pdp11

IAS/RSX BASIC User's Guide

Order No. DEC-11-LIBIA-B-D

digital

First Printing, September 1976

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

5/82-15

CONTENTS

CONTENTS (Cont.)

TABLES

PREFACE

Most features of IAS/RSX BASIC are the same as those in other versions of BASIC-11. (DIGITAL's name for a family of BASICs for the PDP-11). These features are described in the BASIC-11 Language Reference Manual (DEC-11-LIBBB-A-D). Please read the BASIC-11 Language Reference Manual before reading this User's Guide.

This guide describes the system dependent features of IAS/RSX BASIC. They are:

- Procedure for starting BASIC
- 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

This guide (except Chapter 4) should be read by and be available for reference by all users of IAS/RSX BASIC. Chapter 4, the description of writing assembly language routines should be read only by users who are interfacing assembly language routines with BASIC.

This guide assumes that you are familiar with the use of and the login procedure (if any) of your system. These are described in the following manuals (for RSX-11D, RSX-11M, and IAS users respectively):

RSX-11D User's Guide (DEC-11-OXDUA-B-D)
RSX-11M Operator's Procedures Manual (DEC-11-OMOGA-B-D)
IAS User's Guide (DEC-11-OIUGA-A-D)

The IAS login procedure is summarized in this manual for your convenience.

This guide (except Chapter 4) also assumes that BASIC has been installed on your system. If BASIC has not been installed on your system, see the IAS/RSX BASIC Installation Guide (DEC-11-LIBJA-B-D).

Chapter 4 assumes that you are an experienced MACRO-11 programmer and that you will install BASIC once you have incorporated the routines into BASIC.

DOCUMENTATION CONVENTIONS

These are the documentation conventions which are used throughout this manual.

The following symbols have special meaning.

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).
RUBOUT	Type the RUBOUT key (DELETE 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 $\left\{\begin{array}{l}\text{THEN statement}\\ \text{THEN line number}\\ \text{GO TO line number}\end{array}\right\}$
...	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 lower-case letters	Replace these elements according to the description provided in text. See below for list of commonly used lower-case items.

This list describes some lower case 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.

Lower-Case item	Abbreviation	Meaning
expression	exp	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 in the format 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 was put after it. For example: 5%, 3%, 2, 7
line number	___	Any 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 there is more than one lower-case word in a format, the words are numbered 1, 2, 3, etc. For example:

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

Throughout this manual, the term BASIC means BASIC-11 or IAS/RSX BASIC.

To differentiate between what BASIC prints and what you type, the user type-in is printed in red ink. For example:

```
RUNNH

WHAT NUMBERS?  5,10
THE SUM IS 15

READY
```

All user type-in is terminated by the RETURN key unless the text indicates a different terminator.

CHAPTER 1

GETTING STARTED WITH IAS/RSX BASIC

1.1 STARTING BASIC

You can use BASIC on an RSX-11D (Resource Sharing Executive), RSX-11M, or IAS (Interactive Applications System) system. You must login the system (if your system has a login procedure) before starting BASIC.

1.1.1 Starting BASIC on IAS

The Program Development system (PDS) provides the standard IAS interface to the computer. The installation's IAS system manager determines who may use PDS and decides which terminals will support it.

To invoke PDS, type CTRL/C. The system responds to CTRL/C by displaying an identifying message, the current time and date, and the PDS> prompt. Type the LOGIN command:

```
          IAS  PROGRAM DEVELOPMENT SYSTEM   VERSION 1.1
                 14:06:03          27-JUL-76

PDS> LOGIN
```

After you enter the LOGIN command, PDS prints

```
USERID?
```

Type your user name

NOTE

The system manager assigns each user a user name and a password, which are then registered with IAS. If you do not have a user name and a password consult your system manager.

After you enter your user name, PDS prints

```
PASSWORD?
```

Type your password. PDS does not print the characters you type.

If the password given is incorrect, PDS prompts PASSWORD? again.

When you enter the correct password, IAS displays some identifying information. For example:

```
USER B11   UIC [200,100] TT07:   TASK 102  14:06:49 27-JUL-76
IAS V01.1        4-JUN-76        <SY IS DB2>

PDS>
```

The numbers enclosed in square brackets, in this example [200,100], are the user file directory or ufd (see Section 2.1).

To start BASIC, type the BASIC command, which has no parameters. PDS starts BASIC and BASIC prints an identifying message.

```
PDS> BASIC

IAS/RSX BASIC V02-01

READY
```

The READY message indicates that BASIC is ready to accept program lines, commands, or immediate mode statements.

1.1.2 Starting BASIC on RSX-11D and RSX-11M

To start BASIC on RSX-11D or RSX-11M, you must first invoke the MCR (Monitor Control Routine). See the RSX-11D User's Guide or the RSX-11M Operator's Procedures Manual for information on invoking MCR. After MCR has printed the MCR> prompt, type:

```
MCR>BASIC
```

(You can also abbreviate BASIC by just typing BAS.) BASIC then prints an identifying message and the READY message:

```
IAS/RSX  BASIC  V02-01

READY
```

The READY message indicates that BASIC is ready to accept program lines, commands, or immediate mode statements.

1.2 USING CTRL/C TO STOP BASIC PROGRAMS

When you want to stop execution of a BASIC program, use CTRL/C. However, CTRL/C has a different effect on IAS systems than on RSX-11D and RSX-11M systems.

1.2.1 Using CTRL/C on IAS

On IAS systems, typing CTRL/C causes BASIC to stop execution of the program and print the STOP and READY messages:

```
STOP AT LINE xxxxx
READY
```

Where xxxxx is the line number of the last BASIC statement executed. (If BASIC is not executing a program, it does not print AT LINE xxxxx.)

1.2.2 Using CTRL/C on RSX-11D and RSX-11M

On RSX-11D and RSX-11M, typing CTRL/C causes the system to suspend BASIC and return control to the MCR which prints its prompt message. After typing CTRL/C, to return control to BASIC, type:

```
     MCR>STOP(ESC)
```

BASIC then prints the STOP and READY messages:

```
     STOP AT LINE xxxxx
     READY
```

Where xxxxx is the line number of the last BASIC statement executed. (If BASIC is not executing a program it does not print AT LINE xxxxx.)

There is an exception to the CTRL/C procedure for RSX-11D and RSX-11M. If BASIC is executing an INPUT statement when you type CTRL/C, you must also type the ESCAPE key. Then type STOP and the ESCAPE key. Instead of printing the STOP message, BASIC prints a question mark (?) to request the original input. Type the ESCAPE key, which cancels the request. Then BASIC prints the STOP and READY messages. An example of this exception follows:

```
     ? (CTRL/C)(ESC)
     MCR>STOP(ESC)
     ? (ESC)
     STOP AT LINE 100

     READY
```

NOTE

> If you type STOP(RET) instead of STOP(ESC), BASIC prints the ?SYNTAX ERROR message in response to anything that you type subsequently. If this happens, type the ESCAPE key as many times as it is necessary for BASIC to respond with a READY message (with no error message).

1.3 TERMINATING THE SESSION (BYE COMMAND)

To terminate a session with BASIC, use the BYE command. Simply type:

```
     BYE
```

This terminates BASIC and returns control to MCR (for RSX-11D and RSX-11M systems) or to PDS (for IAS systems). If you logged in, you must log out. The BYE command only terminates your session with BASIC.

On IAS systems, to log out, type

```
     PDS> LOGOUT
```

PDS prints a terminating message. After you enter the LOGOUT command, you must type CTRL/C and the LOGIN command to use the terminal again.

1.4 PRECISION OF STORED NUMBERS

IAS/RSX BASIC stores numbers to 7 digits of accuracy. Repeated mathematical operations can magnify the imprecision. BASIC only prints six digits unless you use the PRINT USING statement (see the BASIC-11 Language Reference Manual). All BASIC functions maintain 7 digits of accuracy except the SIN and COS functions which have a reduced accuracy of 5 1/2 digits in certain ranges.

1.5 SYSTEM DEPENDENT ERROR MESSAGES

IAS/RSX BASIC prints error messages in the long format (see the BASIC-11 Language Reference Manual). The following lists error messages with system dependent descriptions:

?CANNOT DELETE FILE

Attempt to delete file is not allowed by the file privilege system.

?FILE PRIVILEGE VIOLATION

Access of file is not allowed by the file privilege system.

?FILE ALREADY EXISTS

NAME statement attempts to change a file to a new file specification which is the same as an existing file (including the version)

?NOT ENOUGH ROOM

This message is not produced by IAS/RSX BASIC. If there is not enough room on a device, BASIC prints the ?CHANNEL I/O ERROR message.

CHAPTER 2

FILES

2.1 FILE SPECIFICATION

IAS/RSX BASIC uses the same file specification format as the RSX-11D, RSX-11M, and IAS systems. This section summarizes the file specification and describes the default values used by IAS/RSX BASIC.

The format of the file specification is:

```
dev:[ufd]name.ext;ver
```

where:

dev:	is a device name of the form XXnn: where XX is a 2-letter mnemonic for the device, and nn is the 1- or 2-digit unit number from 0 to 77.
[ufd]	is the user file directory of the form [m,n] where m and n are octal numbers in the range 1 to 377.
name	is the name of the file, an alphanumeric character string one to nine characters long. All alphabetic characters must be upper-case letters.
.ext	is the file extension, an alphanumeric string from one to three characters long, beginning with a letter. The extension usually identifies some aspect of the file contents. All alphabetic characters must be upper-case letters.
;ver	is the version number, an octal number in the range 1 to 77777 used to differentiate among versions of the same file.

If you do not specify any of the elements of the file specification, IAS/RSX BASIC uses a default value:

The default device (dev:) is SY0:.

The default user file directory ([ufd]) is the system's default value.

The default name depends on how the file specification is being used. If you do not specify the name in a SAVE, REPLACE, or COMPILE command, BASIC assumes that the current program name is the name. If you do not specify the name in an OLD or APPEND command or a CHAIN or OVERLAY statement, BASIC assumes the name NONAME. If you do not specify the name in any other command or statement (UNSAVE command and KILL and NAME statements), BASIC prints the ?ILLEGAL FILE SPECIFICATION message.

The default value for the extension (.ext) also depends on how the file specification is being used. If you do not specify the extension in a statement accessing a data file (OPEN, KILL, and NAME), BASIC assumes the extension .DAT. If you do not specify the extension in a RUN or OLD command or a CHAIN statement, BASIC first sees if the specified file exists with the extension .BAC. If that file exists, BASIC uses it. If the file does not exist, BASIC assumes the extension .BAS and looks for that file. If you do not specify the extension in the COMPILE command, BASIC assumes the extension .BAC. If you do not specify the extension in the SAVE, REPLACE, UNSAVE, APPEND, or CAT command or the OVERLAY statement, BASIC assumes the extension .BAS.

The default version number is the highest existing version number. However, when you are creating a file it assumes the highest existing version number plus 1.

If you specify either a ufd, an extension, or a version, you must also specify the file name.

If when creating a file with the SAVE command you specify a version that exists, BASIC prints the ?USE REPLACE error message.

If you specify LP: or LPnn: as the device, BASIC always spools the data to the line printer. (The data is first stored in a temporary disk file, then the file is automatically printed on the line printer, and, finally, the temporary file is deleted.) BASIC ignores any name, extension, or version that you specify and creates a temporary file with the current program name as the file name and .LST as the extension. If there is no line printer spooler on your system or if the spooler is not active, BASIC prints the ?ERROR CLOSING CHANNEL message and you must exit from BASIC to transfer the .LST file to the line printer.

Wild cards (*) can only be used in the CAT command (see section 2.3).

2.2 THE OPEN STATEMENT - SYSTEM DEPENDENT FEATURES

The format of the OPEN statement is:

```
OPEN string [{FOR INPUT }] AS FILE [#]expr[DOUBLE BUF][,RECORDSIZE expr][,MODE expr][,FILESIZE expr]
            [{FOR OUTPUT}]
```

where:

expr1	is the channel number of the file. It can be any BASIC expression and have a value between 1 and 12.
DOUBLE BUF	is ignored if specified.
,RECORDSIZE expr2	is ignored if specified.
,MODE expr3	is ignored if specified.
,FILESIZE expr4	is ignored if specified.

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

2.3 LISTING FILE DIRECTORIES (CAT COMMAND)

Use the CAT (catalog) command to find out what files exist on your system. You can list program files, data files, a specific file, all files, or files in another user's ufd. The form of the command is:

CAT[file specification]

where file specification is in the format described in section 2.1.

You can use an asterisk (*) to replace any element in the file specification except for the dev: or the [ufd]. This asterisk is called a wild card specification and indicates that you want all files that match the rest of the file specification ignoring the element(s) replaced with the wild card.

BASIC assumes a wild card for the version unless you specify a version.

BASIC assumes the extension .BAS if you do not specify any.

If you just type

```
CAT
```

BASIC lists all the BASIC programs in your ufd. This is equivalent to typing

```
CAT *.BAS;*
```

For example, to list all the BASIC programs in your area:, type:

```
CAT
BYECHK.BAS;1    OVRLY.BAS;3      RND.BAS;1
VAOPN1.BAS;2    ARRAY1.BAS;2     PART1.BAS;2
PART1.BAS;3     ARRA.BAS;1       VAOPN1.BAS;3
PROG1.BAS;2

READY
```

And to list all the compiled BASIC programs in ufd [11,104], type:

```
CAT [11,104]*.BAC;*
PROG.BAC;1      TIME.BAC;1       TIME.BAC;2
DECODE.BAC;3

READY
```

And to list all versions of the data files with the name MONEY, type:

```
CAT MONEY.DAT;*
MONEY.DAT;1     MONEY.DAT;2

READY
```

NOTE

You cannot interrupt the CAT command with CTRL/C. You must use CTRL/O to terminate the listing of the files.

2.4 EFFECT OF SUPERSEDING FILES

If you create a new file with the same file specification as an existing file and do not specify the version, BASIC does not delete the old file. Instead it creates a new file with a version number one greater than the highest existing version of that file. If you create a new file with an OPEN FOR OUTPUT statement or REPLACE command with the same file specification and same version number as an existing file, BASIC deletes the old file. If you attempt to do this with a SAVE command, BASIC prints the ?USE REPLACE (?RPL) error message and does not create a new file.

CHAPTER 3

UTILITY FUNCTIONS

3.1 UTILITY FUNCTIONS AVAILABLE ON IAS/RSX BASIC

IAS/RSX BAS 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 (SYS)
- Check if a CTRL/C has been typed (SYS)

3.2 SETTING THE TERMINAL MARGIN (TTYSET FUNCTION)

Use the TTYSET function to set your terminal's margin. BASIC prints on a line until a number or string would extend past the margin you set. BASIC 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	may be either a numeric constant (as specified in format) or an expression, but must have a value of 255 (for compatibility with other versions of BASIC).
expression	determines the terminal margin. The margin is set to the value of the expression minus 1. If the expression equals 0, then there is no change from the previous margin.

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

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

To set BASIC to print to the full width of a VT50 display terminal (80 columns), type:

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

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

Ensure that the system's margin for your terminal is the same as or greater than the margin you specify in TTYSET.

If the value of the expression is less than zero, one, or greater than 256, BASIC prints the ?ARGUMENT ERROR message. If the first argument has a value other than 255, BASIC prints the same message.

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

BASIC stops printing to the terminal when the CTRL/O key is typed; however, the RCTRLO function causes BASIC to resume printing. Use the RCTRLO function if you want to ensure that certain data is printed on the terminal even if a CTRL/O has been 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                                  Type CTRL/O here
     SUM= 4172

     READY
```

3.4 DISABLING CTRL/C FROM STOPPING BASIC PROGRAMS (RCTRLC AND CTRLC FUNCTIONS)

In certain situations you may not want to have your program interrupted by the CTRL/C key command. The RCTRLC function disables CTRL/C and prevents it from stopping the BASIC program. The CTRLC function enables the CTRL/C key command.

The format of the functions are:

[LET]variable=RCTRLC

[LET]variable=CTRLC

where:

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

After BASIC executes the RCTRLC function, typing CTRL/C on the terminal does not stop the program. On RSX-11D and RSX-11M, typing CTRL/C still suspends the task and returns control to MCR, and typing STOP ESC returns control to BASIC, but the program is not stopped.

After BASIC executes the CTRLC function, typing CTRL/C stops the program. BASIC does not save any CTRL/C that is typed while CTRL/C is disabled.

When BASIC prints the READY message, it automatically enables the CTRL/C key 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
MCR>STOP ESC
INTERRUPTS OKAY
CTRL/C
MCR>STOP ESC
STOP AT LINE 1130

READY
```

For information on a system function which 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. Do not disable CTRL/C until your program has been thoroughly debugged.

3.5 TERMINATING YOUR PROGRAM (ABORT FUNCTION)

If you want a program to delete itself when it terminates, use the ABORT function. The ABORT function is equivalent to an END statement except that ABORT can optionally delete your program 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 and has an undefined value after the statement is executed.
expression	determines if the program is to be deleted. If expression equals 0, BASIC does not delete the program. If expression equals 1, BASIC deletes it.

Consider the following examples:

```
        Delete when done                  Do not delete when done

LIST                                LIST

ABORT  21-JUN-76  14:52:45          ABORT  21-JUN-76  14:54:00

10 PRINT "123"                      10 PRINT "123"
20 A=ABORT(1)                       20 A=ABORT(0)
30 PRINT "456"                      30 PRINT "456"

READY                               READY
RUNNH                               RUNNH

123                                 123

READY                               READY
LIST                                LIST

NONAME  21-JUN-76  14:53:30         ABORT 21-JUN-76  14:54:30

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

                                    READY
```

3.6 SYSTEM FUNCTIONS

System functions perform several system-dependent operations. If you want to use a program that contains system functions on another system, you probably must change the program.

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 to print the ?ARGUMENT ERROR (?ARG) message.

Table 3-1
Summary of System Functions

Value of expression1	Function Performed
1	Single character input. Target variable contains the ASCII value of the next character typed at the terminal.
4	Terminates BASIC and returns control to system (equivalent to the BYE command).
6	Determines if CTRL/C has been typed while CTRL/C is disabled by RCTRLC function. Target variable equals 1 if CTRL/C has been typed and equals 0 if CTRL/C has not been typed.

3.6.1 Single Character Input (SYS(1) Function)

Use the single character input system function to process input one character at a time. By using SYS(1), you can create an interactive program where the user gets an instant response to each character typed.

When BASIC is executing SYS(1), it does not print on the terminal the characters that are typed on the keyboard (printing typed characters on the terminal is called echoing). Consequently, if you want the user of the program to see what is typed, the program must print each character that the user enters.

SYS(1) returns the 7-bit ASCII value of any character typed on the terminal except CTRL/C. (See the <u>BASIC-11 Language Reference Manual</u> for a list of the ASCII values.) If CTRL/C is typed when BASIC is executing SYS(1) and CTRL/C is enabled, then BASIC prints the STOP and READY messages (on IAS, RSX-11D, and RSX-11M). If CTRL/C is disabled, then BASIC continues executing SYS(1) and waits for another character. Typing CTRL/C while SYS(1) is being executed never suspends BASIC and returns control to MCR.

```
LISTNH
10 PRINT "TYPE A CHARACTER --";
20 A=SYS(1)
30 PRINT CHR$(A) \ REM ECHO THE CHARACTER
40 PRINT "THE ASCII VALUE OF ";CHR$(A);" IS";A

READY
RUNNH
TYPE A CHARACTER --Z
THE ASCII VALUE OF Z IS 90

READY
```

3.6.2 Terminating BASIC (SYS(4) Function)

If you want to terminate BASIC from a BASIC program, use system function SYS(4). It is equivalent in effect to the BYE Command.

For example:

```
10 PRINT "GOODBYE"                Print message.
20 A=SYS(4)                       Terminate BASIC and
                                  return control to MCR.
```

3.6.3 Checking for CTRL/C (SYS(6) Function)

If you have disabled CTRL/C with the RCTRLC function and want to check if a user has typed CTRL/C, 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                       Disable CTRL/C.
30 B=SYS(6)                       Check for CTRL/C.
40 IF B=1 THEN 100                If it has been typed, go to 100.
50 PRINT "STILL EXECUTING"
60 GO TO 30
100 PRINT "PROGRAM TERMINATING"
110 A=CTRLC                       Reenable CTRL/C.
120 A=ABORT(1)                    Delete program and return to
                                  READY.
READY
RUNNH

STILL EXECUTING
STILL EXECUTING
CTRL/C                            Type CTRL/C
MCR>STOP ESC
STILL EXECUTING
PROGRAM TERMINATING

READY
```

CHAPTER 4

WRITING ASSEMBLY LANGUAGE ROUTINES FOR BASIC

4.1 INTRODUCTION TO ASSEMBLY LANGUAGE ROUTINES

To perform an operation that is beyond BASIC's capabilities, you must write an assembly language routine that does what you want and then add the routine to BASIC. For example, you can write routines to perform mathematical operations, special device input and output, and array manipulation. Once these routines are written and added to BASIC, any BASIC programmer can execute or call the routine by means of the CALL statement. (See the BASIC-11 Language Reference Manual for a description of the CALL statement.)

An advantage of adding assembly language routines to BASIC is that a programmer familiar only with BASIC can use routines written by an assembly language programmer. For example, an assembly language programmer writes and adds to BASIC a routine which samples data from a laboratory device. Then a BASIC programmer can read and analyze the data without being aware of the details of the data transfer.

A second advantage is that programs using these routines can be written, tested, and debugged much faster in BASIC than if all the programming were done in assembly language. There are two reasons for this:

1. BASIC is easier to program than assembly language, and

2. BASIC programs can be written, executed, and debugged with one procedure but assembly language programs require several different procedures (editing, assembling, task building or linking, executing, and debugging).

There are four stages in adding a routine to BASIC:

1. Defining it in the routine name table

2. Passing arguments and results between the routine and BASIC

3. Writing the routine

4. Incorporating the routine in BASIC.

NOTE

This chapter assumes that you are an experienced MACRO-11 programmer, and that you are familiar with your operating system and the operation of its utility programs (editors, MACRO assemblers, task builders or linkers, etc.)

Assembly language routines that use the FORTRAN IV call interface as defined in IAS/RSX FORTRAN IV User's Guide (DEC-11-LMFUA-C-D) can be called from either FORTRAN IV or IAS/RSX BASIC. However, these routines must not access any routines or global locations in FORTRAN IV itself.

4.2 DEFINING A ROUTINE - THE USER ROUTINE NAME POINTER TABLE

You must define each routine in the user routine name pointer table. The table consists of a list of pointers. Each pointer specifies the location of the routine name and the starting address of the routine. The list of pointers is terminated by a pointer with a value of 0.

Each pointer specifies a location which contains the following information in the order listed:

- number of characters in name
- routine name
- starting address

In the IAS/RSX BASIC software kit, the file [11,104] CALLI.MAC contains an empty routine table. You must edit CALLI.MAC and insert the pointers to the routine names after FTBL:. After you add the pointers to CALLI.MAC, it contains:

NOTE

The items printed in red in the listing below indicate the entries that you should edit into the existing file CALLI.MAC.

```
        .GLOBL  FTABI,BKGI
        .GLOBL  RT1NM,RT2NM
        .GLOBL  RT3NM,...,RTnNM
FTABI:  .WORD   FTBL
FTBL:   .WORD   RT1NM
        .WORD   RT2NM
        .WORD   RT3NM
          .
          .
          .
        .WORD   RTnNM
        .WORD   0
```

Where the pointers to the routine names are RT1NM, RT2NM, RT3NM,..., and RTnNM.

The pointers to the routine names must be declared globals unless the routine names are in the same source file as the routine name pointer table.

It is convenient to put the routine name in the routine itself. The beginning of each routine should contain:

```
        .GLOBL  RT1NM
RT1NM:  .BYTE   1$-0$
0$:     .ASCII  "routine name"
1$:
        .EVEN
        .WORD   RT1ST
RT1ST:  ;START OF ROUTINE
        .
```

where the "routine name" is the name the routine is called from BASIC and RT1ST is the starting address of the routine.

An alternative method of arranging this data is to include the routine name in the same source file as the routine name pointer table. In this case you should declare the starting addresses as globals. This method should be used for adding routines to BASIC originally written for FORTRAN IV.

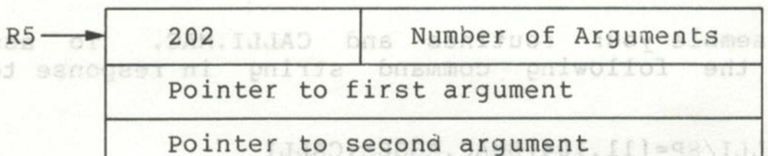
4.3 COMMUNICATING BETWEEN THE ROUTINE AND BASIC

When BASIC executes the CALL statement, it evaluates the arguments and provides the routine with a list of pointers to evaluated arguments' locations.

You must ensure that all BASIC programmers who use your routines have the argument types in the order your routine expects by publicizing the correct data type order. If a CALL statement has the wrong argument types, the results are unpredictable.

When BASIC transfers control to the start of your routine, R5 points to a location which has the constant 202 in the high order byte and the number of arguments in the low order byte. Following this location are pointers to the arguments in the CALL statement. If an argument is null (if there is no data item between the two commas), the value of the pointer is -1.

The argument list format is:



BASIC uses standard PDP-11 1-word integers (2's complement integers) and 2-word floating point numbers (1 sign bit, 8 bits of exponent, and 23 bits of mantissa). These correspond to FORTRAN IV's INTEGER*2 and REAL*4 data types.

Strings are stored as a series of ASCII characters. The pointer points to the first character of the string. You must also have another argument in the CALL statement which specifies the length of the string. In the process of reading and, especially, writing a string, you must ensure that the manipulations occur within the characters of the string only.

If you want to use an array in an assembly language routine, the dimensions of the array must be specified in one or two arguments. The pointer points to the first element of the array, for example A(0) or B%(0,0) and you must know the dimensions to access the elements of the array. In BASIC, arrays are stored with the second subscript varying fastest. (This is the opposite from FORTRAN IV.) You must use the dimensions to calculate the offset of each element. In the process of reading and, especially writing elements of an array, you must ensure that the manipulations occur within the bounds of the array.

4.4 FORMAT OF THE ROUTINE

The routine must start with the address you specified in the routine name table. Use an RTS PC instruction to terminate your routine. You may use the stack during your routine but you must leave it unchanged. R4 contains the stack limit pointer. If you use the stack you should ensure that the limit is not exceeded. R5 points to the argument list and should be left unchanged. For example:

```
          .GLOBL  RT1NM
RT1NM:    .BYTE   1$-0$
0$:       .ASCII  "routine name"
1$:
          .EVEN
          .WORD   RT1ST
RT1ST:    .       ;START OF ROUTINE
          .
          .       ;BODY OF ROUTINE
          .
          RTS     PC
```

4.5 INCORPORATING THE ROUTINES INTO BASIC

You must first assemble your routines and CALLI.MAC. To assemble CALLI.MAC, type the following command string in response to your system's prompt:

```
MAC CALLI,CALLI/SP=[11,104]BMAC,ASSEM,CALLI
```

Then concatenate CALLI.OBJ with the object modules from your assembly language routines into a file named CALLI.OBJ.

At this point you must edit the .CMD or .ODL file in [11,104] which is used to build BASIC (see the <u>IAS/RSX BASIC Installation Guide</u>). You must change in the file the text "CALLIN" to "CALLI" and "CALLSN" to "CALLS". Once you make these edits, you can build BASIC as described in the <u>IAS/RSX BASIC Installation Guide</u>.

APPENDIX A

CHANGES FROM IAS BASIC V01

A.1 SUMMARY OF TECHNICAL CHANGES

The following features are not in IAS BASIC V01 but are in IAS/RSX BASIC V02.

- Integer data type
- Virtual array files
- Statements

  CALL
  DIM #

- Functions

  ABORT
  CTRLC
  RCTRLC
  RCTRLO
  SYS
  TTYSET

- Commands

  CAT
  COMPILE
  DEL
  RESEQ
  SUB

Table A-1 describes the features which are in IAS BASIC V01 and are changed in IAS/RSX BASIC V02.

Table A-1
Changes from IAS BASIC V01

Type	Name	Change from IAS BASIC V01
Statements	CHAIN	Line number specification can be an expression.
	COMMON	BASIC checks to see that COMMON statements are in the same order in each subprogram.
	DATA	Unquoted strings are allowed and the DATA statement cannot have any statements follow it on the same line (multi-statement line).

Table A-1 (Cont.)
Changes from IAS BASIC V01

Type	Name	Change from IAS BASIC V01
Statements	DEF	If the function returns a string, the name must end with a dollar sign. If the function returns an integer, the name must end with a percent sign.
	OPEN	The FILESIZE specification is always ignored.
Functions	BIN	Parentheses around arguments are required.
	OCT	Parentheses around arguments are required.
Commands	LIST	More than one line specification is allowed but the -END line specification is not allowed.

A.2 USING THE CONVERSION AID

IAS/RSX BASIC V02 includes a conversion aid to help you modify programs written under IAS BASIC V01. You should use the conversion aid if you have programs that include any of the following:

- User-defined functions that return strings
- BIN or OCT functions without parentheses
- DATA statements on multi-statement lines

The conversion aid lists all occurrences in a BASIC program of user-defined functions, BIN or OCT functions without parentheses, and DATA statements that have other statements after them. You must keep a record of these lines and read the program with the OLD command. Then you must replace or edit with the SUB command the lines containing the errors.

In addition to just listing the user-defined function names (FNs), the conversion aid can change the FN names. If you specify that the aid should change FN names, the aid requests if each FN name should be changed and, if so, what the new name is.

To use the conversion aid ensure that the file CONVRT.BAS (which is provided in the IAS/RSX BASIC software kit) is in your area. If you use the aid often, you should COMPILE the CONVRT program (see the BASIC-11 Language Reference Manual). Then start BASIC and run the program CONVRT. When you respond to the conversion aid, type Y if you want the feature, type N if you do not want the feature, and always include an extension in file specifications. If you only type the RETURN key in response to the output file specification, BASIC does not create an output file - it only checks the input file.

For example, if the file PROG2.BAS contains the following program

```
LISTNH
10 READ R,A,B,C,D$,E$
20 DEF FNA(A$,B$,N,M)=SEG$(A$,1,N)+SEG$(B$,M,LEN(B$))
30 DEF FNB(X,Y,Z)=X^Y+Z*R
40 N=OCT"177 777"
50 D=OCT(D$)
60 E=OCT(E$)
70 IF D*E<N THEN 32767
80 PRINT FNA(D$,E$,A,C)
90 PRINT FNB(A,B,C)
100 GO TO 32767
10000 DATA 4,2,3,4 \ REM INITIALIZE VARS
10010 DATA "1347","67266"
32767 END

READY
```

the conversion process would be:

```
RUN CONVRT


INPUT FILENAME? PROG2.BAS
OUTPUT FILENAME? PROG2N.BAS
CHECK DATA (Y OR N)? Y
CHECK OCT & BIN (Y OR N)? Y
CHECK FN'S (Y OR N)? Y
CHANGE FN NAMES (Y OR N)? Y


REFERENCE TO FNA:
20 DEF FNA(A$,B$,N,M)=SEG$(A$,I,N)+SEG$(B$,M,LEN(B$))

IS CHANGE REQUIRED (Y OR N)? Y
NEW NAME IS? FNZ$

REFERENCE TO FNB:
30 DEF FNB(X,Y,Z)=X^Y+Z*R

IS CHANGE REQUIRED (Y OR N)? N

REFERENCE TO FNA:
80 PRINT FNA(D$,E$,A,C)

REFERENCE TO FNB:
90 PRINT FNB(A,B,C)

OCT/BIN WITHOUT PARENTHESES:
40 N=OCT"177 777"

DATA NOT LAST:
10000 DATA 4,2,3,4 \ REM INITIALIZE VARS.

CONVERSION COMPLETE
ANOTHER CONVERSION (Y OR N)? N

READY
```

```
OLD PROG2N

READY
LISTNH
10 READ R,A,B,C,D$,E$
20 DEF FNZ$(A$,B$,N,M)=SEG$(A$,I,N)+SEG$(B$,M,LEN(B$))
30 DEF FNB(X,Y,Z)=X^Y+Z*R
40 N=OCT"177 777"
50 E=OCT(D$)
60 E=OCT(E$)
70 IF D*E<N THEN 32767
80 PRINT FNZ$(D$,E$,A,C)
90 PRINT FNB(A,B,C)
100 GO TO 32767
10000 DATA 4,2,3,4 \ REM INITIALIZE VARS
10010 DATA "1347","67266"
32767 END

READY

SUB 40 @"@("@
40 N=OCT("177 777"

READY

SUB 40 @"@")@2
40 N=OCT("177 777")

READY
10000 DATA 4,2,3,4
REPLACE PROG2N

READY
```

You may have to make one other modification to your programs, specifically changing the order of the elements in COMMON statements. If you run your program and BASIC prints the ?COMMON OUT OF ORDER (?COO) error message, you must change the order of the COMMON statements in each subprogram so that they are the same as in the first subprogram. See the BASIC-11 Language Reference Manual for information on the COMMON statement.

INDEX