BASIC–11/RT–11 User's Guide

AA–5071B–TC

digital

software

BASIC–11/RT–11 User's Guide

AA–5071B–TC

March 1983

This document describes the system-dependent features of BASIC–11/RT–11. In conjunction with the *BASIC–11 Language Reference Manual* (AA–1908A–TC), this document provides the information required to write and run a BASIC–11 program under the RT–11 operating system.

This document supersedes the *BASIC–11/RT–11 User's Guide* (AA–5071A–TC).

Operating System: RT–11 Version 5.0

Software: BASIC–11/RT–11 Version 2.1

To order additional documents from within DIGITAL, contact the Software Distribution Center, Northboro, Massachusetts 01532.

To order additional documents from outside DIGITAL, refer to the instructions at the back of this document.

digital equipment corporation · maynard, massachusetts

First Printing, September 1977
Revised, March 1983

Printed in U.S.A.

A postage-paid READER'S COMMENTS form is included on the last page of this document. Your comments will assist us in preparing future documentation.

M21800

CONTENTS

FIGURES

TABLES

PREFACE

Before reading this manual, you should be familiar with the BASIC-11 language and the RT-11 operating system. If necessary, read the following manuals before reading this user's guide:

- BASIC-11 Language Reference Manual (AA-1908A-TC)

- Introduction to RT-11 (AA-5281C-TC)

  or

- RT-11 System User's Guide (AA-5279C-TC)

Most features of BASIC-11/RT-11 V2.1 are the same as in other versions of BASIC-11 (DIGITAL's name for the family of BASICs for the PDP-11). These features are described in the BASIC-11 Language Reference Manual (AA-1908A-TC).

Some features of BASIC-11/RT-11 are specific to the RT-11 operating system software, and may differ from other BASIC language software.

This guide describes the following system-dependent features of BASIC-11/RT-11:

- Procedure for starting BASIC-11
- Effect of the CTRL/C key command
- Accuracy of storing numbers
- Format of error messages
- Format of the file specification
- Effects of parameters in the OPEN statement
- Procedure for checking files
- Effect of superseding files
- Effects of the utility functions
- Procedure for using assembly language routines
- Procedure for terminating BASIC-11

All BASIC-11 users should read all of this guide except Chapter 4. Only users who are adding assembly language routines to BASIC-11 need to read Chapter 4. Chapter 4 is written for the experienced RT-11 MACRO-11 programmer.

This guide assumes that you have linked your BASIC-11 software according to the procedure described in the BASIC-11/RT-11 Installation Guide.

Documentation Conventions

This section describes the documentation conventions, notations, and symbols used throughout this manual.

The following symbols denote special terminal keys that you will use frequently when using BASIC-11.

Symbol	Meaning
<CTRL/X>	While pressing the CTRL key, type the letter indicated after the slash.
<RET>	Type the RETURN key.
<ESC>	Type the ESCAPE key (ALTMODE on some terminals).
<DEL>	Type the DELETE key (RUBOUT on some terminals).

In addition, this manual uses certain conventions when describing the format of statements, functions, and commands.

These are:

Convention	Meaning
	The enclosed elements are optional. For example: LET variable=expression
	A choice of one element among two or more possibilities, for example: IF relational expression THEN statement / THEN line number / GO TO line number
...	Preceding element can be repeated as indicated. For example: line number CLOSE#expr1,#expr2,...
Items in capital letters and special symbols	Type these elements exactly as they appear in the format, for example: LET RUN # Items in capital letters are called keywords.
Items in lowercase letters	Replace these elements according to the description provided in text. See below for list of commonly used lowercase items.

This list describes some lowercase items commonly used in format descriptions. The general meaning of each item is given. Unless a specific format description places restrictions on an item, its general meaning applies. See the BASIC-11 Language Reference Manual for more information on these items.

Lowercase Item	Abbreviation	Meaning
expression	expr	Any valid BASIC-11 expression. It is always a numeric expression unless the description specifically states that it can be a numeric or string expression. For example: (5*SIN(X))^Y
file specification	---	A file specification as described in Section 2.1
integer	int	Any positive integer number constant or any positive numeric constant that could be an integer if a percent sign were put after it. For example: 5%, 3%, 2, 7
line number	---	Any valid line number. For example: 10, 100, 32767
string	---	Any string expression. For example: "ABC", C$+SEG$(A$,3,4)
variable	var	A floating-point, integer, or string variable.

If more than one of the same lowercase word appears in a format, the words are numbered 1, 2, 3, etc. For example:

```
CLOSE #expr1,#expr2,#expr3,...
```

Examples of computer output and input are presented in **bold type.** To differentiate between what BASIC-11 prints and what you type, the user input is printed in red ink. For example:

```
RUNNH

WHAT NUMBERS?  5,10
THE SUM IS 15

READY
```

All user input is terminated by the RETURN key unless the text indicates a different terminator.

CHAPTER 1

GETTING STARTED WITH BASIC-11/RT-11

1.1 OPTIONAL FEATURES

BASIC-11/RT-11 provides many optional features. If you include all optional features, you can perform all operations described in the BASIC-11 Language Reference Manual or in this guide. By excluding some or all optional features, you can increase the amount of memory available for programs, increase the speed of program execution, or both.

Optional Statements:

CALL
PRINT USING

Optional Commands:

SUB
RESEQ

Optional Functions:

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

Miscellaneous Optional Features:

- Double-precision arithmetic
- Short error messages
- Exponentiation (for example, the expression A^B)
- Ability to run BASIC-11 as foreground or background job
- Features affecting program space availability and program execution speed

You must specify the inclusion or exclusion of some optional features when you link BASIC-11. You select other optional features when you run BASIC-11. The features you can choose when you link BASIC-11 are:

- All optional statements
- All optional commands
- SQR, SIN, COS, ATN, EXP, LOG, and LOG10 functions
- All miscellaneous optional features

The features you can choose at run time are the following optional functions:

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

Before using BASIC-11 you must link a version with the optional features you want. See the BASIC-11/RT-11 Installation Guide for instructions to link BASIC-11 and for information about allowed program size and speed of execution tradeoffs.

1.2 STARTING BASIC-11

You can use BASIC-11 with the single-job (SJ), the foreground/background (FB), or the extended memory (XM) RT-11 Version 5 monitor. When using the FB or XM monitor, you can run BASIC-11 as either the foreground or the background job.

Before starting BASIC-11, you must bootstrap the RT-11 operating system and enter the DATE and TIME commands. See the Introduction to RT-11 for a description of these procedures.

1.2.1 Running BASIC-11 with the SJ Monitor or as the Background Job

To run BASIC-11 with the SJ monitor or as the background job under the FB or the XM monitor, enter either the BASIC or the RUN command. The BASIC command runs the file BASIC.SAV on your system device. To enter the BASIC command, type:

```
.BASIC
```

To use another version of BASIC, type:

```
.RUN file specification
```

where:

file specification	specifies the file containing the version of BASIC that you want.

For example, if you have a version of BASIC on device DX1: with file name BAS8K, and you want that version instead of the one in BASIC.SAV, you should enter:

```
.RUN DX1:BAS8K
```

If you specify a file that does not exist, RT-11 prints the message:

```
?KMON-F-File not found DEV:FILNAM.TYP
```

If there is not enough room in memory to run BASIC-11, one of the following messages is printed:

```
NOT ENOUGH MEMORY FOR BASIC

     or

?KMON-F-Insufficient memory
```

This error often results from a large foreground job that has not been unloaded.

If there are no errors, BASIC-11 prints an identifying message and inquires whether you want the optional functions that you can select at run time.

```
.BASIC
BASIC-11/RT-11 V2.1
OPTIONAL FUNCTIONS (ALL, NONE, OR INDIVIDUAL)?
```

To include all of the optional functions, type A<RET>. To exclude all of the optional functions, type N<RET>. (You must always terminate input to BASIC-11 with the RETURN key.) BASIC-11 then prints the READY message. For example:

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

READY
```

Typing only the RETURN key in response to the optional functions request is equivalent to typing A.

If you want to include some but not all optional functions, type I<RET>. BASIC-11 then prints a query for each function individually. To include a function type a Y; otherwise type an N. Typing only the RETURN key in response to the function request is equivalent to typing Y. If you type anything else, BASIC-11 repeats its request. After you have typed a Y or an N in response to each function query, BASIC-11 prints the READY message. For example:

```
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? N
ASC? N
CHR$? N
POS? N
SEG$? N
VAL? N
TRM$? N
STR$? N
PI? N
INT? Y
DAT$? N
CLK$? N

READY
```

1.2.2 Running BASIC-11 as the Foreground Job

To run BASIC-11 as the foreground job, use the FRUN command. Type:

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

where:

file specification	specifies the file containing BASIC-11
number	is the size of the user area (that is, the number of words to be reserved). It must be 1000. or greater. The decimal point identifies the number as decimal, not octal.

You must specify the user area size, or else no area will be reserved and BASIC-11 will not be able to run.

The user area will actually be approximately 100 words more than you request. For example, the following command reserves approximately 3100 words.

```
.FRUN BASIC/N:3000.
```

If the file specified does not exist, RT-11 prints the message:

```
?KMON-F-File not found DEV:FILNAM.TYP
```

If the number of words requested in the FRUN command is not large enough, BASIC-11 prints the message:

```
NOT ENOUGH MEMORY FOR BASIC
```

If there are no errors, RT-11 prints its prompting dot (.). After the CTRL/F command is typed the F> characters are printed to indicate that command input is being directed to the foreground job. BASIC-11 then prints an identifying message and inquires whether you want the optional functions. For example:

```
.FRUN BASIC/N:3000.

.<CTRL/F>
F>
BASIC-11/RT-11 V2.1
OPTIONAL FUNCTIONS (ALL, NONE, OR INDIVIDUAL)?
```

Type a CTRL/F in response to the dot prompt and answer the optional function query as described in the previous section.

NOTE

To use a device other than the system device, you must load the handler before you run BASIC-11 in the foreground. See the _RT-11 System User's Guide_ for more information about foreground jobs.

1.2.3 Running BASIC-11 from an Indirect File

You can run BASIC-11 and answer the initial dialog by using an indirect file. You can only run BASIC-11 in this way as the background job or in the single-job monitor. This technique is useful when you select the optional functions individually.

You cannot enter any BASIC-11 command, program line, or immediate mode statement through an indirect file.

To create the indirect file, direct the editor to create a file with a file type .COM that contains all anticipated responses to system queries. For example:

```
.R EDIT
*EWMINRUN.COM <ESC><ESC>
*IR BASIC
I
N
N
N
N
N
N
Y
Y
Y
Y
Y
N
N
N
N
N
N
N
N
N
Y
N
N
<ESC><ESC>
*EX <ESC><ESC>
.
```

To start BASIC-11, type an at sign (@) followed by the file name. The complete initial dialog is printed on the terminal. For example:

```
.@MINRUN

.R BASIC
BASIC-11/RT-11 V2.1
OPTIONAL FUNCTIONS (ALL, NONE, OR INDIVIDUAL)? I
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
```

See the _RT-11 System User's Guide_ for more information on using indirect files.

1.3 STOPPING BASIC-11 PROGRAMS (CTRL/C COMMAND)

To stop execution of a BASIC-11 program, use the CTRL/C command. If you type one CTRL/C, BASIC-11 interrupts your program the next time it requests input. If you type two consecutive CTRL/Cs, BASIC-11 interrupts your program immediately. After BASIC-11 interrupts your program, it prints:

```
STOP AT LINE xxxxx

READY
```

where:

xxxxx   is the number of the line that BASIC-11 was executing when the CTRL/C command halted the program.

However, if you were not executing a program line, BASIC-11 prints:

```
STOP

READY
```

When you type CTRL/C, the system prints ^C. For example:

```
10 GO TO 10
RUNNH

^C ^C
STOP AT LINE 10

READY
```

NOTE

CTRL/C does not return control to the RT-11 monitor. You must type the BYE command (see Section 1.4) to return control to RT-11.

1.4 TERMINATING THE SESSION (BYE COMMAND)

To terminate a session with BASIC-11, type the BYE command. The BYE command returns control to the RT-11 monitor, which prints its prompting dot. For example:

```
BYE

.
```

Once you have entered the BYE command you cannot use the RT-11 REENTER command to return to BASIC-11. Instead, you must restart BASIC-11 as described in Section 1.2. If you want to reuse your BASIC-11 program, save it before you enter the BYE command.

If you ran BASIC-11 as the foreground job, you must unload it after you enter the BYE command. Type:

```
.UNLOAD FG

.
```

1.5 FLOATING-POINT NUMBER PRECISION

You can use BASIC-11 with either single- or double-precision arithmetic. Single-precision arithmetic allows floating-point numbers to seven digits of precision. Thus, single-precision BASIC-11 stores the numbers 1.000001 and 1.000000 (seven digits) differently but stores 1.0000001 and 1.0000000 (eight digits) as the same number. Double-precision arithmetic allows you to specify floating-point numbers to 15 digits of precision.

If you need more than seven digits of precision, you should use BASIC-11 with double-precision arithmetic. However, double-precision BASIC-11 has two disadvantages:

1. It allows less BASIC-11 program space, because BASIC-11 itself requires more memory and because all floating point constants, variables, and arrays require twice the amount of memory that single-precision needs.

2. Arithmetic operations and functions run more slowly with double precision than with single precision.

The PRINT statement only prints six digits even when you are using double-precision arithmetic. Consequently, if you want to print a number with more than six digits, you must use the PRINT USING statement or the STR$ function. The following example was run using double-precision arithmetic.

```
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
```

Double-precision programs compiled by BASIC-11 are assigned the default file type .BAX and single-precision programs compiled by BASIC-11 are assigned the default file type .BAC. The different default file types are necessary because double-precision BASIC-11 cannot read a program compiled by single-precision BASIC-11, and vice versa. If you are using double-precision BASIC-11 and you specify the file type of a program compiled by single-precision BASIC-11, or vice versa, the results are unpredictable.

1.6 SYSTEM-DEPENDENT ERROR MESSAGES

Some of the error messages listed in the BASIC-11 Language Reference Manual either have special meaning in BASIC-11/RT-11 or are not produced by it. These error messages are:

?CANNOT DELETE FILE (?CDF)
BASIC-11/RT-11 does not produce this message.

?ERROR CLOSING CHANNEL (?ECC)
BASIC-11/RT-11 does not produce this error message. If an error occurs when BASIC-11/RT-11 is trying to close a channel, BASIC-11/RT-11 prints the message ?CHANNEL I/O ERROR (?CIE).

?FILE ALREADY EXISTS (?FAE)
BASIC-11/RT-11 does not produce this message.

?FILE PRIVILEGE VIOLATION (?FPV)
BASIC-11/RT-11 does not produce this message.

?FILE TOO SHORT (?FTS)
The file is too small to contain the output. If the error occurs in a data file, specify a larger FILESIZE. If the error occurs in a program file, delete unused files with the UNSAVE command and then try the operation again.

?ILLEGAL DEF (?IDF)
BASIC-11/RT-11 does not produce this message.

?ILLEGAL FILE LENGTH
The FILESIZE specified is less than -1 (see Section 2.2).

?ILLEGAL RECORD SIZE (?IRS)
BASIC-11/RT-11 does not produce this message.

?NOT A VALID DEVICE (?NVD)
BASIC-11/RT-11 does not produce this message.

?NOT ENOUGH ROOM (?NER)
Not enough room is available for the FILESIZE specified. Delete unused files with the UNSAVE command.

CHAPTER 2

FILES

2.1 FILE SPECIFICATION

BASIC-11 uses the standard RT-11 file specification format. The format is:

device:filename.type

where:

device — is the device name. It can be any device name listed in Table 2-1 or any assigned device name (see the RT-11 System User's Guide).

filename — is the one- to six-character name of the file.

type — is the zero- to three-character type of the file.

Table 2-1
RT-11 Device Names

Code	Device
DLn:	RL01/02 Disk
DMn:	RK06/07 Disk
DUn:	RC25/RD51 Disk, RX50 Diskette
DXn:	RX01 Diskette
DYn:	RX02 Diskette
LP:	Line printer
LS:	Serial line printer
MMn:	TJU16 Magtape
MTn:	TM11 Magtape

(continued on next page)

Table 2-1 (Cont.)
RT-11 Device Names

Code	Device
RKn:	RKØ5/RK11 Disk
TT:	Console terminal keyboard/printer
SYn:	System device (the volume from which the monitor was bootstrapped)
DK:	The default storage volume

If you do not specify any of the elements of the file specification, BASIC-11 uses a default value.

The default device is DK:. The default for the file name and file type depends on the statement or command in which the file specification appears. Table 2-2 shows the file name defaults, and Table 2-3 shows the file type defaults.

Table 2-2
Default File Names

Statement or Command	Default
SAVE,REPLACE,COMPILE	the current program name
OLD,APPEND,CHAIN OVERLAY	the file name NONAME
UNSAVE,OPEN,KILL NAME	no default but prints the ?ILLEGAL FILE SPECIFICATION (?IFS) error message instead

Table 2-3
Default File Types

Statement or Command	Single-precision BASIC-11 Default	Double-precision BASIC-11 Default
OPEN,KILL,NAME	.DAT	.DAT
SAVE,REPLACE,UNSAVE APPEND	.BAS	.BAS
COMPILE	.BAC	.BAX
RUN,OLD	.BAC (.BAS if a .BAC cannot be found)	.BAX (.BAS if a .BAX cannot be found)

When you create a file whose file specification is the same as an existing file, the older file is deleted (superseded) when the new file is closed. You can avoid accidental deletions by using the SAVE command to save new files. If the SAVE command specifies a file name that already exists, BASIC-11 prints the following error message:

?USE REPLACE (?RPL)

This gives you an opportunity to decide whether you want to supersede the old file, or to store the file under a different file specification.

2.2 THE OPEN STATEMENT - SYSTEM-DEPENDENT FEATURES

The format of the OPEN statement is:

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

where:

string	is a file specification as described in Section 2.1.
expr1	is the channel number of the file. It can have any value between 1 and 12.
DOUBLE BUF	causes the file to be double buffered. Double buffering increases the speed of some file operations but requires additional memory for the second buffer.
RECORDSIZE expr2	is ignored if specified.
MODE expr3	is ignored if specified.
FILESIZE expr4	if positive, specifies the maximum number of 256-word blocks the file can occupy. If FILESIZE is missing or expr4 equals Ø, it requests the standard BASIC-11/RT-11 file allocation (that is, either half the largest free area or all of the second-largest free area, whichever is larger). If expr4 equals -1, it requests the absolute largest free area. If expr4 is less than -1, the error message ?ILLEGAL FILE LENGTH appears.

The elements of the OPEN statement described above are the system dependent elements. The other elements of the OPEN statement are described in the BASIC-11 Language Reference Manual.

2.3 LISTING YOUR FILE DIRECTORY

You must return control to the RT-11 monitor before you list your file directory. First save your current BASIC-11 program (if you wish to reuse it later) and then enter the BYE command. The monitor prints the dot prompt. For example:

```
SAVE TEMP

READY
BYE
.
```

Following the prompt, type the RT-11 DIRECTORY command. A simplified format of the RT-11 directory command (see the RT-11 System User's Guide for a complete description) is:

```
DIRECTORY   /PRINTER    file specification
```

where:

PRINTER	specifies that the directory is to be printed on the line printer. (If omitted, the directory is printed on the terminal.)
file specification	specifies the files that you want listed. If you omit the file specification, all files are listed.

The DIRECTORY command wildcard feature allows you to specify files with similar file names, similar file types, or both. If you substitute an asterisk for the file name and specify a file type, all files with that file type are listed. For example, the following command lists all BASIC-11 source programs on the line printer:

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

Similarly, if you substitute an asterisk for the file type, and specify a file name, all files with that file name are listed, regardless of their file types. For example, the following command lists all files with the file name TEST:

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

If you specify a percent sign in place of any characters in a file name or file type (for example, TEST%%.BAS), all the files whose specifiers match the other characters in the specification are listed (TESTAB.BAS, TEST01.BAS, and TESTER.BAS would be listed, if they existed, for the specification TEST%%.BAS).

To list all the BASIC-11 programs and all the compiled BASIC-11 programs, type:

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

Note that this command also lists files with the file type .BAK and .BAT. Because the /PRINTER option was not specified, the listing is printed on the terminal.

After you list your directory, return to BASIC-11 by using the BASIC command. Restore your saved program with the OLD command, and then, delete the temporary file. For example:

```
.BASIC
BASIC-11/RT-11  V2.1
OPTIONAL FUNCTIONS (ALL, NONE, OR INDIVIDUAL)? A

READY
OLD TEMP

READY
UNSAVE TEMP

READY
```

CHAPTER 3

UTILITY FUNCTIONS

3.1 BASIC-11 UTILITY FUNCTIONS

BASIC-11 has utility functions to:

- Change the terminal width (TTYSET)
- Cancel the effect of CTRL/O (RCTRLO)
- Disable CTRL/C (CTRLC and RCTRLC)
- Terminate your program (ABORT)
- Input a single character from your terminal (SYS)
- Terminate BASIC-11 (SYS)
- Check if a CTRL/C has been typed (SYS)
- Enable lowercase support (SYS)

In the following sections, BASIC-11 utility functions are shown in the context of a LET statement with a dummy target variable, as follows:

```
LET variable = (utility function)
```

where:

variable	is the target variable.
utility function	is one of the functions described in this chapter.

Utility functions can appear in any arithmetic expression. The LET statement format is recommended because it is the simplest statement.

3.2 SETTING THE TERMINAL MARGIN (TTYSET FUNCTION)

Use the TTYSET function to set your terminal's right margin. BASIC-11 prints on a line until a number or string extends past the margin you set. BASIC-11 then prints a return and line feed on the current line and prints the string or number on the next line.

The format of the TTYSET function is:

```
LET variable=TTYSET(255%,expression)
```

where:

variable — is the target variable and contains an undefined value after the statement is executed.

255% — is either a numeric constant (as specified in format) or an expression with an integer value of 255 (for compatibility with other versions of BASIC).

expression — specifies the right margin of the terminal. The margin is set to the value of the expression minus 1. If the expression equals Ø, BASIC-11 does not change the previous margin.

For example, to set BASIC-11 to print to the full width of an LA36 DECwriter II (132 columns), type:

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

To set BASIC-11 to print to the full width of a VT5Ø display terminal (8Ø columns), type:

```
A=TTYSET(225%,81%)
```

If you do not specify the TTYSET function, BASIC-11 assumes a terminal with 72 columns.

Make sure that the system's margin for your terminal is equal to or greater than the margin you specify in TTYSET.

If the value of the expression is less than Ø, equal to 1, or greater than 256, BASIC-11 prints the ?ARGUMENT ERROR (?ARG) message. If the first argument has a value other than 255, BASIC-11 prints the same message.

3.3 CANCELING THE EFFECT OF CTRL/O (RCTRLO FUNCTION)

BASIC-11 stops terminal output when the CTRL/O key is typed; however, the RCTRLO function causes BASIC-11 to resume printing. Use the RCTRLO function to ensure that certain data is printed on the terminal even if a CTRL/O is typed.

The format of the function is:

```
LET   variable=RCTRLO
```

where:

variable — is the target variable and contains an undefined value after the statement is executed.

Consider the following example:

```
LISTNH
10 REM PROGRAM TO INPUT DATA
20 REM FROM FILE AND PRINT SUM
30 OPEN "NUMBR" 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

 4
 16
 147
 26
<CTRL/O>
SUM= 4172

READY
```

BASIC-11, while executing the loop from line 50 to line 90, prints out numbers. If CTRL/O is typed BASIC-11 stops printing. But when BASIC-11 executes line 100, it resumes printing.

3.4 DISABLING CTRL/C (RCTRLC AND CTRLC FUNCTIONS)

In certain parts of a program you may need to override CTRL/C interrupts from the terminal. The RCTRLC function disables CTRL/C and prevents it from stopping a BASIC-11 program. The CTRLC function enables the CTRL/C command.

The format of the functions are:

```
LET   variable=RCTRLC

LET   variable=CTRLC
```

where:

variable — is the target variable; it contains an undefined value after the statement is executed.

After BASIC-11 executes the RCTRLC function, typing CTRL/C on the terminal does not stop the program.

After BASIC-11 executes the CTRLC function, typing CTRL/C stops the program. BASIC-11 does not save any CTRL/C that is typed while CTRL/C is disabled. If the program encounters a CTRL/C function, and no prior RCTRLC function is in effect, the CTRL/C function has no effect.

When BASIC-11 prints the READY message, it automatically enables the CTRL/C command.

For example:

```
LISTNH
1000 REM DO NOT ALLOW INTERRUPTS
1010 A=RCTRLC
1020 PRINT "NO INTERRUPTS"
1030 FOR I= 1 to 1000 \ S=S+I \ NEXT I
1100 REM NOW ALLOW INTERRUPTS
1110 A=CTRLC
1120 PRINT "INTERRUPTS OKAY"
1130 FOR I = 1 to 1000 \ S=S+I \ NEXT I
32767 END

READY
RUNNH

NO INTERRUPTS

<CTRL/C>

INTERRUPTS OKAY

<CTRL/C>

STOP AT LINE 1130

READY
```

For information on a system function that determines if CTRL/C has been typed while CTRL/C is disabled, see Section 3.6.3.

NOTE

> Once CTRL/C is disabled it is not possible to interrupt BASIC-11. Do not disable CTRL/C until your program is debugged.

3.5 TERMINATING YOUR PROGRAM (ABORT FUNCTION)

If you want a program to delete itself from memory when it terminates, use the ABORT function. The ABORT function is equivalent to an END statement except that ABORT can optionally delete your program from memory and change the program name to NONAME (equivalent to the SCR command).

The format of the ABORT function is:

```
LET   variable=ABORT(expression)
```

where:

variable — is the target variable; it contains an undefined value after the statement is executed.

expression — determines if the program is to be deleted from memory. If the expression equals Ø, BASIC-11 does not delete the program. If the expression equals 1, BASIC-11 deletes the program.

Consider the following examples:

Delete from memory when program completed

```
LIST

ABORT   21-JAN-83   14:52:45

1Ø PRINT "123"
2Ø A=ABORT(1)
3Ø PRINT "456"

READY
RUNNH

123

READY
LIST

NONAME   21-JAN-83   14:53:3Ø


READY
```

Do not delete when program completed

```
LIST

ABORT   21-JAN-83   14:54:ØØ

1Ø PRINT "123"
2Ø A=ABORT(Ø)
3Ø PRINT "456"

READY
RUNNH

123

READY
LIST

ABORT   21-JAN-83   14:54:3Ø

1Ø PRINT "123"
2Ø A=ABORT(Ø)
3Ø PRINT "456"

READY
```

3.6 SYSTEM FUNCTIONS

System functions perform system-dependent operations.

The formats of the system functions are:

```
[LET]   variable= SYS(expression1  [,expression2]  )
```

where:

variable — is the target variable.

expression1 — determines the function to be performed.

expression2 — is an optional argument used in some system functions.

Table 3-1 summarizes the functions performed according to the specified value of expression1. Any value of expression1 other than those specified causes BASIC-11 to print the ?ARGUMENT ERROR (?ARG) message.

Table 3-1
Summary of System Functions

Value of expression1	Function Performed
1	Processes input one character at a time. Target variable contains the ASCII value of the next character typed at the terminal.
4	Terminates BASIC-11 and returns control to the system monitor (equivalent to the BYE command).
6	Determines if CTRL/C has been typed while CTRL/C is disabled by the RCTRLC function. Target variable equals 1 if CTRL/C has been typed and equals Ø if CTRL/C has not been typed.
7	Enables or disables lowercase input from your terminal. If expression2 equals Ø, lowercase character input is allowed. If expression2 equals 1, lowercase character input is converted to the equivalent uppercase character input.

3.6.1 Single Character Input

Use the single character input system function, SYS(1), to process input one character at a time.

SYS(1) returns the seven-bit ASCII value of any character typed on the terminal except CTRL/C (see the BASIC-11 Language Reference Manual for a list of the ASCII values). If CTRL/C is typed when BASIC-11 is executing SYS(1) and CTRL/C is enabled, then BASIC-11 prints the STOP and READY messages. If CTRL/C is disabled, then BASIC-11 continues executing SYS(1) and waits for another character.

```
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
```

3.6.2 Terminating BASIC-11

To terminate BASIC-11 from a BASIC-11 program, use system function SYS(4). It is equivalent in effect to the BYE Command.

For example:

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

READY
RUNNH
GOODBYE
.
```

3.6.3 Checking for CTRL/C

If you have disabled CTRL/C with the RCTRLC function and want to check if CTRL/C has been typed, use system function SYS(6). The function returns a 1 if CTRL/C has been typed and a 0 if it has not been typed.

For example:

```
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
<CTRL/C><CTRL/C>
STILL EXECUTING
PROGRAM TERMINATING

READY
```

3.6.4 Enabling Lowercase Support

If you want to enter lowercase characters at your terminal, use the system function SYS(7,expr2). The RT-11 operating system usually converts all lowercase alphabetic characters to uppercase. Executing the function SYS(7,0) causes RT-11 to stop converting lowercase characters and to pass them unchanged. To cause RT-11 to resume converting lowercase characters, you must execute the function SYS(7,1). After you exit from BASIC-11, the monitor continues to process characters as it did before BASIC-11 was active.

The following example demonstrates how to enable and disable lowercase. The program is first run to enable lowercase by causing the function SYS(7%,0%) to be executed. The program is then modified to allow the user to enter a lowercase response. Finally, the modified form of the program is run; this disables lowercase. The modified program is then saved.

```
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 CHARACTERS (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$<>"n" THEN IF Aa$<>"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
```

If you type lowercase letters when lowercase is disabled, they are echoed as uppercase.

Note that BASIC-11 converts lowercase keywords and variable names to uppercase characters but leaves string constants, strings entered at the terminal, and remarks unchanged.

CHAPTER 4

USING ASSEMBLY LANGUAGE ROUTINES WITH BASIC-11

4.1 INTRODUCTION TO ASSEMBLY LANGUAGE ROUTINES

BASIC-11 allows you to add assembly language routines (ALRs) to expand or extend BASIC-11's capabilities. For example, you can write routines to communicate with special devices (such as laboratory equipment) or to manipulate arrays. Once added to BASIC-11, such routines can be executed in immediate mode or in programs, by means of the CALL statement (see the BASIC-11 Language Reference Manual). The advantages to programming in both BASIC-11 and assembly language rather than programming in just assembly language are listed below.

- Only the programmer writing the routine has to know assembly language. The application programmers only have to know BASIC-11.

- It is easier to write, debug, and modify BASIC-11 programs. You can write, execute, debug, and modify your program without leaving BASIC-11.

- You can execute ALRs without writing a program, using immediate mode CALL statements.

NOTE

> This chapter assumes that you are an experienced MACRO-11 programmer and that you are familiar with your operating system and its utility programs (editors, MACRO-11 assembler, task builders, linkers, and so forth).

This chapter describes:

- ALR format.

- The procedure to access arguments.

- Use of auxiliary routines provided by BASIC-11.

See the BASIC-11/RT-11 Installation Guide and Release Notes for the procedure to add the routines to BASIC-11.

ALRs that use the FORTRAN IV call interface (as defined in RT-11/RSTS/E FORTRAN IV User's Guide) can be called from either FORTRAN IV or RT-11 BASIC-11. However, these ALRs must not access any routines or global locations in FORTRAN IV itself.

4.2 FORMAT OF THE ASSEMBLY LANGUAGE ROUTINE

To write an assembly language routine (ALR) that you can add to BASIC-11, you first must specify the name of the routine and its starting address in the user routine name table (see Figure 4-1). You must include a pointer for each ALR after the global location FTBL. Each pointer specifies the location of the routine name and starting address. A word containing all zeros terminates the pointer list.

NOTE

> ALR names must not contain embedded blanks. For compatibility with FORTRAN IV, routine names longer than six ASCII characters should be avoided (although BASIC-11 imposes no length restriction other than the limit of the program line size).

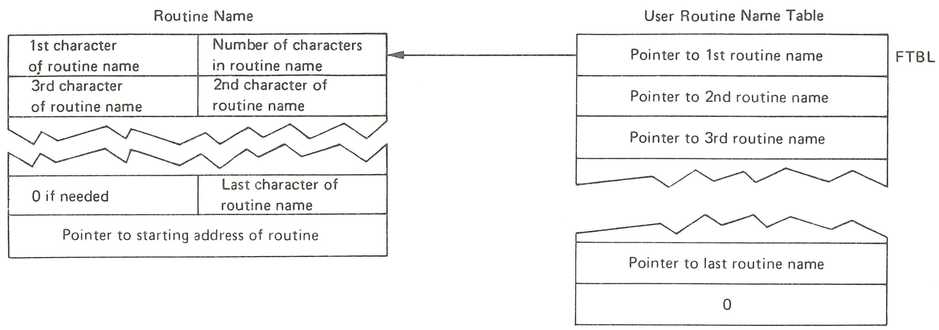

Figure 4-1 User Routine Name Table and Routine Name Formats

The BASIC-11 software kit includes a file BSCLI.MAC, with global location FTBL. This file is the basis of the pointer table. You build the pointer table by adding entries between global location FTBL and the .WORD Ø entry, using the system editor.

Normally, placing the ALR's routine name at the beginning of the routine is recommended. In this case the pointers in the user routine name table should be globals. For example, if you have written three routines named INITIT, ADDER, and CHKSTA, the routine name list should be:

```
                .
                .
                .
        .GLOBL  FTABI
        .GLOBL  INITNM, ADDNM, CHKSNM
FTABI:  .WORD   FTBL
FTBL:   .WORD   INITNM   ;User routine
        .WORD   ADDNM    ;Name list
        .WORD   CHKSNM
        .WORD   Ø
                .
                .
                .
        .END
```

NOTE

> You should edit the items printed in red in this listing into the file BSCLI.MAC. The items printed in black are already in the file.

The locations, INITNM, ADDNM, and CHKNM should be at the beginning of the INITIT, ADDER, and CHCKST, respectively. For example:

```
     ;      The INIT routine
            .GLOBL INITNM
INITNM:     .BYTE 6    ;Number of characters in name
            .ASCII "INITIT"
            .EVEN
            .WORD INITST
INITST:                           ;Start of routine
               .
               .
               .
```

An alternative method is to add the routine name and starting address after the routine name table. In this case the starting addresses of the routines should be globals. Using the same examples as above, the routine name table should be:

```
            .GLOBL     FTABI
            .GLOBL     INITST, ADDST, CHKSST
FTABI:      .WORD      FTBL
FTBL:       .WORD      INITNM
            .WORD      ADDNM
            .WORD      CHKSNM
            .WORD      Ø
INITNM:     .BYTE      6      ;Number of characters in name
            .ASCII     "INITIT"

            .EVEN
            .WORD      INITST
ADDNM       .BYTE      5
            .ASCII     "ADDER"
            .EVEN
            .WORD      ADDST
CHKSNM:     .BYTE      6
            .ASCII     "CHKSTA"
            .EVEN
            .WORD      CHKSST
               .
               .
               .
            .END
```

Each ALR should start with the global address specified. For example:

```
;           THE INITIT ROUTINE
            .GLOBL     INITST
INITST:            ;Start of routine
               .
               .
               .
```

You should use this alternative method when you are adding an ALR written for FORTRAN IV to BASIC-11.

All the examples in this chapter use the recommended method (where the routine name packet is at the start of the routine).

Once you have defined the name and starting address of the routine, you can write the routine itself. The ALR can use the stack but it must ensure that the stack limit is not exceeded. BASIC-11 puts the stack limit in R4 before transferring control to the ALR. If you use any of the mathematical operations or function routines provided by BASIC-11, ensure that enough free space is available on the stack before executing the routine (15 free words for single-precision routines and 30 free words for double-precision routines). The ALR must end with an RTS PC instruction with the stack unchanged from its original state. The format of the INITIT routine is:

```
     ;     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 ACCESSING THE ARGUMENTS - THE ARGUMENT LISTS

When BASIC-11 executes the CALL statement, it evaluates the arguments and provides the routine with two lists. One contains pointers to the evaluated arguments and the other contains descriptors of the argument types. An assembly language routine (ALR) should ensure that the list contains the expected number and the right type of arguments.

Argument checking ensures that errors in a BASIC-11 program will not cause a fatal error in the ALR or in BASIC-11 itself. If no argument checking is done and a CALL statement contains an incorrect data type, the ALR produces unpredictable results. For example, if the ALR expects an integer array and the CALL statement contains a string expression, the ALR could overwrite sections of the stack. If the ALR checks arguments for errors, it can protect itself from errors in BASIC-11 programs. (There is no protection from errors in the ALR itself.)

A FORTRAN IV-compatible ALR cannot check arguments unless it first checks and determines that the language calling it is BASIC-11, because FORTRAN IV does not provide an argument descriptor list.

Before BASIC-11 transfers control to the ALR, it evaluates the arguments in the CALL statement. It creates a list of pointers to the arguments and a list of argument descriptors. Figure 4-2 shows the argument descriptor lists that BASIC-11 creates before it transfers control to the ALR.

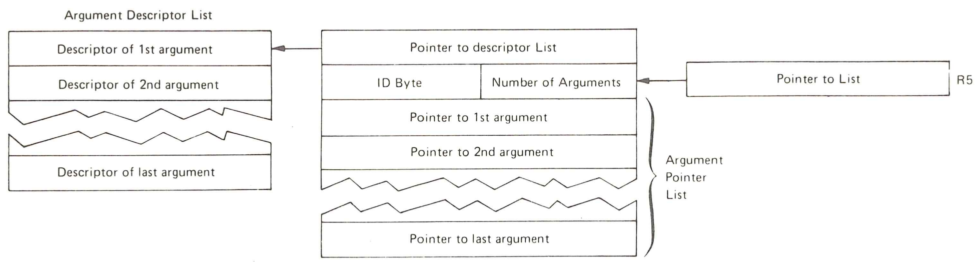

Figure 4-2 Assembly Language Routine Argument Lists

As shown in Figure 4-2, R5 points to a word that specifies the number of arguments in the CALL statement and identifies the language calling the ALR. The argument pointer list starts at the next word and the pointer to the argument descriptor list is stored in the previous word.

Each byte of the word pointed to by R5 is meaningful. The low-order byte contains the number of arguments. The high-order byte identifies the language. If the calling language is BASIC-11, the high-order byte has a value of 2Ø2. If the calling language is FORTRAN IV, the high-order byte has a value of Ø.

The pointers in the argument pointer list specify the location of the evaluated arguments. There are two exceptions, pointers for null arguments and pointers for string array arguments.

If an argument is null then its pointer does not point to that argument but instead contains a value of -1. A CALL statement argument list with two adjacent commas or a terminating command produces a null argument. For example, CALL "INITIT" (A, B,, D,) produces the following arguments: A, B, null, D, and null.

If the argument is a string array, then the pointer does not point to that argument but instead contains a value needed to access the string array (see Section 4.3.2). If the argument is an unsubscripted string or an element of a string array, the pointer specifies the location of the first character of the string.

The argument descriptor list specifies the data type of each argument. It also indicates whether the argument is an array or not and whether the ALR can return a result in the argument.

BASIC-11 provides additional information for strings and arrays. In these cases the word in the argument descriptor list is a pointer to the descriptor word, which has the additional information after it. Figure 4-3 describes the format of the descriptor word. BASIC-11 indicates if a word in the list is a pointer or a descriptor word by the value of the Ø bit. If the Ø bit is clear, then the word in the descriptor list is a pointer. If the Ø bit is set, then the word in the descriptor list is the descriptor word. Note that the descriptor word for strings and arrays has a value of Ø in the Ø bit.

NOTE

> All numbers in this chapter that specify the contents of a word or a section of a word are octal numbers.

15 14 13 — Array

12 11 10 9 8 — Size of Data Type

7 — Read Only

6 5 4 3 2 1 — Data Type

0 — More Than One Word Descriptor

Argument Type	Contents of Bits 15-13
Single value	0
Array	1
Null	0

Argument Type	Contents of Bit 7
Variable	0
Expression (Read Only)	1
Null	1

Argument Type	Contents of Bit 0
Numeric Scalar	1
Numeric Array	0
String Scalar	0
String Array	0
Null	1

Argument Type	Contents of Bits 12-8
Integer	2
Single Precision Floating Point Number	4
Double Precision Floating Point Number	10
String	2
Null Argument	0

Argument Type	Contents of Bits 6-1
Integer	11
Single Precision Floating Point Number	20
Double Precision Floating Point Number	21
String	40
Null Argument	77

Figure 4-3 Format of the Argument Descriptor Word

The ALR can return arguments only to variables and arrays. If the argument is an expression, constant, or element of a virtual array, the seventh bit of the argument descriptor word is set and the ALR must not return a value to that argument.

Bits 12 through 8 of the argument descriptor word specify the size of the data type. The ALR does not need to check this information because each argument type -- specified in bits 6 through 1 -- has a fixed size. The contents of bits 12 through 8 for a string argument can be ignored.

BASIC-11 provides additional information for array and string arguments. BASIC-11 specifies the total number of bytes in the array, the number of subscripts, the high limit of the first subscript, and the high limit of the second subscript if there are two subscripts. BASIC-11 also provides a string reference pointer for string arguments. This pointer is used by routines provided by BASIC-11 to access the string arguments. See Section 4.3.2 for a description of these routines. Figure 4-4 describes the format of array and string descriptors.

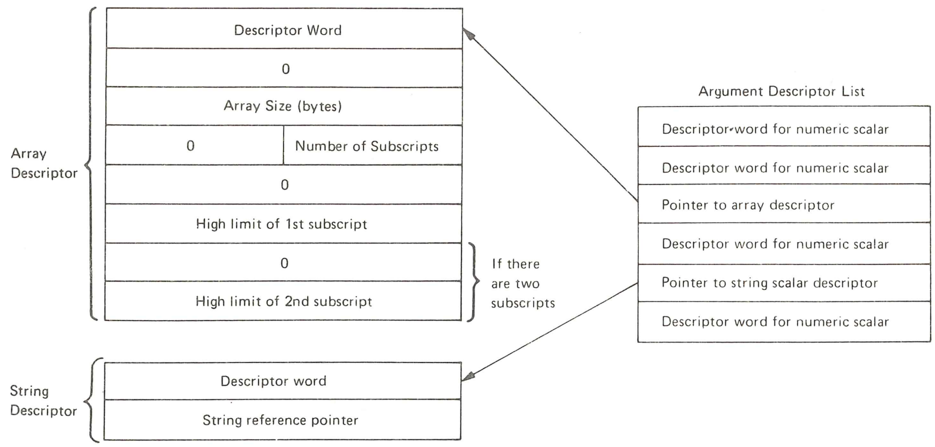

Figure 4-4 Format of Array and String Argument Descriptors

4.3.1 Numeric Arrays

If the CALL statement specifies an element of a numeric array, for example, A (1Ø), BASIC-11 considers it a one-dimensional array starting with the specified element and ending with the last element of the array. BASIC-11 considers it a one-dimensional array even if the entire array is two-dimensional.

BASIC-11 and FORTRAN IV store arrays differently. BASIC-11 array subscripts start at Ø, but FORTRAN IV array subscripts start at 1. In BASIC-11 arrays, the second subscript varies faster, but in FORTRAN IV arrays the first subscript varies faster. If you are designing a routine to be called from either BASIC-11 or FORTRAN IV, you must consider these differences in the ALR.

4.3.2 Strings and String Arrays

This section describes the routines BASIC-11 provides to allow the assembly language routine (ALR) to access strings. It also describes some example routines which use these string access routines. BASIC-11 allows dynamic-length strings, whose length can change during program execution. The BASIC-11 string access routines keep track of the location and size of strings. Consequently, an ALR cannot change a BASIC-11 string without using the string access routines.

The procedures for accessing strings and for accessing elements of string arrays are different. Note that if the CALL statement specifies an element of a string array (for example, A$(1Ø)), BASIC-11 considers it a string scalar. Only if the entire array is passed (for example, A$()), does BASIC-11 consider it a string array.

The ALR must locate and retrieve the string reference pointer word and pass it to the string access routines. For a string argument, the string reference pointer is the word following the descriptor word. For a string array argument, The ALR must calculate the string reference pointer to access any element of the array. The string reference pointer is a word whose value is determined by the following formula:

string reference pointer=2*offset+argument pointer

where:

offset	is the position of the element in the array.
argument pointer	is the value for the string array in the list of argument pointers. (Note that the argument pointer for a string array does not point to the argument itself.)

The offset for an element of a one-dimensional array is equal to the value of its subscript. The offset for an element of a two-dimensional array is defined by this formula:

offset=subscript1*(maximum value of subscript2+1)+subscript2

For example, consider two arrays -- A$(1Ø) and B$(3,5) -- with argument pointers of A and B, respectively. NOTE: All numbers in the following list are decimal.

Element	2*offset+argument pointer	string reference pointer
A$(Ø)	2*Ø+A	A
A$(4)	2*4+A	8+A
B$(Ø,5)	2*(Ø*6+5)+B	1Ø+B
B$(1,5)	2*(1*6+5)+B	22+B
B$(2,Ø)	2*(2*6+Ø)+B	24+B

The string access routines use the string reference pointer that the ALR provides to find and manipulate the string.

BASIC-11 provides four string access routines:

$FIND
$ALC
$STORE
$DEALC

The $FIND routine returns the length of a string and a pointer to the first character. The $ALC routine allocates a temporary string. An ALR can only write characters directly to strings created by $ALC. The $STORE routine assigns the value of one string to a second string and changes the first string to a null string. The $DEALC routine deallocates space used by the temporary string on the stack.

The ALR should use the following general procedure to manipulate a string argument and then return the resultant string. First, the ALR accesses the string argument by using the $FIND routine. Then it creates a temporary string with the $ALC routine. It then reads the characters of the string argument, manipulates them in the desired way, and writes the characters out to the temporary string. After this the ALR uses the $STORE routine to copy the temporary string to a string argument, which can be the original string. Finally, it uses the $DEALC routine to remove data placed on the stack by the $ALC routine.

Table 4-1, "Using String Access Routines", describes the four string access routines. The table describes the initial setup, including the format of the subroutine jump (JSR) instruction required to invoke the string access routine. It also describes the expected results and how to interpret them, that is, it indicates how to determine whether or not you made a correct initial setup in preparation for the string access routine.

If the ALR calls $FIND, $ALC, $STORE, and $ALC, it must specify them as global locations.

Before calling any of these routines, you must ensure that R5 contains its initial value, the value it had when BASIC-11 transferred control to the ALR. That is, R5 must point to the word identifying BASIC-11 and specifying the number of arguments.

NOTE

> These routines require that a register contain the same value in bits 6-1 as an argument descriptor word for a string argument. You can ensure this by moving a value of 100 into the specified register (puts a value of 40 in bits 6-1) or by moving an argument descriptor word in the specifed register.

4.4 USING ROUTINES PROVIDED BY BASIC-11

BASIC-11 provides routines that handle error conditions, print messages on the terminal, and perform mathematical operations and functions.

4.4.1 Error Handling and Message Routines

BASIC-11 provides two error handling routines, $ARGER and $BOMB, and two message printing routines, $MSG and $CHROT. The $ARGER routine produces the fatal ?ARGUMENT ERROR (?ARG) message. The ALR should call $ARGER when it detects an incorrect argument. The $BOMB routine allows the ALR to specify its own fatal message. The $MSG routine prints any message on the terminal and then returns control to the ALR. The $CHROT routine prints any single character on the terminal and then returns control to the ALR.

Table 4-1
Using String Access Routines

Routine	Program Setup	Result with No Errors Detected	Result with Errors Detected
$FIND (return location and length of string)	RØ<-string reference pointer R1<-1ØØ R5<-initial value Execute: JSR PC, $FIND	RØ = address of first string character R1 = length of string R2 = 1ØØ R3,R4,R5 unchanged C-bit = Ø (char) Z-bit = if a null string (R1=Ø)	RØ contains error code: if RØ=1, R1 did not equal 1ØØ if RØ=2, R5 did not contain correct initial value R3,R4,R5 unchanged C-bit = 1
$ALC (allocate temporary string)*	RØ<-required string length R1<-1ØØ R5<-initial value Execute: JSR PC, $ALC	RØ = address of first string character R1 = length of string R2 = 1ØØ R3,R4,R5 unchanged C-bit = Ø (char) Z-bit = 1 if a null string (R1=Ø) SP = string reference pointer stack contains several words of internal pointers. Remove these words from the stack by the $DEALC routine	RØ contains error code: if RØ=Ø, indicates insufficient free space for requested string if RØ=1, R1 did not equal 1ØØ if RØ=2, R5 did not contain correct initial value R3,R4,R5 unchanged C-bit = 1
$STORE (store value of a string in a second string, make first string null)	RØ<-string reference pointer of string to be copied R1<-string reference pointer of receiving string R2<-1ØØ R5<-initial value Execute: JSR PC, $STORE	RØ,R1,R2,R3,R4,R5 unchanged C-bit = Ø string whose pointer was in RØ is null string whose pointer was in R1 contains former value of the other string	RØ contains error code: if RØ=1, R2 did not equal 1ØØ if RØ=2, R5 did not contain correct initial value R1,R2,R3,R4,R5 unchanged C-bit = 1
$DEALC (remove from stack the internal pointers produced by $ALC routine)*	Return stack to the state that it was immediately following $ALC routine. Do this by removing any words you have added to the stack since calling the $ALC routine; this ensures that the string reference pointer is in the SP. R2<-1ØØ R5<-initial value Execute: JSR PC, $DEALC	RØ,R1,R2,R3,R4,R5 unchanged C-bit = Ø Stack return to the state that existed before $ALC was called	RØ contains error code: if RØ=1, R2 did not equal 1ØØ if RØ=2, R5 did not contain correct initial value R1,R2,R3,R4,R5 unchanged C-bit = 1 Stack

* Any temporary string created by $ALC must be removed by $DEALC before the ALR ends.

If the ALR calls $ARGER, $BOMB, $MSG, or $CHROT, it must specify them as global locations.

Call the $ARGER routine by executing the instruction:

```
JMP $ARGER
```

The $ARGER routine prints the error message on the terminal in one of the following formats:

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

where:

xxxxx   is the line number of the CALL statement.

If the CALL statement was an immediate mode statement, then AT LINE xxxxx is not printed. Control then returns to BASIC-11, which prints the READY message.

Call the $BOMB routine by executing the following instruction:

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

where:

message   is the string of characters that you wish to print.

The $BOMB routine prints the error message on the terminal in the form:

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

where:

xxxxx   is the line number of the CALL statement.

If the CALL statement was an immediate mode statement, then AT LINE xxxxx is not printed. Control then returns to BASIC-11, which prints the READY message.

Call the $MSG routine by executing the instruction:

```
JSR  R1,$MSG
.ASCII 'message'
.BYTE 15,12,Ø          ;Must have carriage return
.EVEN                  ;and line feed and end with Ø
```

where:

message   is the string of characters that you wish to print.

The $MSG routine prints the message you specify on the terminal, and then returns control to the instruction that follows the .EVEN instruction.

Call the $CHROT routine as follows:

1. put the 8-bit ASCII code of the character in the low order byte of RØ

2. execute the instruction:

```
JSR  PC,$CHROT
```

$CHROT prints the character specified in RØ on the terminal, and then returns control to the ALR.

4.4.2 Mathematical Operation and Function Routines

Assembly language routines (ALRs) can use BASIC-11's mathematical operation and function routine to perform operations and functions that you can use in a BASIC-11 program. ALRs can use the same routine that BASIC-11 uses to perform these operations and functions. An advantage of this is that the ALR need not duplicate routines that already exist in BASIC-11.

NOTE

Assembly language routines that use the FP11 Floating Point unit are required to save and restore the FPU status. If the assembly language routine will modify the FPU status, it must preserve the FPU status on entry by executing the following instruction:

```
STFPS     -(SP)
```

and restore the status (prior to returning to the calling program) by executing the instruction:

```
LDFPS     (SP)+
```

Tables 4-2 and 4-3 describe the BASIC-11 mathematical operations and functions. They show how each operation or function appears in the BASIC-11 language, and name the BASIC-11 routine that performs it. Note that certain operations and functions require one routine for single-precision arithmetic, a different routine for double-precision arithmetic, and yet another for integer arithmetic.

Table 4-2
BASIC-11 Mathematical Operations

Operation	Operator	Meaning	BASIC Equivalent	Single-Precision Routine	Double-Precision Routine
Addition	+	Adds two floating-point numbers	C=A + B	$ADR	$ADD
Subtraction	-	Subtracts one floating-point number from another	C=A - B	$SBR	$SBD
Multiplication	*	Multiplies two floating-point numbers	C=A * B	$MLR	$MLD
		Multiplies two integers	C%=A%*B%	$MLI	$MLI
Division	/	Divides one floating-point number by another	C=A / B	$DVR	$DVD
		Divides one integer by another integer	C%=A%/B%	SDVI	SDVI
Exponentiation	^	Raises a floating-point number by a floating-point exponent.	C=A ^ B	XFF$	XDD$
		Raises a floating-point number by an integer exponent.	C=A B%	XFI$	XDI$
		Raises an integer by an integer exponent.	C%=A%^B%	XII$	XII$

Table 4-3
BASIC-11 Mathematical Functions

Function	Description	BASIC Equivalent	Single-Precision Routine	Double-Precision Routine
Data type conversion	Converts floating-point number to integer	B% = A	$RI	$DI
	Converts integer to floating	B = A%	$IR	$ID
Truncation	Truncates a floating-point number to a floating-point whole number	B=SGN(A)* INT(ABS(A))	$INTR	$DINT
Sine	Finds the sine of a radian value	B=SIN(A)	SIN	DSIN
Cosine	Finds the cosine of a radian value	B-COS(A)	COS	DCOS
	Finds the arctangent in radians of a number	B=ATN(A)	ATAN	DATAN
Logarithm	Finds the natural log (base e) of a number	B=LOG(A)	ALOG	DLOG
	Finds the common log (base 1Ø) of a number	B=LOG1Ø(A)	ALDG1Ø	DLOG1Ø
Square root	Finds the square root of a number	B=SQR(A)	SQRT	DSQRT
Exponential	Finds the value of e raised to a number	B=EXP(A)	EXP	DEXP

If you are running a BASIC-11 system designed for double-precision arithmetic, either the single- or the double-precision routine names can be used. Either routine name will execute the double-precision routine; this fact allows you to use the same code for different systems regardless of precision. However, you must still be aware of which precision you are using, and ensure that the data manipulations in the program properly reflect the BASIC-11 configuration on which programs are running. To be compatible with FORTRAN IV you must use only the double-precision routine names to execute the double-precision routines.

All routines that have a dollar sign ($) in their name must be called in threaded code mode. To call routines in threaded code mode, first call a special subroutine, $POLSH. After calling $POLSH, list the names of the threaded code routines you wish to call. In threaded code mode, each routine is executed in the order listed. All arguments and results are passed on the stack. Finally, list the name of a second special subroutine, $UNPOL, which ends threaded code mode.

You must specify $POLSH, $UNPOL and any routine names you specify as globals.

The call to $POLSH is in the following format:

```
     JSR    R4,$POLSH
```

Figure 4-5 describes the state of the stack before and after each threaded code routine.

As examples, consider the following segments of routines:

Segment 1 divides an integer stored in TEMP1 by an integer stored in TEMP2 and stores the quotient in 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    Ø
TEMP2   .WORD    Ø
RESULT  .WORD    Ø
```

Routine Names	Equivalent BASIC Statement	Stack Before Execution	Stack After Execution
$ADR $SBR $MLR $DVR	C=A+B C=A-B C=A*B C=A/B	B { High-order FP word ← Stack pointer B { Low-order FP word A { High-order FP word A { Low-order FP word	C { High-order FP word ← Stack pointer C { Low-order FP word
$ADD $SBD $MLD $DVD XDD$	C=A+B C=A-B C=A*B C=A/B C=A^B	B { High-order FP word ← Stack pointer B { Low-order FP word B { Lower-order FP word B { Lowest-order FP word A { High-order FP word A { Low-order FP word A { Lower-order FP word A { Lowest-order FP word	C { High-order FP word ← Stack pointer C { Low-order FP word C { Lower-order FP word C { Lowest-order FP word
$MLI $DVI XII$	C%=A%*B% C%=A%/B% C%=A%^B%	B% Integer word ← Stack pointer A% Integer word	C% Integer word ← Stack pointer
XFI$	C-A^B%	B% Integer word ← Stack pointer A { High-order word A { Low-order FP word	C { High-order FP word ← Stack pointer C { Low-order FP word
XDI$	C=A^B%	B% Integer word ← Stack pointer A { High-order FP word A { Low-order FP word A { Lower-order FP word A { Lowest-order FP word	C { High-order FP word ← Stack pointer C { Low-order FP word C { Lower-order FP word C { Lowest-order FP word
$RI	B%=A	A { High-order FP word ← Stack pointer A { Low-order FP word	B% Integer word ← Stack pointer
$DI	B%=A	A { High-order FP word ← Stack pointer A { Low-order FP word A { Lower-order FP word A { Lowest-order FP word	B% Integer word ← Stack pointer
$IR	B=A%	A% Integer word ← Stack pointer	B { High-order FP word ← Stack pointer B { Low-order FP word
$ID	B=A%	A% Integer word ← Stack pointer	B { High-order FP word ← Stack pointer B { Low-order FP word B { Lower-order FP word B { Lowest-order FP word
$INTR	B=SGN(A)* INT(ABS(A))	A { High-order FP word ← Stack pointer A { Low-order FP word	B { High-order FP word ← Stack pointer B { Low-order FP word
$DINT	B=SGN(A)* INT(ABS(A))	A { High-order FP word ← Stack pointer A { Low-order FP word A { Lower-order FP word A { Lowest-order FP word	B { High-order FP word ← Stack pointer B { Low-order FP word B { Lower-order FP word B { Lowest-order FP word

Note: FP stands for Floating Point

Figure 4-5 State of Stack for Threaded Code Routines

Segment 2 multiplies two single-precision floating-point numbers, FLOATA and FLOATB, and stores the product in 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 mode
        MOV      (SP)+,FLOATC          ;Store result
        MOV      (SP)+,FLOATC+2        ;in FLOATC
            .
            .
            .
FLOATA: .WORD    Ø,Ø
FLOATB: .WORD    Ø,Ø
FLOATC: .WORD    Ø,Ø
```

Segment 3 converts a double-precision floating-point number stored at FLOAT to an integer and stores it at INTMDW.

```
;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 mode
        .WORD    $DI                   ;Specify routine name
        .WORD    $UNPOL                ;Leave threaded code mode
        MOV      (SP)+,INTMDW          ;Store result
            .
            .
            .
FLOAT:  .WORD    Ø,Ø,Ø,Ø
INTMDW: .WORD    Ø
```

Although the foregoing examples have only one routine name after each call to $POLSH, you can specify any number of routine names. You must always follow the last of routine name with the $UNPOL routine.

The sine, cosine, arctangent, logarithm, square root, and exponential routines each use an argument list similar to the BASIC-11 CALL argument list. An ALR must establish the argument list before calling the routine. The format of the argument list for the single-precision routines, SIN, COS, ATAN, ALOG, ALOG1Ø, SQRT, and EXP, is:

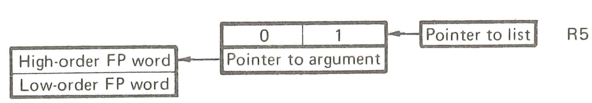

Figure 4-6 Argument List for Supplied Single-Precision Routines

The format of the argument list for the double-precision routines, DSIN, DCOS, DATAN, DLOG, DLOG1Ø, DSQRT, and DEXP is:

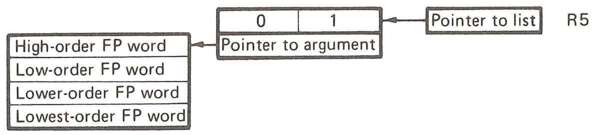

Figure 4-7 Argument List for Supplied Double-Precision Routines

In both cases, the routines are called by the instruction:

```
JSR    PC, routine name
```

The single-precision routines return the result in RØ and R1; the high-order word is in RØ and the low-order word is in R1.

The double-precision routines return the result in RØ, R1, R2, and R3. The high-order word is in RØ and the low-, lower-, and lowest-order words are in R1, R2, and R3, respectively.

You must specify as global any routine name that you call.

These routines do not preserve any registers.

NOTE

You should save the initial value of R5 before loading the pointer to the argument for these routines. You will need the saved value to execute any threaded code routine to access arguments.

Consider the following segment of a routine that finds the square root of a single-precision floating-point number, NUM1, and stores the result in NUM2:

```
;Segment which finds square root

        .GLOBL   SQRT
        MOV      R5, TEMP5          ;Save old value of R5
        MOV      R1, TEMP1          ;Save any other register
        MOV      RØ, TEMPØ
        MOV      #ARG, R5           ;Set up R5
        JSR      PC, SQRT           ;Call routine
        MOV      RØ, NUM2           ;Store high order result
        MOV      R1, NUM2+2         ;Store low order result
        MOV      TEMP5, R5          ;Restore saved
        MOV      TEMP1, R1          ;Registers
        MOV      TEMPØ, RØ
            .
            .
            .
```

INDEX

HOW TO ORDER ADDITIONAL DOCUMENTATION

From	Call	Write
Chicago	312–640–5612 8:15 A.M. to 5:00 P.M. CT	Digital Equipment Corporation Accessories & Supplies Center 1050 East Remington Road Schaumburg, IL 60195
San Francisco	408–734–4915 8:15 A.M. to 5:00 P.M. PT	Digital Equipment Corporation Accessories & Supplies Center 632 Caribbean Drive Sunnyvale, CA 94086
Alaska, Hawaii	603–884–6660 8:30 A.M. to 6:00 P.M. ET *or* 408–734–4915 8:15 A.M. to 5:00 P.M. PT	
New Hampshire	603–884–6660 8:30 A.M. to 6:00 P.M. ET	Digital Equipment Corporation Accessories & Supplies Center P.O. Box CS2008 Nashua, NH 03061
Rest of U.S.A., Puerto Rico*	1–800–258–1710 8:30 A.M. to 6:00 P.M. ET	

*Prepaid orders from Puerto Rico must be placed with the local DIGITAL subsidiary (call 809–754–7575)

From	Call	Write
Canada		
British Columbia	1–800–267–6146 8:00 A.M. to 5:00 P.M. ET	Digital Equipment of Canada Ltd 940 Belfast Road Ottawa, Ontario K1G 4C2 Attn: A&SG Business Manager
Ottawa–Hull	613–234–7726 8:00 A.M. to 5:00 P.M. ET	
Elsewhere	112–800–267–6146 8:00 A.M. to 5:00 P.M. ET	
Elsewhere		Digital Equipment Corporation A&SG Business Manager*

*c/o DIGITAL's local subsidiary or approved distributor

BASIC–11/RT–11
User's Guide
AA–5071B–TC

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

— Assembly language programmer
— Higher-level language programmer
— Occasional programmer (experienced)
— User with little programming experience
— Student programmer
— Other (please specify) ____________

Name ____________ Date ____________

Organization ____________ Telephone ____________

Street ____________

City ____________ State ____________ Zip Code or Country ____________

Do Not Tear — Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SSG/ML PUBLICATIONS, ML05–5/E45
DIGITAL EQUIPMENT CORPORATION
146 MAIN STREET
MAYNARD, MA 01754

Do Not Tear — Fold Here

Cut Along Dotted Line

Printed in U.S.A.