущие время и дату (см. ВВЕДЕНИЕ В ОПЕРАЦИОННУЮ СИСТЕМУ RT-6o). Запуск осуществляется по команде МОНИТОРА.

```
.BASIC
```

Для нестандартных вариантов интерпретатора стартовая команда следующая:

```
.RUN file specification
```

где: `file specification` – спецификация файла, содержащего требуемую версию интерпретатора

Например, для запуска интерпретатора BAS 8K с устройства DX1: вместо стандартного варианта BASIC.SAV , следует ввести следующую команду:

```
.RUN DX1:BAS8K
```

Если специфицируемый файл не существует на указанном устройстве, системой печатается следующее сообщение об ошибке:

```
?KMON-F-File not found
```

Если доступная область памяти недостаточная для запуска интерпретатора BASIC выдается одно из следующих сообщений:

```
NOT ENOUGH MEMORY FOR BASIC
```

или

```
?KMON-F-Not euough memory
```

Чаще всего, это сообщение выдается в случае, если из памяти не удалена большая оперативная задача.

Если ошибки отсутствуют, BASIC печатает идентификационное сообщение и включает функции подобранные пользователем:

```
.BASIC
BASIC /RT-60 V02-035
OPTIONAL FUNCTIONS (ALL,NONE,OR INDIVIDUAL) ?
```

Для включения всех функций следует нажать клавишу A. Для отвержения этих функций следует на терминале пользователя нажать клавишу N. Каждый из этих ответов необходимо закончить символом <CR>. В ответ на введение пользователем A или N BASIC включает, или отвергает функции и печатает сообщение READY. Например:

```
OPTIONAL FUNCTIONS (ALL,NONE,OR INDIVIDUAL) ? A
READY
```

Нажатие только клавиши <CR> эквивалентно ответу A. Если пользователь желает выбрать некоторые функции из набора, он должен ответить I. Тогда BASIC печатает имена индивидуальных функций. Для включения функции пользователь должен ответить Y; в противном случае следует ответить N. Нажатие только клавиши <CR> эквивалентно ответу Y. Ошибочный ответ вызывает повторную распечатку запроса. Удовлетворение всем запросам интерпретатора заканчивается распечаткой сообщения READY.

Например:

```
BASIC-60/RT-60 V02-03$
OPTIONAL FUNCTIONS(ALL, NONE, OR INDIVIDUAL)? I
SYS? N
RCTRLO? N
ABORT? N
TTYSET? N
CTRLC &CTRLC? N
TAB? N
RND? Y
ABS? Y
SGN? Y
BIN? Y
OCT? N
LEN? N
ASC? N
CHR$? N
POS? N
SEG$? N
VAL? N
TRM$? N
STR$? N
PI? Y
INT? N
DAT$? N
CLK$? N

READY
```

I.2.2. Запуск интерпретатора BASIC в качестве оперативной задачи.

Для запуска интерпретатора в первом плане используется команда Монитора FRUN.

```
.FRUN file specification /BUFFER:number
```

где:

`file specification` – спецификация файла, содержащего требуемую версию интерпретатора

`number` – размер области пользователя (т.е. количество резервированных слов); размер должен составлять 1000. или более слов (здесь точка определяет десятичное число).

Спецификация размера области необходима для запуска интерпретатора BASIC .Её размер будет приблизительно на I00 слов больше,чем специфицированный.Например,следующей командой резервируется приблизительно 3I00 слов.

```
.FRUN  BASIC/ BUFFER:3000
```

Если специфицированный файл не существует,выдается следующее сообщение об ошибке:

```
?KMON-F-File not found
```

Если количество слов,отводимых в команде FRUN ,не удовлетворяет интерпретатору ,он печатает сообщение:

```
NOT ENOUGH MEMORY FOR BASIC
```

Если ошибки отсутствуют ,система печатает точку и F> ,которые указывают,что следующее сообщение будет выдаваться оперативной задачей.Затем BASIC выводит идентификационное сообщение и запросы на включение/отвержение функций.Например:

```
.FRUN BASIC/BUFFER:3000.

.
F>
BASIC-60/RT-60 V02-03
OPTIONAL FUNCTIONS (ALL, NONE, OR INDIVIDUAL)?
```

Пользователь должен набрать <CTRL/F> и один из вариантов ответа (A, N или I).

ПРИМЕЧАНИЕ

Если используется устройство, отличное от системного, прежде чем запустить BASIC в качестве оперативной задачи, необходимо загрузить соответствующий драйвер. Оперативная задача подробно описана в СПРАВОЧНИКЕ ПОЛЬЗОВАТЕЛЯ ОПЕРАЦИОННОЙ СИСТЕМЫ RT-60.

I.2.3. Запуск интерпретатора BASIC из промежуточного файла.

Для запуска интерпретатора и ответа на его начальные вопросы можно использовать промежуточный файл. Этот способ инициирования возможен только для однозадачного Монитора или во втором плане двухзадачного Монитора. Этот прием особенно пригодный, если отбор необязательных функций производится индивидуально. В промежуточном файле не допускаются команды BASIC , программные строки, ни операторы прямого режима.
Для создания промежуточного файла можно использовать редактор текста EDIT. При этом промежуточный файл должен иметь тип .COM и включать соответственные ответы на системные вопросы. Например:

```
N
N
N
Y
Y
Y
Y
Y
N
N
N
N
N
N
N
N
N
Y
N
N
^^
*EX^^
```

Для запуска интерпретатора из созданного таким образом промежуточного файла необходимо напечатать символ @ (знак эт), за которым должна следовать спецификация этого файла.На терминале распечатается весь начальный диалог.Например:

```
.@MINRUN

.R BASIC
BASIC-60/RT-60 V02-035
OPTIONAL FUNCTIONS (ALL, NONE, OR INDIVIDUAL)?
SYS? N
RCTRLO? N
ABORT? N
TTYSET? N
CTRLC & RCTRLC? N
TAB? N
```

```
RND? Y
ABS? Y
SGN? Y
BIN? Y
OCT? Y
LEN? N
ASC? N
CHR$? N
POS? N
SEG$? N
VAL? N
TRM$? N
STR$? N
PI? N
INT? Y
DAT$? N
CLK$? N

READY
```

Использование промежуточных файлов описано в СПРАВОЧНИКЕ ПОЛЬЗОВАТЕЛЯ ОПЕРАЦИОННОЙ СИСТЕМЫ RT-60.

I.3. Останов программ BASIC (команда CTRL/C)

Для прекращения выполнения программы BASIC используется командный ключ CTRL/C . После введения одинарной команды CTRL/C BASIC продолжает выполнение программы до тех пор, пока не встретит оператора ввода данных. Команда <CTRL/C >, набранная дважды, останавливает выполнение программы немедленно. Интерпретатор печатает сообщение:

```
STOP AT LINE xxxxx

READY
```

где:

XXXXX - представляет номер строки, выполняемой в момент введения команды CTRL/C.

Однако, если не выполнялась никакая программная строка, выдается следующее сообщение:

```
STOP

READY
```

Команда CTRL/C отражается на терминале эхом ^C.

Например:

```
10 GO TO 10
RUNNH
^C
^C

STOP AT LINE 10

READY
```

ПРИМЕЧАНИЕ

CTRL/C не возвращает управления в Монитор.
Для этого служит команда **BASIC BYE** (п.I.4.).

I.4. Завершение работы интерпретатора BASIC (команда BYE).

Команда **BYE** предназначена для завершения работы интерпретатора **BASIC**. По команде **BYE** управление возвращается в Монитор, который печатает свой идентификатор (точку). Например:

```
BYE
```

После введения команды **BYE** команда Монитора **REENTER** не обеспечивает возврата в **BASIC**. Вместо её, повторный запуск интерпретатора необходимо осуществить в соответствии с указаниями п.I.2. Если программа **BASIC** придумана для многократного использования, перед введением команды **BYE** необходимо сохранить эту программу.

Если **BASIC** выполняется как оперативная задача, после введения **BYE** необходимо удалить её из памяти, используя команду Монитора:

```
UNLOAD FG
```

I.5.Точность чисел с плавающей запятой

В системе **RT-60 BASIC** может обладать арифметикой с обычной (одинарной) точностью или с двойной точностью.Арифметика с обычной точностью обеспечивает запись чисел с плавающей запятой,в которых количество значащих цифр не превышает семи. Таким образом,**BASIC** с обычной точностью хранит числа I.000001 и I.000000 по разному,но числа I.0000001 и I.0000000 (восемь значащих цифр) запоминает в одинаковом виде.Арифметика с двойной точностью позволяет специфицировать числа с плавающей запятой порядка пятнадцати значащих цифр.

Если требуется повышенная точность,необходимо использовать **BASIC** с арифметикой,точность которой удвоенная.Однако,**BASIC** с двойной точностью имеет два ограничения:

I.Он оставляет программе пользователя меньшую область,поскольку **BASIC** собственно требует большего объёма памяти и все константы,переменные и массивы типа с плавающей запятой требуют дважды больше памяти,чем требовалось бы этими величинами с обычной точностью.

2.Арифметические операции и функции с двойной точностью выполняются значительно медленнее,чем с обычной точностью.

Оператор **PRINT** ,независимо от точности интерпретатора,всегда печатает шесть цифр.Следовательно,для распечатки чисел состоящих более чем из шести цифр,необходимо использовать оператор **PRINT USING** или функцию **STR$**.

Следующий пример иллюстрирует программу,использующую арифметику с удвоенной точностью:

```
LISTNH
10 X=4.237194237
20 Y=6.9090909
30 PRINT X*Y
40 PRINT USING "##.########",X*Y
50 PRINT STR$(X*Y)

READY
RUNNH

 29.2752
29.2751601
29.275160144389

READY
```

Для интерпретатора с двойной точностью типом компилированной программы будет .BAX,в то время как для интерпретатора с обычной точностью этим типом будет .BAC(по умолчанию). Различие между этими типами не допускает к чтению интерпретатором с удвоенной точностью программ,компилированных с обычной точностью и наоборот.При попытке использования интерпретатора **BASIC** с удвоенной точностью и типа файла программы,компилированной с обычной точностью,или наоборот,получаются непредвиденные результаты.

I.6. Сообщения об ошибках,зависящих от системы

Некоторые из сообщений,перечисленные в ОПИСАНИИ ЯЗЫКА **BASIC** либо имеют специальное значение в **BASIC**,либо они им вообще не выдаются.Эти сообщения следующие:

?CANNOT DELETE FILE (?CDF)

Это сообщение не выдается интерпретатором **BASIC.**

?ERROR CLOSING CHANNEL (?ECC)

Это сообщение не выдается интерпретатором.Если ошибка появится при попытке интерпретатора закрыть канал, **BASIC** выведет сообщение

?CHANNEL I/O ERROR (?CIE)

?FILE ALREADY EXISTS (?FAE)

Это сообщение не выдается интерпретатором **BASIC.**

?FILE PRIVILEGE VIOLATION (?FPV)

Это сообщение не выдается интерпретатором **BASIC.**

?FILE TOO SHORT (?FTS)

Файл слишком мал для вывода. Если эта ошибка возникнет при выводе файла данных, необходимо увеличить значение аргумента **FILESIZE** . Если ошибка появится в файле с программой, следует удалить неиспользованные файлы и возобновить операцию.

?ILLEGAL DEF (?IDF)

Это сообщение не выдается интерпретатором **BASIC.**

?ILLEGAL FILE LENGTH

Значение аргумента **FILESIZE** меньше, чем-1(п.2.2)

?ILLEGAL RECORD SIZE (?IRS)

Это сообщение не выдается интерпретатором **BASIC.**

?NOT A VALID DEVICE (?NVD)

Это сообщение не выдается интерпретатором **BASIC.**

?NOT ENOUGH (ROOM)**(?NER)**

Недостаток места для специфицированного аргумента **FILESIZE.**

Командой **UNSAVE** удалить ненужные файлы.

2. ФАЙЛЫ

2.1. Спецификация файла

В BASIC используется стандартные спецификации файлов RT-60. Формат спецификации следующий:

[device:] [filename] [.type]

где:

device - имя устройства, им может быть одно из имен, перечисленных в таблице 2.1., или имя, присвоенное пользователем (см. СПРАВОЧНИК ПОЛЬЗОВАТЕЛЯ ОПЕРАЦИОННОЙ СИСТЕМЫ RT-60).

filename - имя файла, состоящее из 1-6 знаков;

type - тип файла, состоящий из 0-3 знаков.

Имена устройств RT-60 Таблица 2.1.

Имя(код)	Устройство
CTn	Кассета РК-1
DXn:	Память на гибких дисках типа SP-60M.
LP:	Печатающее устройство.
PC:	Станция бумажной ленты SPTP-3 (высокоскоростные считыватель и перфоратор).
TT:	Системный терминал (печатающее устройство и клавиатура).
SY:	Системное устройство (устройство или носитель, из которого загружена операционная система).
DK:	Носитель по умолчанию.
RKn:	Память на твердых дисках.
MTn:	Память на магнитной ленте.

Если пропущен какой-нибудь элемент спецификации, принимается его значение по умолчанию.

Устройством по умолчанию (стандартным устройством) считается DK:. Стандартные спецификации имена файла и его типа обуславливаются оператором или командой, в которой определена спецификация файла. Имена файлов по умолчанию приведены в таблице 2.2., а типы файлов по умолчанию - в таблице 2.3.

Стандартные имена файлов Таблица 2.2.

Оператор или команда	По умолчанию
SAVE, REPLACE, COMPILE	имя текущей программы;
OLD, APPEND, CHAIN, OVERLAY	имя файла NONAME;
UNSAVE, OPEN, KILL, NAME	не выдается, но печатается следующее сообщение об ошибке: ?ILLEGAL FILE SPECIFICATION (?IFS)

Стандартные типы файлов Таблица 2.3.

оператор или команда	BASIC обычной точности	BASIC двойной точности
OPEN, KILL, NAME	.DAT	.DAT
SAVE, REPLACE, UNSAVE, APPEND	.BAS	.BAS
COMPILE	.BAC	.BAX
RUN, OLD	.BAC (если .BAC не найдено, принимается .BAS)	.BAX (если .BAX не найдено, принимается .BAS)

Если создается файл, спецификация которого дублирует имя и тип существующего файла, старый файл будет удален (вытеснен) после закрытия нового файла. Это не может привести к неумышленному удалению, если выдана команда SAVE для сохранения нового файла, так как при попытке вытеснения файла BASIC печатает следующее предупреждение:

?USE REPLACE (?RPL)

которое заставляет пользователя либо согласиться на уничтожение существующего файла, либо сохранить файл, присваивая ему новое имя.

2.2. Оператор OPEN - зависящие от системы элементы

Оператор OPEN имеет следующий формат:

```
OPEN string [FOR INPUT ]                    [#]
            [FOR OUTPUT]  AS FILE [#] expr1 [DOUBLE BUF]
[,RECORDSIZE expr2] [,MODE expr3] [,FILESIZE expr4]
```

где: **string** - спецификация файла (п.2.I.);

expr1 - номер канала, связанный с файлом; номер может принимать значение от I до I2;

DOUBLE BUF - вызывает использование файла с двумя буферами; при этом увеличивается быстродействие некоторых операций с файлом, но требуется дополнительная память для второго буфера;

RECORDSIZE expr2 - игнорируется

MODE expr3 - игнорируется

FILESIZE expr4 - если положительное, специфицирует максимальное количество блоков (256 слов каждый), занимаемых файлом. Если аргумент FILESIZE про-

пущен или значение выражения **expr4** равно нулю, производится стандартная резервация памяти для файла (т.е. либо половина найбольшей свооодной области, либо вся другая по величине область, в зависимости от того, которая из этих областей больше). Если значение **expr4** равно -I, требуется абсолютно найбольшая область свободной памяти. Если это значение меньше, чем -I, выдается следующее сообщение об ошибке:

?ILLEGAL FILE LENGTH

Описанные выше элементы оператора **OPEN** - это аргументы, зависящие от системы. Остальные аргументы оператора **OPEN** приведены в **ОПИСАНИИ ЯЗЫКА BASIC**.

2.3. Распечатка каталога

Для распечатки каталога пользователь должен передать управление в Монитор. Если необходимо сохранить программу **BASIC** используя команду **SAVE** , а затем ввести команду BYE для возвращения управления в Монитор. Например:

```
SAVE TEMP

READY
BYE

.
```

В ответ на точку Монитора ввести команду **DIRECTORY**. Упрощенный формат этой команды (в **СПРАВОЧНИКЕ ПОЛЬЗОВАТЕЛЯ ОПЕРАЦИОННОЙ СИСТЕМЫ RT-6o** представлено укомплектованное описание формата) следующий:

DIRECTORY [/PRINTER] file specification

где:

/PRINTER -ключ,направляющий распечатку на печатающее устройство;если он пропущен,каталог распечатается на терминале:

file specification -специфицирует файлы,которые будут распечатаны; если не специфицированы никакие файлы,распечатке подлежат все файлы.

Элемент'белое поле' в команде **DIRECTORY** позволяет специфицировать файлы с одинаковыми именами или типами или с одними и другими.Если вместо имени указана звездочка,но чётко объявлен тип файла,будут распечатаны все файлы с этим типом. Например,следующая команда выводит на печатающее устройство все исходные программы **BASIC**:

```
.DIRECTORY/PRINTER *.BAS
```

Аналогично,если в поле типа файла помещена звездочка,но указано имя файла,будут печататься все имена файлов,независимо от их типов.Например ,следующей командой распечатаются все файлы с именем **TEST**:

```
.DIRECTORY/PRINTER TEST.*
```

Если в имени или типе файла вместо любого знака стоит символ процента (%),будут распечатаны все файлы,спецификации которых будут согласованы по остальным знакам(т.е.без учёта белого поля);например,спецификации **TEST%% .BAS** будут соответствовать файлы **TESTAB.BAS,TEST01.BAS** и **TESTER .BAS.**

Для распечатки всех компилированных программ **BASIC** можно ввести следующую команду:

```
.DIRECTORY *.BA%
```

Следует отметить,что этой командой будут распечатаны файлы

типа .BAS, .BAC,.BAX,.BAT, .BAK.
Поскольку не специфицирован ключ /PRINTER , распечатка направлена на терминал.
После распечатки каталога пользователь может вернуть управление в BASIC (командой BASIC) и восстановить программу по команде OLD; дополнительно можно удалить промежуточный файл.
Например:

```
.BASIC
BASIC-60/RT-60 V02-03$
OPTIONAL FUNCTIONS (ALL, NONE, OR INDIVIDUAL)? A

READY
OLD TEMP

READY
UNSAVE TEMP

READY
```

3. УСЛОВНЫЕ ФУНКЦИИ

3.I. Условные функции BASIC

Условные функции BASIC позволяют пользователю:

I. Установить ширину распечатки на терминале (TTYSET);

2. Отменить действие CTRL/O (RCTRLO);

3. Заблокировать CTRL/C (CTRLC и RCTRLC);

4. Завершить программу (ABORT);

5. Ввести одиночный знак с терминала (SYS);

6. Завершить работу интерпретатора (SYS);

7. Проверить, введен ли CTRL/C (SYS);

8. Разрешить использование строчных букв (SYS).

В последующих пунктах услуговые функции BASIC рассматриваются в контексте оператора LET с формальными параметрами, т.е:

[LET] variable =utility function

где: variable -переменная

utility function -одна из функций, описанных в последующих пунктах

Как правило, услуговые функции могут появиться в любом арифметическом выражении. Предпочитается формат оператора LET, поскольку он является простейшим оператором и, следовательно, повышает удобство чтения программы.

3.2. Установление формата распечатки на терминале (функция TTYSET).

Функция TTYSET предусмотрена для установки правого поля распечатки. BASIC печатает программную строку так долго, пока число или знаковая последовательность не придет к концу строки (т.е. к правому полю терминала). Попытка нарушить установленный формат строки вызывает продолжение распечатки текущего элемен-

та в следующей строке.

Формат функции TTYSET следующий:

[LET] variable = TTYSET(255%, expression)

где:

variable - представляет собой переменную и после выполнения оператора принимает неопределенное значение;

255% - либо числовая константа (как это указано в формате), либо целое число 255 (по аналогии к другим версиям интерпретатора BASIC);

expression - определяет правое поле терминала; это поле определяется значением выражения минус I; если значение выражения равно нулю, не выполняются никакие изменения.

Например, для установки максимальной физически допустимой ширины печатающего устройства терминала можно ввести:

```
A = TTYSET(255%, 133%)
```

Для установки максимальной физически допустимой ширины строки дисплея терминала можно ввести:

```
A = TTYSET(255%, 81%)
```

Если не специфицируется новая ширина выводимой строки, по умолчанию BASIC принимает 72-знаковый формат.

Необходимо обратить внимание, чтобы ширина строки, определенная функцией TTYSET , не превышала фактического формата печатающего устройства терминала.

Если значение выражения expression меньше 0, равно I или более 256, BASIC выдает сообщение об ошибке:

```
?ARGUMENT ERROR  (?ARG)
```

Это сообщение выдается также в случае, если первым аргументом не является число 255.

3.3. Отмена действия CTRL/O (функция RCTRLO).

После введения команды CTRL/O BASIC приостановливает вывод на терминал;для возобновления вывода можно использовать функцию RCTRLO .Эта функция обеспечает распечатку на терминале некоторых данных помимо действия команды CTRL/O

Функция RCTRLO имеет следующий формат:

[LET] variable = RCTRLO

где: variable -представляет собой переменную,которая после выполнения оператора принимает неопределённое значение

Рассмотрим следующий пример:

```
LISTNH
10 REM PROGRAM TO INPUT DATA
20 REM FROM FILE AND PRINT SUM
30 OPEN "NUMBER" FOR INPUT AS FILE #1
40 PRINT "DATA IN FILE:"
50 IF END #1 THEN 100
60 INPUT #1,D
70 PRINT D
80 T=T+D
90 GO TO 50
100 A=RCTRLO
110 PRINT
120 PRINT "SUM=";T

READY
RUNNH

DATA IN FILE:
 4
 16
 147
 26
^O

SUM= 4172

READY
```

При выполнении цикла (от строки 50 до 90) BASIC печатает числа. После введения CTRL/O распечатка приостанавливается. Однако после выполнения строки 100 (она отменяет CTRL/O), BASIC возобновляет вывод чисел.

3.4. Блокировка CTRL/C (функции RCTRLC и CTRLC).

Для некоторой части программы пользователь может потребовать блокировки ее прерывания, вызванного вводом с клавиатуры команды CTRL/C . Для этого предусмотрена функция RCTRLC, которая блокирует действие CTRL/C и препиатствует остановлению программы. Обратное действие возможно после введения функции CTRLC.

Форматы функций RCTRLC и CTRLC следующие:

```
[LET] variable = RCTRLC
[LET] variable = CTRLC
```

где: variable - переменная, которая после выполнения оператора принимает неопределенное значение

Если BASIC выполнит функцию RCTRLC , ключ CTRL/C не вызывает приостановления программы.

Если выполнена функция CTRLC , команда CTRL/C немедленно прекратит программу. В период действия RCTRLC все возможные команды CTRL/C теряются.

После распечатки сообщения READY BASIC автоматически разрешает ввод команды CTRL/C.

Например:

```
LISTNH
1000 REM DO NOT ALLOW INTERUPTS
1010 A=RCTRLC
1020 PRINT "NO INTERUPTS"
```

```
1030 FOR I=1 TO 1000 \ S=S+1 \ NEXT I
1100 REM NOW ALLOW INTERUPTS
1110 A= CTRLC
1120 PRINT "INTERUPTS OKAY"
1130 FOR I=1 TO 1000 \ S=S+1 \ NEXT I
32767 END

READY
RUNNH

NO INTERUPTS
^C
INTERUPTS OKAY
^C



STOP AT LINE 1130

READY
```

Дополнительные информации о команде CTRL/C , блокированной системной функцией, содержатся в п.3.6.3.

ПРИМЕЧАНИЕ

Если CTRL/C заблокировано, попытки остановить BASIC будут неудачными. Не рекомендуется блокировать CTRL/C во время отладки программы.

3.5. Завершение выполнения программы (функция ABORT).

Для того, чтобы после прекращения выполнения программы удалить ее из памяти, следует использовать функцию ABORT. Функция ABORT аналогична оператору END , но дополнительно эта функция может удалить программу из памяти и присвоить программе новое имя NONAME (эквивалентно команде SCR). Формат функции ABORT следующий:

[LET] variable = ABORT (expression)

где:

variable - переменная, которая после выполнения оператора принимает неопределенное значение;

expression – определяет, будет ли программа удаляться из памяти или нет. Если значение выражения равно 0, BASIC не удаляет программы; если это значение равно 1, программа удаляется.

Рассмотрим следующую программу:

Программа удаляется из памяти

```
LIST

ABORT       14-DEC-82   00:10:05

10 PRINT "123"
20 A=ABORT(1)
30 PRINT "456"

READY
RUNNH

123

READY
LIST

NONAME      14-DEC-82   00:10:05

READY
```

Программа не удаляется из памяти

```
LIST

ABORT       14-DEC-82   00:10:05

10 PRINT "123"
20 A=ABORT(0)
30 PRINT "456"

READY
RUNNH

123

READY
LIST

ABORT       14-DEC-82   00:10:06

10 PRINT "123"
20 A=ABORT(0)
30 PRINT "456"

READY
```

3.6. Системные функции (функции SYS).

Функции SYS выполняют операции, связанные с системой.

Формат функций SYS следующий:

[LET] variable = SYS(expression 1, expression 2)

где: variable – переменная

expression 1 – определяет функцию, которая должна выполниться

expression 2 – необязательный аргумент, используемый в некоторых функциях

В таблице 3.I. перечислены функции SYS , действие которых определяется значением выражения expression 1 . Если специфицируется значение, отличное от указанных в табл.3.I., BASIC выдает сообщение об ошибке:

?ARGUMENT ERROR (?ARG)

Системные функции Таблица 3.I

Значение expression 1	Выполняемые операции
I	Ввод одиночного знака; переменная содержит значение ASCII следующего знака, который вводится с терминала.
4	Заканчивает работу интерпретатора BASIC и возвращает управление в системный Монитор (аналогично команде BYE).
6	Позволяет проверить, набрана ли команда CTRL/C с момента введения RCTRLC ; CTRL/C набрана, если значение переменной равно 0; если это значение равно I, CTRL/C не набрана.

7	Разрешает или запрещает вводить строчные буквы с терминала пользователя. Если значение выражения expression 2 равно 0, ввод строчных букв разрешается. Если это значение равно 2, строчные буквы будут преобразовываться в их эквиваленты из набора прописных букв.

3.6.I. Ввод одиночного знака

Функция SYS (I) используется для ввода одиночного знака с терминала.

SYS (I) выдает 7-битовое значение ASCII любого знака (за исключением CTRL/O), введенного с терминала. (Набор значений ASCII представлен в ОПИСАНИИ ЯЗЫКА BASIC). Если при выполнении SYS (I) вводится CTRL/C и он не запрещается никакой функцией, BASIC печатает STOP и сообщение READY ; если CTRL/C запрещено, BASIC продолжает выполнение функции SYS (I) и ожидает поступления следующего знака. BASIC не принимает этого знака, если не нажата клавиша <CR>.

Например:

```
LISTNH
10 PRINT "TYPE A CHARACTER: ";
20 A=SYS(1)
40 PRINT "THE ASCII VALUE OF ";CHR$(A);" IS";A

READY
RUNNH

TYPE A CHARACTER: Z
THE ASCII VALUE OF Z IS 90

READY

READY
```

3.6.2. Завершение работы интерпретатора BASIC

При помощи функции SYS(4) можно завершить работу интерпретатора из программы BASIC . Эта операция производится аналогично команде BYE.

Например:

```
LISTNH
10 PRINT "GOODBYE"
20 A=SYS(4)

READY
RUNNH

GOODBYE
```

3.6.3. Проверка команды CTRL/C.

Если команда CTRL/C запрещена функцией RCTRLC и необходимо проверить, введена ли эта команда, следует использовать функцию SYS(6). Эта функция выдает значение 1, если команда CTRL/C набрана, и 0, если CTRL/C не набрана.

Например:

```
LISTNH
10 A=RCTRLC : REM DISABLE CTRL/C.
30 B=SYS(6) : REM CHECK FOR CTRL/C.
40 IF B=1 THEN 100
50 PRINT "STILL EXECUTING"
60 GO TO 30
100 PRINT "PROGRAM TERMINATING"
110 A=CTRLC : REM REENABLE CTRL/C
120 A=ABORT(1)

READY
RUNNH

STILL EXECUTING
STILL EXECUTING
¬C¬C
STILL EXECUTING
PROGRAM TERMINATING

READY
```

3.6.4. Использование строчных букв

Функция SYS (7, expr 2) управляет вводом строчных букв с терминала. Система RT-60 , как правило, преобразует все строчные буквы алфавита в их эквиваленты из набора прописных букв. Выполнив функцию SYS (7,0), пользователь вызывает, что строчные буквы будут передаваться без преобразования. Для отмены этого режима ввода используется функция SYS (7,I).
После выхода из интерпретатора BASIC режим ввода строчных букв обуславливается последней функцией SYS (7, expr 2).
В следующем примере демонстрируется разрешение и завершение ввода строчных букв. Сначала разрешается ввод строчных букв, так как выполняется функция SYS(7%, 0%). После этого программа модифицируется, позволяя пользователю вводить ответы, включающие строчные буквы. В конце, программа выполняется в измененной форме, запрещающей запись строчных букв. Затем преобразованная таким образом программа сохраняется.

```
LISTNH
10 REM PROGRAM TO CHANGE LOWER CASE CONVERSION
20 PRINT "DO YOU WANT TO ENTER LOWER CASE CHARACTERS (Y OR N)";
30 INPUT A$
40 IF A$="Y" THEN 100
50 IF A$<>"N" THEN 20
60 A=SYS(7%,1%) ! REM DISABLE LOWER CASE
70 GO TO 32767
100 A=SYS(7%,0%) ! REM ENABLE LOWER CASE
32767 END

READY
RUNNH

DO YOU WANT TO ENTER LOWER CASE CHARACTERS (Y OR N)? Y
```

```
READY
45 if a$ ="y" then 100 ! rem Check for lower case y
sub 50 @20@if a$()"n" then 20 ! Rem Check for lower case n
50 IF A$()"N" THEN if a$()"n" then 20 ! Rem Check for lower case n

READY
listnh
10 REM PROGRAM TO CHANGE LOWER CASE CONVERSION
20 PRINT "DO YOU WANT TO ENTER LOWER CASE CHARACTER (Y OR N)";
30 INPUT A$
40 IF A$="Y" THEN 100
45 if a$ ="y" THEN 100 ! REM Check for lower case y
50 IF A$()"N" THEN IF A$()"n" THEN 20 ! REM Check for lower case n
60 A=SYS(7%,1%) ! REM DISABLE LOWER CASE
70 GO TO 32767
100 A=SYS(7%,0%) ! REM ENABLE LOWER CASE
32767 END

READY
runnh

DO YOU WANT TO ENTER LOWER CASE CHARACTERS (Y OR N)? n

READY
SAVE LOWCHM

READY
```

Если пользователь вводит строчные буквы, которые запрещены, эхо на терминале будет повторятья в виде прописных букв. Обратим внимание, что **BASIC** преобразует ключевые слова в виде строчных букв в форму со строчными буквами, однако текстовые константы и знаковые последовательности останутся неизменными.

4. ОБЪЕДИНЕНИЕ ИНТЕРПРЕТАТОРА BASIC С ПОДПРОГРАММАМИ НА ЯЗЫКЕ МАКРОАССЕМБЛЕРА

4.I. Описание подпрограмм на языке макроассемблера

BASIC позволяет включать программы, написанные на языке макроассемблера (ALR) для расширения возможностей интерпретатора. Например, могут включаться подпрограммы связи со специальными устройствами /например, с лабораторным оборудованием/ или процедуры обработки массивов. После объединения интерпретатора с подпрограммами они могут выполняться в прямом режиме или в программе BASIC , в зависимости от значения оператора CALL (см. ОПИСАНИЕ ЯЗЫКА BASIC). Преимущества ALR , включенных в BASIC , по отношению к чистым программам на языке макроассемблера, следующие:

- только программист, составляющий ALR должен обладать знанием о макроассемблере: это не требуется от пользователя, которому должно быть известно программирование на языке BASIC;
- по отношению к программам на макроассемблере программы на языке BASIC составляются, выполняются, актуализируются значительно проще:
- ALR можно выполнить в прямом режиме интерпретатора без записи всей программы;

ПРИМЕЧАНИЕ

В настоящем справочнике предполагается, что пользователь ознакомлён с операционной системой и услуговыми системными программами (редактором, ассемблером MACRO, компоновщиком и т.д.).

В дальнейшем описывается:

-формат AIR,

-процедуры доступа к аргументам,

-использование вспомогательных процедур BASIC.

AIR, использующие интерфейс вызовов FORTRAN IV (определенный в СПРАВОЧНИКЕ ПОЛЬЗОВАТЕЛЯ FORTRAN), могут вызываться как из компилятора FORTRAN, так и интерпретатора BASIC. Однако AIR не могут осуществлять доступа к процедурам или глобальным адресам в самом компиляторе FORTRAN.

4.2. Формат подпрограммы на языке макроассемблера

Для составления подпрограммы на языке макроассемблера, которую можно встроить в интерпретатор BASIC необходимо определить её имя и её адрес запуска в таблице имени подпрограммы пользователя (рис.4.I). Указатели всех AIR необходимо поместить за меткой FTBL. Каждым указателем специфицируется ячейка имени AIR и стартовый адрес. Слово, содержащее одни нули, назначает конец списка указателей.

ПРИМЕЧАНИЕ

В имени AIR не должны содержаться пробелы. По аналогии к FORTRAN нельзя специфицировать имени AIR, длина которой превышает 6 знаков, так как BASIC признает лишь одно ограничение, относящееся к длине программной строки.

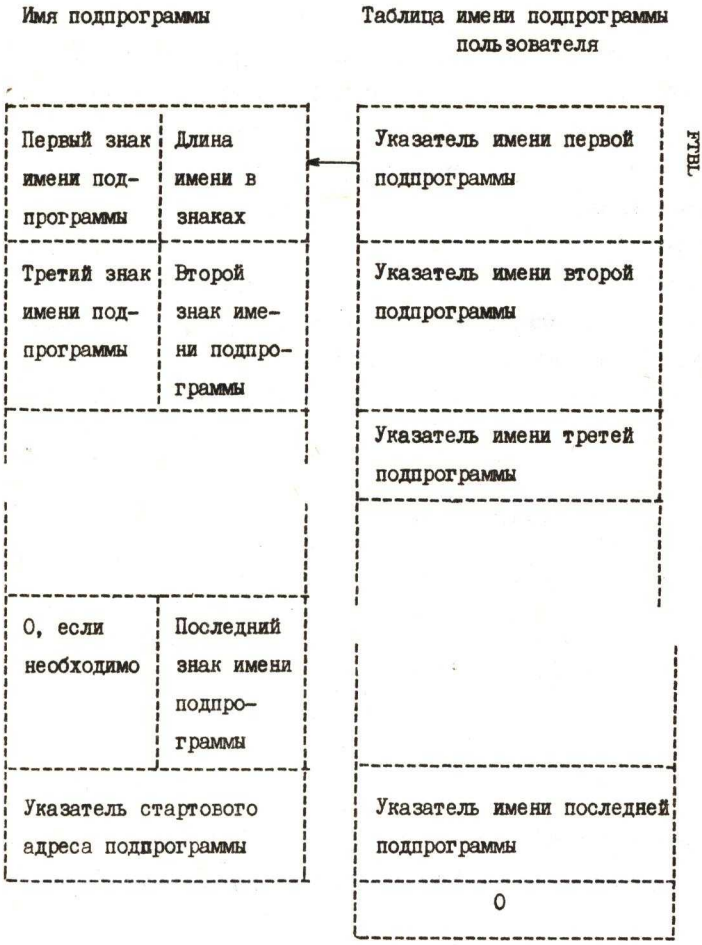


Рис.4.I. Таблица имени подпрограммы пользователя и формат имени подпрограммы

В дистрибутивном программном обеспечении (оно не поставляется пользователям) содержится файл **BSCLI.MAC** ,включающий глобальное имя **FTBL** .Это базовый файл для таблицы указателя.Она стройтся путём добавления входов между глобальное имя **FTBL** и нулевой вход(**.WORD 0**) при помощи программы **EDIT**.Рекомендуется помещать процедуры **AIR** в начале подпрограммы.В этом случае указатели в таблице имени подпрограммы пользователя должны быть глобальными символами.Например,если необходимо включить три подпрограммы **INITIT,ADDER** и**CHKSTA** список имён подпрограммы должен быть следующим:

```
          .
          .
          .
        .GLOBL  FTABI
        .GLOBL  INITNM, ADDNM, CHKSNM
FTABI:  .WORD   FTBL
FTBL:   .WORD   INITNM          ;USER ROUTINE
        .WORD   ADDNM           ;NAME LIST
        .WORD   CHKSNM
        .WORD   0
          .
          .
          .
        .END
```

Метки INITNM,ADDNM и CHKNM не должны назначать начала подпрограмм **INITIT,ADDER** и **CHKSTA** соответственно.Например:

```
        THE INIT ROUTINE
        .GLOBL  INITNM
INITNM: .BYTE   6
        .ASCII  'INITIT'        ;NUMBER OF CHARACTERS IN NAME
        .EVEN
        .WORD   INITST
INITST:                         ;START OF ROUTINE
```

Альтернативным методом является добавление имени подпрограммы и стартового адреса за таблицей имени подпрограммы. В этом случае стартовые адреса подпрограммы должны быть глобальными именами. Используя программу из предыдущего примера, таблица имени подпрограммы должна быть следующей:

```
        .GLOBL  FTABI
        .GLOBL  INITST, ADDST, CHKSST
FTABI:  .WORD   FTBL
FTBL:   .WORD   INITNM
        .WORD   ADDNM
        .WORD   CHKSNM
        .WORD   0
INITNM: .BYTE   6
        .ASCII  'INITIT'            ;NUMBER OF CHARACTERS IN NAME
        .EVEN
        .WORD   INITST
ADDNM:  .BYTE   5
        .ASCII  'ADDER'
        .EVEN
        .WORD   ADDST
CHKSNM: .BYTE   6
        .ASCII  'CHKSTA'
        .EVEN
        .WORD   CHKSST
          .
          .
          .
        .END
```

Каждая ALR должна запускаться со специфицированного стартового адреса.Например:

```
          THE INITIT ROUTINE
          .GLOBL  INITST
INITST:                                 ;START OF ROUTINE
          .
          .
          .
```

Этот альтернативный метод рекомендуется использовать при включении ALR,составленных для FORTRAN,в BASIC .
Во всех последующих примерах применяется рекомендуемый метод (в котором пакет имени подпрограммы назначает старт подпрограммы).
После определения имени и стартового адреса подпрограммы можно записать текст самой подпрограммы. ALR может использовать стек,но не должны нарушать его пределов.Перед передачей управления в ALR BASIC устанавливает в регистре R4 предельное значение стека.Если используются математические операции или подпрограммы-функции BASIC ,необходимо убедиться,что область стека достаточна для выполнения требуемых процедур(I5 свободных слов для процедур с обычной точностью или 30 свободных слов для процедур с двойной точностью). ALR должна заканчиваться инструкцией RTS PC ,а стек должен иметь такой вид ,как перед выполнением ALR .Формат подпрограммы INITIT следующий:

```
        THE INIT ROUTINE
        .GLOBL  INITNM
INITNM: .BYTE   6
        .ASCII  'INITIT'
        .EVEN
        .WORD   INITST
INITST:                         ;START OF ROUTINE
;         .
;         .
;         .
;       MAIN BODY OF ROUTINE
;         .
;         .
;         .
        RTS     PC              ;END OF ROUTINE
```

4.3. Доступ к аргументам - список аргументов

Выполняя оператор CALL , BASIC осуществляет преобразование аргументов и представляет подпрограмме два списка. Первый из них содержит указатели преобразованных аргументов, а второй - описатели типов аргументов. Процедуры ALR должны проверять допустимость чисел в списке и правильность типов аргументов. Проверка должна заключаться в устранении возможных причин неустранимых ошибок как в ALR , так и в интерпретаторе BASIC . Если проверка аргументов не производилась, а оператором CALL вызывается ошибка типа данных, в результате выполнения ALR могут выдастся непредвиденные значения. Например, если ALR оперирует с массивом целого типа и оператор CALL включает текстовое выражение, ALR может разрушить стек. При проверке аргументов ALR подпрограмма может сама себя защитить от ошибок со стороны интерпретатора BASIC . Специальная защита ALR не предусмотрена.

Процедуры на языке FORTRAN , соответствующие ALR , не в состо-

нии проводить проверок аргументов, так как компилятор FORTRAN не предоставляет списка описателей аргументов.

Перед передачей управления в AIR BASIC преобразует аргументы, специфицированные в операторе CALL . BASIC создает список указателей аргументов и список описателей аргументов. На рис.4.2 указан список описателей аргументов, который создается интерпретатором перед передачей управления в AIR.

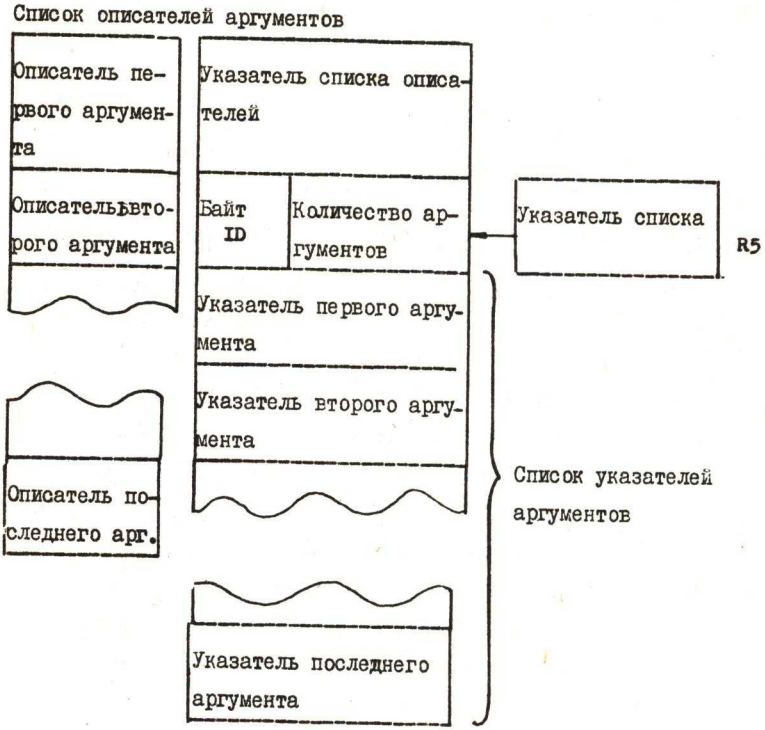


Рис.4.2. Списки аргументов подпрограммы на языке макроассемблера.

Из рис.4.2. следует, что регистр R5 указывает слово, определяющее количество аргументов в операторе CALL и идентифицирующее язык, который вызывает ALR . Список указателей аргументов начинается в следующем слове и список описателей аргументов хранится в предыдущем слове. Каждый байт слова, указанного регистром R5 , имеет определенное значение. В младшем байте хранится количество аргументов; старший байт идентифицирует язык. Если вызывающим языком является BASIC , значением старшего байта будет 202. Если вызов осуществляется из FORTRAN, значением старшего байта будет 0.

Указатели списка указателей аргументов специфицируют адреса аргументов, подлежащих преобразованию. Существуют два исключения: указатели нулевых аргументов и указатели аргументов массивов текстового типа.

Если аргументом является нуль-аргумент, указатель вместо адреса аргумента содержит значение -I. Нуль-аргумент получается в списке аргументов оператора CALL ,если две запятые расположены друг рядом с другом или завершающей командой. Например, оператором CALL "INITIT" (A, B,, D,) выдаются следующие аргументы : A, B, нуль, D и нуль.

Если аргумент представляет собой текстовой массив, указатель не специфицирует адреса этого аргумента, а вместо него содержит значение, обеспечивающее доступ к текстовому массиву (п.4.3.2). Если аргументом является знаковая последовательность без установленных индексов или элемент текстового массива, указатель специфицирует адрес первого знака этой последовательности.

Список описателей аргументов определяет типы данных для каждого аргумента и указывает, когда аргумент является массивом

и когда ALR может возвратить результат в аргумент. В BASIC включена дополнительная информация, описывающая знаковые последовательности и массивы. Если она требуется, слово в списке описателей аргументов указывает слово описания, в котором содержится дополнительная информация. Формат слова описания изображен на рис.4.3.

О том, является это слово указателем или описателем, решает значение нулевого разряда этого слова. Если этот бит очищен (т.е. равен 0), слово в списке представляет собой указатель. Если этот бит установлен (т.е. равен 1) слово в списке представляет собой описатель. Для знаковых последовательностей и массивов нулевой разряд этого слова очищенный.

ПРИМЕЧАНИЕ

Все числа в этом пункте специфицируют содержимое слова или группы слов в восьмеричном коде.

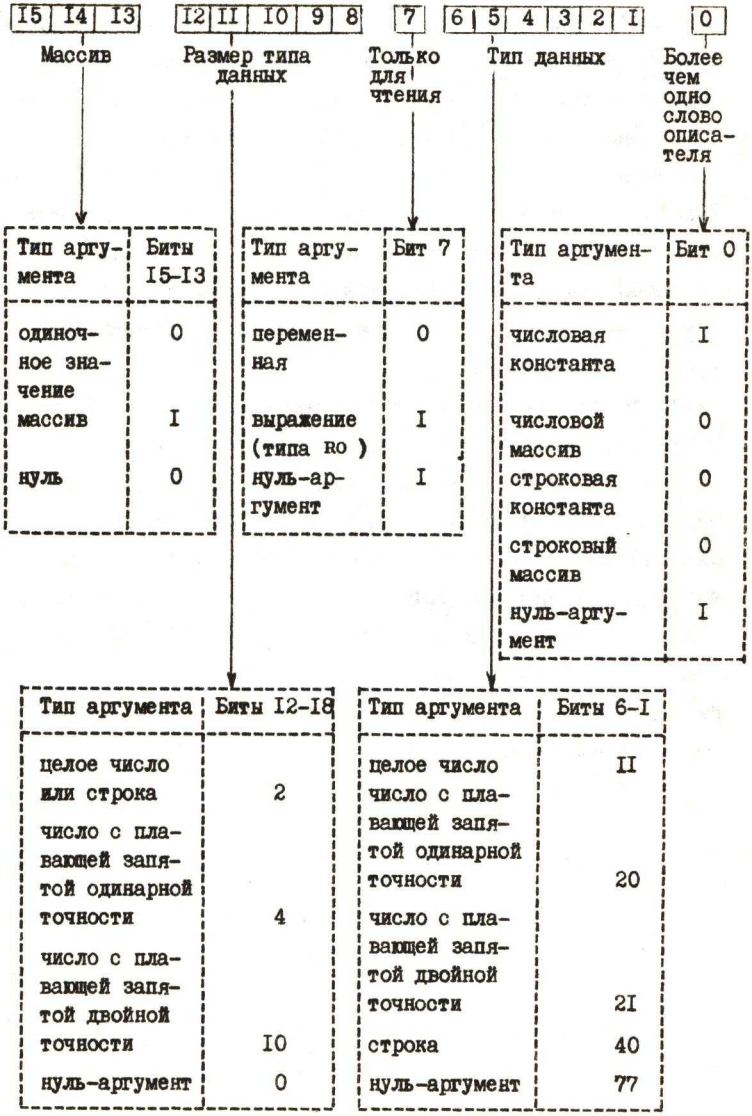


Рис.4.3. Формат слова описания аргумента

ALR могут возвращать аргументы только в переменные и массивы. Если аргумент представляет собой выражение, константу или элемент виртуального массива, бит 7 слова описания аргумента установленный и ALR должен обязательно возвращать значение в этот аргумент.

Биты I2-8 в слове описания аргумента специфицируют размер типа данных. ALR не нуждается в проверке этой информации, поскольку каждый тип аргумента (определенный разрядами 6-I) имеет фиксированный размер. Содержимое битов I2-8 для текстовых аргументов игнорируется.

BASIC выдает дополнительную информацию для аргументов массива и текстовых аргументов, специфицируя общее количество байтов в массиве. Количество индексов массива, верхний предел первого индекса и верхний предел второго индекса (для двухмерного массива). BASIC может также предоставлять указатели текстовых обращений для текстовых аргументов. Процедуры интерпретатора BASIC используют эти указатели для осуществления доступа к текстовым аргументам (п.4.3.2).

На рис. 4.4 представлен формат описателей массива и знаковой последовательности.

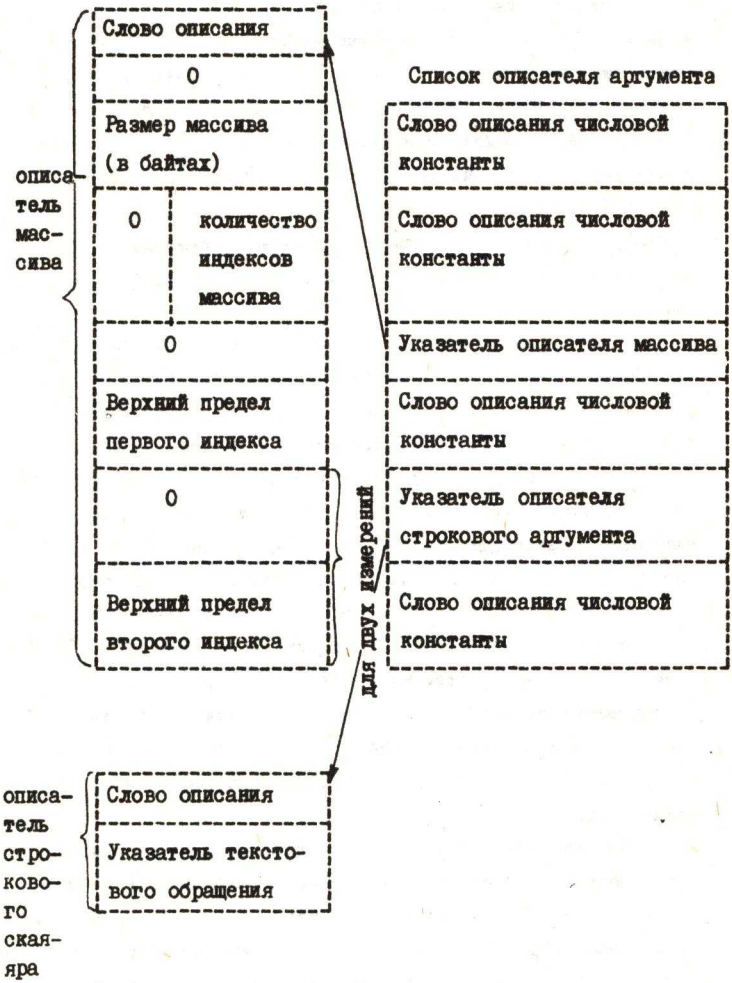


Рис.4.4. Формат описателей аргумента массива и текстового аргумента

4.3.1. Массивы числового типа

Если оператор CALL специфирует элемент массива числового типа, например, A(10), BASIC рассматривает его как одномерный массив, начинающийся с указанного элемента и заканчивающийся последним элементом массива. BASIC интерпретирует этот массив как одномерный, даже если целый массив является двухмерным.

BASIC и FORTRAN хранят массивы по разному. Измерения массива в BASIC начинаются с нуля, а в FORTRAN - с единицы.

Второе измерение массива в BASIC измеряется быстрее, чем первое; в FORTRAN - наоборот. Это разногласие следует учитывать при вызовах ALR из FORTRAN или BASIC.

4.3.2. Знаковые последовательности и текстовые массивы

В этом пункте описываются процедуры BASIC, обеспечивающие подпрограммам на языке макроассемблера доступ к буквенно-цифровым последовательностям. Кроме того, приводятся некоторые примерные процедуры, осуществляющие доступ к знаковым последовательностям. BASIC поддерживает динамические знаковые последовательности, длины которых могут изменяться во время выполнения программы. Процедуры доступа к знаковым последовательностям контролируют местоположение и размер текстов. Следовательно, ALR не в состоянии изменять текст без использования процедур доступа к знаковым последовательностям.

Процедуры доступа к знаковым последовательностям и доступа к текстовым массивам различные. Следует обратить внимание, что если оператор CALL специфицирует элемент текстового массива (например, A$(10)), BASIC интерпретирует его как числовую константу. Если передается целый массив (напри-

мер, A$ ()), он рассматривается как текстовой массив. ALR должна помещать и восстанавливать слово указателя текстового обращения и передавать это слово в процедуру доступа к знаковой последовательности. В случае текстового аргумента указатель текстового обращения представляет собой слово, следующее за словом описания. Для аргумента текстового массива ALR обязана вычислять указатель текстового обращения для осуществления доступа к любому элементу массива.

Указатель текстового обращения - это слово, значение которого определяется по следующей формуле:

указатель текстового обращения = 2 * offset + argument pointer

где:

offset - позиция элемента в массиве

argument pointer - значение для текстового массива в списке указателей аргумента; следует отметить, что указатель аргумента для текстового массива не специфицирует адреса самого аргумента.

Позиция элемента одномерного массива определяется значением его индекса. Позиция элемента двухмерного массива вычисляется по следующей формуле:

позиция = индекс I (максимальное значение индекса 2 + I) + + индекс 2

Например, рассмотрим два массива A$ (I0) и B$ (3,5) с указателями аргументов A и B, соответственно. (ПРИМЕЧАНИЕ: Все числа в следующем списке представляют десятичные значения).

элемент	2 * offset + argument pointer	указатель текстового обращения
A$(0)	2*0 +A	A
A$(4)	2*4 +A	8+A
B$(0,5)	2*(0*6+5)+B	10+B
B$(1,5)	2*(1*6+5)+B	22+B
B$(2,0)	2*(2*6+0)+B	24+B

Процедуры доступа к последовательности используют указатель текстового обращения, который выдается подпрограммой ALR для нахождения и обработки последовательностей.

Процедуры доступа к последовательностям следующие:

$FIND

$ALC

$STORE

$DEALC

Процедурой $FIND выдается длина последовательности и положение первого знака. Процедурой $ALC создается временная последовательность, в которую подпрограмма ALR может только записывать знаки. Процедура $STORE присваивает значение одной последовательности другой последовательности и заменяет первую последовательность пустым текстом (нулевой последовательностью). Процедура $DEALC возвращает в стек область, занимаемую временной последовательностью. Подпрограмма ALR должна использовать следующую общую процедуру для редактирования текстового аргумента и последующего возвращения результирующей последовательности. Сначала ALR выбирает текстовой аргумент, используя процедуру $FIND.

Затем ALR создает временную последовательность при помощи процедуры $ALC. После этого считываются знаки из тексто-

вого аргумента; они редактируются требуемым образом и возвращаются во временную последовательность. Затем ALR использует процедуру $STORE для копирования временной последовательности в текстовой аргумент, которым может быть исходная последовательность. Наконец, ALR удаляет данные из стека, которые были занесены процедурой $ALC ; для этого используется процедура $DEALC.

В таблице 4.I приведены характеристики процедур доступа к последовательностям, т.е. описывается начальная установка, включение формата подпрограммы обращения (JSR), требуемой для вызова процедуры доступа к последовательности. Одновременно дается прогноз ожидаемых результатов и поясняется методика их интерпретации. В частности указывается способ определения, является ли верной начальная установка для процедуры доступа, или нет.

Если ALR вызывает процедуры $ALC, $STORE и $DEALC она должна специфицировать эти имена как глобальные символы. Перед вызовом одной из этих процедур необходимо убедиться, что в регистре R5 установлено начальное значение, т.е. значение при котором BASIC передает управление в ALR. Таким образом, регистр R5 должен указывать слово, специфицирующее количество аргументов.

ПРИМЕЧАНИЕ

Описанными процедурами требуется, чтобы регистр указывал одно и то же значение, которое специфицируется битами 6-I в качестве слова описания аргумента. Это можно обеспечить, записывая значение I00 в указанный регистр (т.е. значение 40 в поле, назначенное разрядами 6-I) или пересылая слово

описания аргумента в определенный регистр.

4.4. Использование процедур, предоставляемых интерпретатором BASIC

BASIC предоставляет процедуры, которые обслуживают ошибочные состояния, печатают на терминале сообщения и выполняют математические операции и функции.

4.4.I. Процедуры обслуживания ошибок и сообщений.

BASIC включает две процедуры обслуживания ошибок ($ARGER и $BOMB)и две процедуры вывода сообщений ($MSG и $CHROT). Процедурой $ARGER печатаются сообщения о неустранимых ошибках: ?ARGUMENT ERROR (?ARG).Подпрограмма ALR должна вызывать процедуру $ARGER если обнаружилась ошибка аргумента.Процедура $BOMB позволяет подпрограмме ALR специфицировать собственные сообщения о неустранимых ошибках.Процедура $MSG печатает на терминале любое сообщение и возвращает управление в ALR .Процедурой $CHROT печатается на терминале одиночный знак, после чего управление переходит в ALR. Если подпрограммой ALR вызываются $ARGER , $BOMB, $MSG или $CHROT , она должна специфицировать эти имена как глобальные адреса.

Для вызова процедуры $ARGER используется следующая инструкция:

```
JMP $ARGER
```

Процедура $ARGER печатает сообщения об ошибках в следующих формах:

```
?ARGUMENT ERROR AT LINE xxxxx
?ARG AT LINE xxxxx
```

где: xxxxx -номер строки с оператором CALL.

Если оператор CALL используется в прямом режиме, строка

```
?ARG AT LINE xxxxx
```

не печатается и управление возвращается в BASIC , который сигнализирует свою готовность сообщением READY.

Процедура $BOMB вызывается следующей инструкцией:

```
JSR  R1, $BOMB
.ASCIZ  `message`
.EVEN
```

где: message - текст пользователя

Процедура $BOMB печатает на терминале сообщение об ошибке, формат которого следующий:

```
?error message AT LINE xxxxx
```

где: XXXXX - номер строки с оператором CALL

Если CALL используется в качестве оператора прямого режима, текст AT LINE xxxxx не печатается. Управление затем возвращается в BASIC , который печатает сообщение READY.

Процедура $MSG вызывается по следующей инструкции:

```
JSR  R1, $MSG
.ASCII  `message`
.BYTE 15,12,0
.EVEN ;  НЕОБХОДИМО СПЕЦИФИЦИРОВАТЬ <CR> ,<LF> и 0.
```

Использование процедур доступа к последовательностям — Таблица 4.I

Процедура	Программные установки	Результат, если нет ошибки	Результат, если обнаружена ошибка
$FIND (выдает позицию и длину последовательности)	R0 ← указатель текстового обращения, R1 ← I00, R5 ← начальное значение, Вызов: JSR PC,**$FIND**	R0=адрес первого знака последовательности, R1=длина последовательности, R2=I00, R3,R4,R5 не изменяются, бит C=0 (знак), бит Z=I, если последовательность пустая (R1=0)	R0 содержит код ошибки: если R0=I, R1 не равно I00 если R0=2, R5 не содержит правильного начального значения, R3,R4,R5 не изменяются, бит C=I
$ALC (создает временную последовательность)*	R0 ← требуемая длина последовательности, R1 ← I00, R5 ← начальное значение, Вызов: JSR PC,**$ALC**	R0=адрес первого знака последовательности, R1=длина последовательности, R2=I00, R3,R4,R5 не изменяются бит C=0, бит Z=I, если последовательность пустая (R1=0), SP=указатель текстового обращения; стек содержит несколько слов с внутренними указателями; эти слова необходимо удалить из стека с помощью процедуры **$DEALC**.	R0 содержит код ошибки: если R0=0, сигнализирует несущественную свободную память для требуемой последовательности если R0=I, R1 не равно I00 если R0=2, R5 не содержит правильного начального значения R3,R4,R5 не изменяются бит C=I

$DEALC (удаляет из стека внутренние указатели, установленные процедурой **$ALC**)*	Возвращает стек в состаяние в котором он находился перед вызовом процедуры **$ALC** .Для выполнения этого следует удалить из стека те слова,которые были туда занесены вызовом процедуры **$ALC** .Это обеспечивает установку указателя текстового обращения в **SP** , R2←I00, R5←начальное значение, Вызов:**JSR PC, $DEALC**	R0,R1,R2,R3,R4,R5 не изменяются, бит C=0, стек возвращается в состояние, в котором он находился до вызова процедуры **$ALC**.	R0 содержит код ошибки: если R0=1,R2 не равно I00 если R0=2,R5 не содержит правильного начального значения, R1,R2,R3,R4,R5 не изменяются, бит C=1,
$STORE (запоминает значение одной последовательности и другой последовательности;первая послед.приводится к нулю)	R0←указатель текстового обращения последовательности,которая будет копироваться, R1←указатель текстового обращения последовательности,в которую направляется копия, R2←I00, R5←начальное значение, Вызов: **JSR PC, $STORE**	R0,R1,R2,R3,R4,R5 не изменяются, бит C=0 последовательность,указатель которой находится в R1,содержит предыдущее значение другой последовательности. последовательность,указатель которой находится в R0,равняется нулю.	R0 содержит код ошибки: если R0=1,R2 не равно I00, если R0=2,R5 не содержит правильного начального значения, R1,R2,R3,R4,R5 не изменяются, бит C=1

* Перед завершением **ALR** любая временная последовательность,созданная процедурой **$ALC** ,должна быть удалена процедурой **$DEALC**

где: **message** -знаковая последовательность,каторая будет распечатана

Процедура **$MSG** печатает на терминале специфицированное пользователем сообщение и затем передает управление инструкции,следующей за директивой **.EVEN.**

Процедура **$CHROT** вызывается следующим образом:

1.В младший байт регистра **R0** помещается 8-битовый код **ASCII** знака;

2.Выполняется инструкция:

JSR PC,$CHROT

Процедура **$CHROT** печатает на терминале знак,содержащийся в **R0**,и затем возвращает управление в **ALR.**

4.4.2. Процедуры математических операций и функций.

Подпрограммы на языке макроассемблера могут использовать процедуры математических операций и функций для выполнения операций и функций,доступных в программе **BASIC**. **ALR** может использовать точно те же процедуры,которые выполняются собственно интерпретатором **BASIC.**В этом случае **ALR** не должна дублировать процедуры,которая уже существует в **BASIC.**

В таблице 4.2 и 4.3 представлены математические операции и функции и описаны способы их использования при программировании на языке **BASIC** ,а также приведены имена процедур **BASIC** ,которые обеспечивают выполнение данных операций и функций.Следует отметить ,что некоторыми операциями и функциями требуется одна процедура для арифметики обычной точности,другая-для арифметики двойной точности и ещё другая для арифметики целых чисел.

Если запущена система **BASIC** с арифметикой двойной точности, можно использовать имена процедур с обычной или с удвоенной точностью.Таким образом,можно использовать один и тот же код,

независимо от точности системы. Все же пользователь должен помнить, какая точность установлена в системе и убедиться в правильном преобразовании данных в программе (т.е. соответствие данных с используемыми аппаратными свойствами).
Для согласования с системой FORTRAN IV рекомендуется использовать только имена процедур с двойной точностью.
Все процедуры, имена которых начинаются со знака доллара ($), необходимо вызывать в режиме объектного кода. Для вызова процедур в этом режиме предварительно необходимо вызвать специальную процедуру $POLSH , а затем перечислить имена процедур в режиме объектного кода, которые будут вызываться. Порядок выполнения этих процедур обуславливается очередностью их имен в списке. Все аргументы и результаты пересылаются в стек. Наконец, необходимо указать имя второй специальной процедуры $UNPOL , которая означает конец выполнения в режиме объектного кода.
Имена $POLSH, $UNPOL и имена процедур пользователя должны специфицироваться как глобальные символы.
Формат вызова процедуры $POLSH следующий:

```
JSR  R4, $POLSH
```

Рис.4.5. поясняется состояние стека перед и после выполнения процедуры в режиме объектного кода.

Математические операции **BASIC** Таблица 4.2.

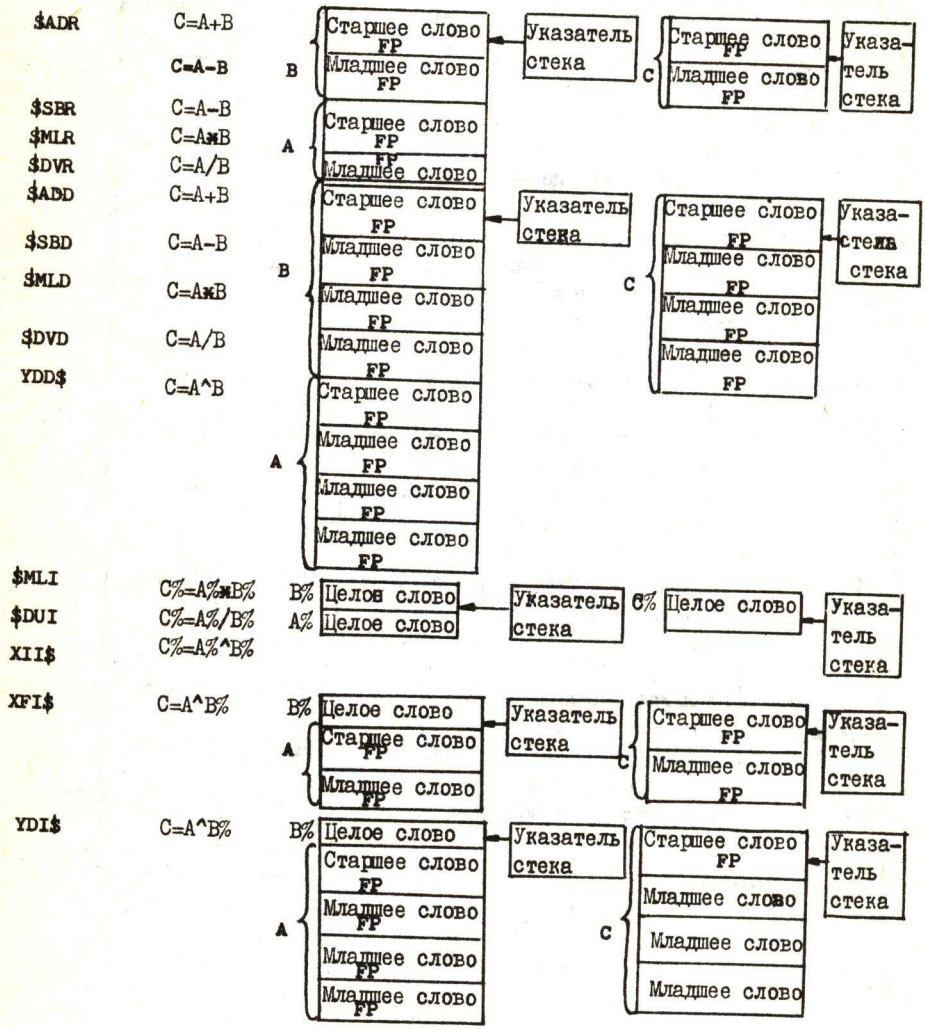
Операция	Знак	Действие	Эквивалент BASIC	Процедура с обычной точностью	Процедура с двойной точностью
Сложение	+	Сложение двух чисел с плавающей запятой.	C=A+B	$ADR	$ADD
Вычитание	-	От одного числа с плавающей запятой отнимается другое такого же типа.	C=A-B	$SBR	$SBD
Умножение	*	Умножаются два числа с плавающей запятой.	C=A*B	$MLR	$MLD
		Умножаются два целых числа.	C%=A%*B%	$MLI	$MLI
Деление	/	Одно число с плавающей запятой разделяется на другое такого же типа.	C=A/B	$DVR	$DVD
		Одно целое число разделяется на другое такого же типа.	C%=A%/B%	$DVI	$DVI
Возведение в степень	^	Число с плавающей запятой возводится в степень такого же типа.	C=A^B	XFF$	XDD$
		Число с плавающей запятой возводится в степень целого типа.	C=A^B%	XFI$	XDI$
		Целое число возводится в степень такого же типа.	C%=A%^B%	XII$	XII$

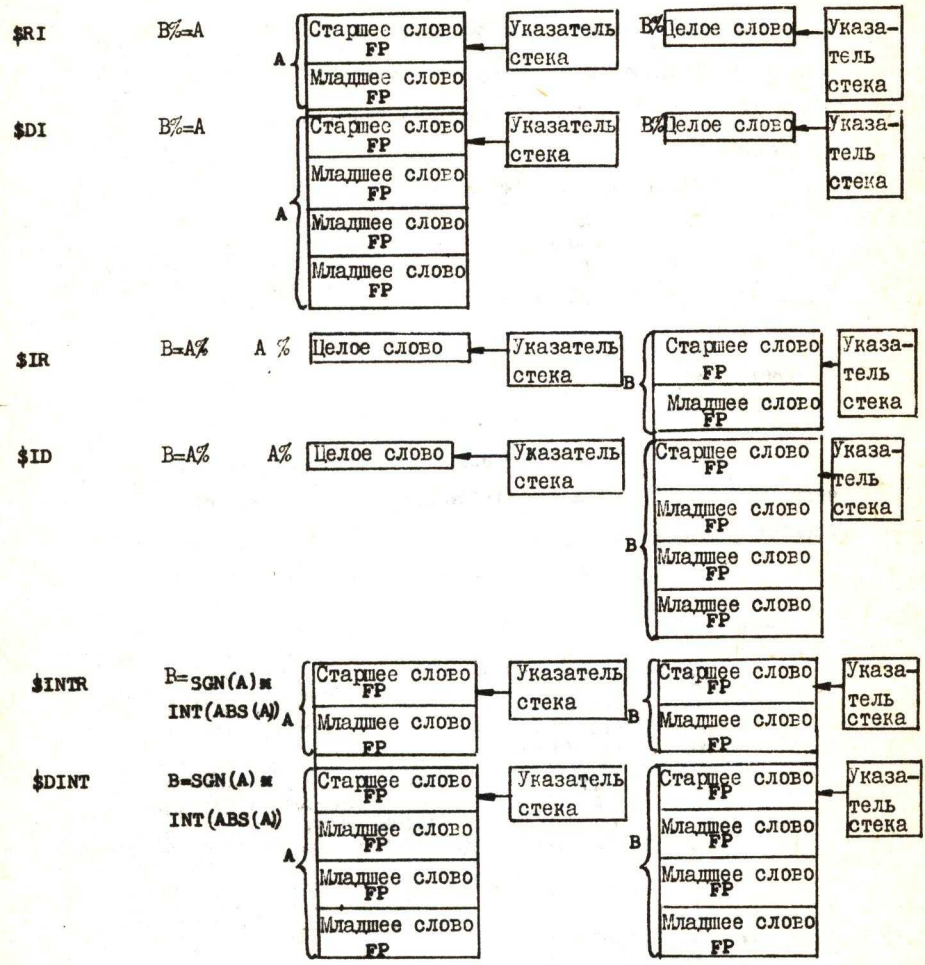
Математические функции BASIC

Таблица 4.3

Функция	Описание	Эквивалент BASIC	Процедура с обычной точностью	Процедура с двойной точностью
Преобразование типа данных	Преобразование числа с плавающей запятой в целое.	B%=A	$RI	$DI
	Преобразование целого числа в число с плавающей запятой.	B=A%	$IR	$ID
Усечение	Усечение числа с плавающей запятой к целой части числа такого же типа.	B=SGN(A)* INT(ABS(A))	$INTR	$DINT
Синус	Определяет синус значения в радианах.	B=SIN(A)	SIN	DSIN
Косинус	Определяет косинус значения в радианах.	B=COS(A)	COS	DCOS
	Определяет в радианах арктангенс числа.	B=ATN(A)	ATAN	DATAN
Логарифм	Определяет натуральный логарифм числа.	B=LOG(A)	ALOG	DLOG
	Определяет десятичный логарифм числа.	B=LOG10(A)	ALOG10	DLOG10
Квадратный корень	Извлекает квадратный корень из числа.	B=SQR(A)	SQRT	DSQRT
Экспоненциальная функция	Определяет значение e, возведенное в степень, представляющую число.	B=EXP(A)	EXP	DEXP

Имена процедур	Эквивалентный оператор BASIC	Стек перед выполнением	Стек после выполнения





ПРИМЕЧАНИЕ: FP означает с плавающей запятой

Рис.4.5. Состояние стека для процедур в режиме объектного кода.

Например, рассмотрим следующие сегменты процедур:

В сегменте I целое число из **TEMP 1** разделяется на целое число из **TEMP 2** и результат заносится в **RESULT:**

```
; SEGMENT 1

        .GLOBL  *POLSH,*UNPOL,*DVI
        MOV     TEMP1,-(SP)             ;SET UP THE
        MOV     TEMP2,-(SP)             ;STACK
        JSR     R4,*POLSH               ;ENTER THREADED CODE MODE
        .WORD   *DVI                    ;SPECIFY ROUTINE NAME
        .WORD   *UNPOL                  ;LEAVE THREADED CODE MODE
        MOV     (SP)+,RESULT            ;STORE RESULT
          .
          .
          .
TEMP1:  .WORD   0
TEMP2:  .WORD   0
RESULT: .WORD   0
```

В сегменте 2 умножаются два числа с плавающей запятой с обычной точностью (**FLOATA** и **FLOATB**) и результат заносится в **FLOATC:**

```
; SEGMENT 2

        .GLOBL  *POLSH,*UNPOL,*MLR
        MOV     FLOATA+2,-(SP)  ;PUT FLOATA
        MOV     FLOATA,-(SP)    ;ON STACK
        MOV     FLOATB+2,-(SP)  ;PUT FLOATB
        MOV     FLOATB,-(SP)    ;ON STACK
        JSR     R4,*POLSH       ;ENTER THREADED CODE MODE
        .WORD   *MLR            ;SPECIFY ROUTINE NAME
        .WORD   *UNPOL          ;LEAVE THREADED CODE NAME
        MOV     (SP)+,FLOATC    ;STORE RESULT
        MOV     (SP)+,FLOATC+2  ;IN FLOATC
          .
          .
          .
FLOATA: .WORD   0,0
FLOATB: .WORD   0,0
FLOATC: .WORD   0,0
```

В сегменте 3 осуществляется преобразование числа с плавающей запятой с двойной точностью в целое число; результат заносится в INTMWD:

```
    $SEGMENT 3

            .GLOBL  *POLSH,*UNPOL,*DI
            MOV     FLOAT+6,-(SP)    ;PUT FLOAT
            MOV     FLOAT+4,-(SP)    ;ON STACK
            MOV     FLOAT+2,-(SP)    ;KEEP DOING IT
            MOV     FLOAT,-(SP)      ;DONE
            JSR     R4,*POLSH        ;ENTER THREADED CODE NAME
            .WORD   *DI              ;SPECIFY ROUTINE NAME
            .WORD   *UNPOL           ;LEAVE THREADED CODE NAME
            MOV     (SP)+,INTMDW     ;STORE RESULT
             .
             .
             .
    FLOAT:  .WORD   0,0,0,0
    INTMDW: .WORD   0
```

Так как в предыдущих примерах за каждым вызовом $POLSH следовало одно имя процедуры, пользователь может специфицировать любое количество имен процедур. За последним из них необходимо специфицировать процедуру $UNPOL.

Всякая процедура синуса, косинуса, арктангенса, логарифма, квадратного корня и возведения в степень использует список аргументов, аналогичный списку аргументов оператора CALL. ALR должна специфицировать список аргументов до выполнения вызова процедуры.

Формат списка аргументов для процедур обычной точности SIN, COS, ATAN , ALOG , ALOG10, SQRT и EXP следующий:

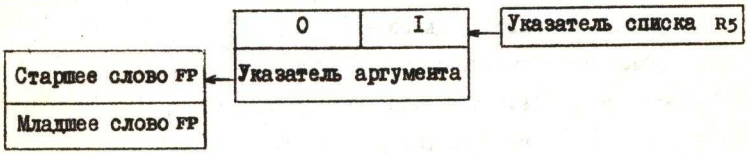


Рис.4.6. Список аргументов для процедур обычной точности.

Формат списка аргументов для процедур двойной точности DSIN,DCOS DATAN , DLOG , DLOG10 , DSQRT и EXP следующий:

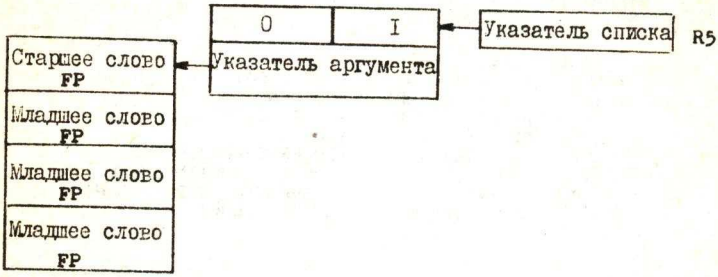


Рис.4.7. Список аргументов для процедур двойной точности.

В обоих случаях процедуры вызываются следующей инструкцией:

JSR PC,routine name

где: **routine name** -имя процедуры

Процедуры обычной точности возвращают результат в R0 и R1; слово старшего порядка помещается в R0,а младшего порядка--в R1.

Процедуры двойной точности возвращают результат в R0,R1,R2 и R3.Слово старшего порядка помещается в R0,а слово младших порядков- в регистры R1, R2 и R3 .

Любое имя вызываемой процедуры должно быть глобальным символом.Этими процедурами не защищаются никакие регистры.

ПРИМЕЧАНИЕ

Перед установкой указателя аргумента для процедур пользователь обязан сохранить начальное значение в R5.Это значение должно сохраняться для выполнения любых процедур в режиме объектного кода,осуществля-

щих доступ к аргументам.

Рассмотрим следующий сегмент процедуры, который вычисляет квадратный корень из числа с плавающей запятой и обычной точности **NUM1** и сохраняет результат в **NUM2**:

```
; SEGMENT WHICH FINDS SQUARE ROOT
        .GLOBL  SQRT
        MOV     R5,TEMP5        ;SAVE OLD VALUE OF R5
        MOV     R1,TEMP1        ;SAVE ANY OTHER REGISTER
        MOV     R0,TEMP0
        MOV     #ARG,R5         ;SET UP R5
        JSR     PC,SQRT         ;CALL ROUTINE
        MOV     R0,NUM2         ;STORE HIGH ORDER RESULT
        MOV     R1,NUM2+2       ;STORE LOW ORDER RESULT
        MOV     TEMP5,R5        ;RESTORE SAVED
        MOV     TEMP1,R1        ;REGISTER
        MOV     TEMP0,R0
          .
          .
          .
ARG:    .WORD   1
        .WORD   NUM1
TEMP5:  .WORD   0
TEMP1:  .WORD   0
TEMP0:  .WORD   0
NUM1:   .FLT2   4
NUM2:   .FLT2   0
```

Нижеуказанный пример демонстрирует укомплектованную подпрограмму на языке макроассемблера, которая вызывается следующим оператором:

```
CALL HYPOT(A,B,C,C%)
```

Подпрограмма вычисляет значение выражения **SQR(A*A+B^2)** присваивает значение переменной C и присваивает усеченное значение переменной C:

```
        .TITLE  HYPOT
        .PSECT  SUBRS,RO,I

        .GLOBL  HYBTAB
HYBTAB: .BYTE   5
        .ASCII  'HYPOT'

        .EVEN
        .WORD   HYPOT

        .GLOBL  *ARGER,*BOMB,*POLSH,*UNPOL
        .GLOBL  *MLR,XFI,*ADR,*SQRT,*RI
```

```
HYPOT:  CMPB    (R5)+,#4        ;ARE THERE 4 ARGUMENTS?
        BEQ     20$             ;YES.
10$:    JMP     $ARGER          ;NO, ISSUE ARGUMENT ERROR.
20$:    CMPB    (R5)+,#202      ;ARE WE BEING CALLED BY BASIC
                                ;WITH ARGUMENT DESCRIPTORS?
        BNE     60$             ;NO.
                                ;YES, CHECK THAT THERE IS ENOUGH
                                ;STACK SPACE. 30. BYTES SHOULD BE
                                ;SUFFICIENT.
        MOV     SP,R3           ;SUBTRACT 30. FROM THE CURRENT SP VALUE.
        SUB     #30.,R3
        CMP     R3,R4           ;IS IT BELOW THE LIMIT?
        BHIS    30$             ;NO.
        JSR     R1,$BOMB        ;YES, ISSUE MESSAGE.
        .ASCIZ  'STACK OVERFLOW IN HYPOT'
        .EVEN
30$:    MOV     -4(R5),R4       ;GET THE POINTER TO THE FIRST ELEMENT
                                ;IN THE ARGUMENT DESCRIPTOR LIST.
        JSR     PC,GETDSC       ;GET THE DESCRIPTOR OF THE 1ST ARGUMENT.
        BIC     #160201,R3      ;IS IT A 2 WORD REAL VALUE?
        CMP     #2040,R3
        BNE     10$             ;NO.
        JSR     PC,GETDSC       ;YES, GET THE DESCRIPTOR OF THE 2ND ARGUMENT.
        BIC     #160201,R3      ;IS IT ALSO A 2 WORD REAL?
        CMP     #2040,R3
        BNE     10$             ;NO.
        JSR     PC,GETDSC       ;GET THE DESCRIPTOR OF THE 3RD ARGUMENT.
        BIC     #160001,R3      ;IS IT A 2 WORD REAL WITH WRITING ALLOWED?
        CMP     #2040,R3
        BNE     10$             ;NO.
        JSR     PC,GETDSC       ;GET THE DESCRIPTOR OF THE 4TH ARGUMENT.
        BIC     #160001,R3
        CMP     #1022,R3        ;IS IT AN INTEGER WITH WRITING ALLOWED?
        BNE     10$             ;NO.
60$:    MOV     (R5)+,R3
                                ;PUSH THE 1ST ARGUMENT ON THE STACK.
        MOV     2(R3),-(SP)     ;NOTE: LOW ORDER IS PUSHED FIRST.
        MOV     (R3),-(SP)
        MOV     2(R3),-(SP)     ;PUSH IT AGAIN BECAUSE WE WILL DO
        MOV     (R3),-(SP)      ;A*A TO GET A^2.
        JSR     R4,$POLSH
        $MLR
        $UNPOL                  ;DO THE MULTIPLY.
        MOV     (R5)+,R3        ;PUSH THE 2ND ARGUMENT.
        MOV     2(R3),-(SP)
        MOV     (R3),-(SP)
        MOV     #2,-(SP)        ;PUSH A 2 BECAUSE WE WILL USE REAL
                                ;TO INTEGER EXPONENTIATION.
        JSR     R4,$POLSH
        XFI$
        $ADR                    ;SQUARE THE 2ND ARGUMENT.
                                ;ADD SQUARE OF 2ND ARGUMENT TO SQUARE
        $UNPOL                  ;OF FIRST ARGUMENT.

                                ;NOW CREATE ON THE STACK THE ARGUMENTS
                                ;REQUIRED BY SQRT.
        MOV     R5,-(SP)        ;SAVE R5
        MOV     SP,R5           ;CREATE POINTER TO VALUE ON THE STACK.
        TST     (R5)+
        MOV     R5,-(SP)
        MOV     #1,-(SP)        ;SHOW ONLY 1 ARGUMENT TO SQRT.
        MOV     SP,R5
        JSR     PC,SQRT         ;GET THE SQUARE ROOT.
```

```
        CMP     (SP)+,(SP)+     ;REMOVE OLD ARGUMENTS FROM THE STACK.
        MOV     (SP)+,R5        ;RESTORE R5.
        MOV     (R5)+,R3        ;POINT TO THE 3RD ARGUMENT.
        MOV     R0,(R3)+        ;STORE THE REAL RESULT IN THE
        MOV     R1,(R3)         ;3RD ARGUMENT.
                                ;NOTE: SQRT RETURNED ITS RESULT IN R0 & R1.
        MOV     R1,2(SP)        ;REPLACE THE SUM OF THE SQUARES
        MOV     R0,(SP)         ;WITH ITS SQUARE ROOT.
        JSR     R4,*POLSH
        *RI                     ;CONVERT TO AN INTEGER.
        *UNPOL
        MOV     (SP)+,@(R5)+    ;STORE THE INTEGER RESULT IN
                                ;THE 4TH ARGUMENT.
        RTS     PC              ;RETURN TO THE CALLER.

;GETDSC RETURNS THE NEXT ARGUMENT'S DESCRIPTOR WORD.

;INPUTS:
;       R4 POINTS TO THE WORD IN THE DESCRIPTOR LIST.

;OUTPUTS:
;       R3 CONTAINS THE DESCRIPTOR WORD FOR THE CURRENT ARGUMENT.
        R4 IS UPDATED TO POINT TO THE NEXT ELEMENT IN THE LIST.

GETDSC: MOV     (R4)+,R3        ;GET THE DESCRIPTOR.
        BIT     #1,R3           ;IS IT A POINTER?
        BNE     10$             ;NO.
        MOV     (R3),R3         ;YES, GET THE ACTUAL DESCRIPTOR.
10$:    RTS     PC
        .END
```