# MU BASIC/RT-11 User's Manual

DEC-11-LIBRA-A-D

MU BASIC/RT-11 VØ1-Ø1

digital equipment corporation · maynard. massachusetts

First Printing, May 1975

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.

# CONTENTS

## TABLES

## FIGURES

# PREFACE

This manual describes the features of the MU BASIC/RT-11 system and the enhancements made to the BASIC-11 language for this system. BASIC-11 is DIGITAL's name for a family of BASIC[1] systems on the PDP-11. The reader is assumed to be familiar with the BASIC-11 Language Reference Manual (DEC-11-LIBBA-A-D), which describes the features common to all versions of BASIC-11.

All users of an MU BASIC/RT-11 system should read this manual. Chapter 1 provides an introduction to the system and summarizes the enhancements of BASIC-11. Chapter 2 describes file manipulation and the input and output features available. Special system functions and commands are described in Chapter 3. Error messages are listed in Chapter 4. Appendix A lists the standard ASCII (American Standard Code for Information Interchange) characters. Appendix B contains a complete summary of all statements, functions, and commands. Appendix C describes the virtual array capability. The documentation conventions used in this manual are described in Appendix B.

This manual assumes that the MU BASIC/RT-11 system is operational. The information to start and maintain an MU BASIC/RT-11 system can be found in the MU BASIC/RT-11 System Installation Guide (DEC-11-LIBMA-A-D), but users who interface only with a working MU BASIC/RT-11 system do not need access to this guide.

[1]BASIC is a registered trademark of the Trustees of Dartmouth College.

# CHAPTER 1

# INTRODUCTION

MU BASIC/RT-11 is a multiuser BASIC-11 system, capable of accommodating up to eight users simultaneously. Each user independently creates and executes BASIC-11 programs. All the features of the MU BASIC/RT-11 system, including statements, commands, functions, and immediate mode, are available to all users.

MU BASIC/RT-11 runs under the RT-11 monitor. Users can access any RT-11-supported device, including disk, DECtape, magtape, cassette, card reader, high-speed paper tape reader/punch, and line printer.

Throughout this manual the term BASIC refers to both the BASIC-11 language and to the MU BASIC/RT-11 System.

## 1.1 GETTING STARTED

To get started on an MU BASIC/RT-11 system it is first necessary to determine whether the system includes the HELLO feature. The HELLO feature restricts access to the system to authorized users.

See the system manager of your installation to determine whether the HELLO feature is present. If the HELLO feature is not present, go to section 1.1.2.

### 1.1.1 Getting Started on Systems with the HELLO Feature

To get started with BASIC, it is necessary to get a user ID and a password from the system manager. Without a user ID and a password, it is impossible to run any BASIC programs. The user ID is a 2-character alphanumeric code. For example:

```
F7
```

A password can be up to six characters long. For example:

```
PASED
```

The combination of valid user ID and password allows access to the system.

When the BASIC system has been started by the system manager, it prints on all terminals:

```
MU BASIC/RT-11 IS ON THE AIR!
PLEASE SAY HELLO
```

The log-on procedure is the method by which a user gains access to the BASIC system. To log on, type HELLO and press the RETURN key. If anything not beginning with the letter "H" is typed, BASIC prints:

```
PLEASE SAY HELLO
```

After HELLO has been typed, BASIC prints:

```
USERID:
```

Type your user ID and press the RETURN key. BASIC then prints:

```
PASSWORD:
```

Type the password assigned and press the RETURN key. The characters that are typed are not displayed on the terminal. This ensures the privacy of the user ID-password combination.

If the user ID and password entered are not valid, BASIC prints:

```
INVALID ENTRY -- TRY AGAIN
USERID:
```

If this message appears, the user ID and password must be reentered. This message is often caused by typing errors.

If the user ID and password are valid, BASIC prints an informational message. For example:

```
          WELCOME TO MU BASIC/RT-11
```

Then BASIC prints:

```
          READY
```

This indicates that the user may enter any BASIC command, immediate mode statement, or program line. To terminate the session, use the BYE command (see section 1.1.3).

NOTE

> The CTRL/O key command terminates output to the terminal and can be used during the log-on procedure to avoid the printing of the informational message.

### 1.1.2 Getting Started on Systems Without the HELLO Feature

When the BASIC system has been started by the system manager, it prints on all terminals:

```
          READY
```

This indicates that the user may enter any BASIC command, immediate mode statement, or program line. The user ID of all users is AS and all users are privileged (see section 1.2). The HELLO command is not recognized and produces a ?SYN (SYNTAX) error message.

To terminate the session, use the BYE command (see section 1.1.3).

### 1.1.3 Terminating the Session - the BYE Command

To terminate a session at the terminal, type

```
          BYE
```

and press the RETURN key. The BYE command deletes the current user's program and deassigns all devices assigned to the current user.

On systems with the HELLO feature, BYE prints a message and then initiates the log-on procedure. The message is:

```
USERID xx LOGGED OFF -- GOODBYE

PLEASE SAY HELLO
```

where xx is the user ID of the current user.

On systems without the HELLO feature, BYE causes BASIC to print the READY message.

## 1.2 RESTRICTED OPERATIONS

Certain operations are restricted, to prevent one user from interfering with other users' programs and files. Some operations are not allowed for any user, while other operations allowed for a privileged user are not allowed for a nonprivileged user. The system manager determines which users are privileged and which are nonprivileged.

The following lists the operations that are restricted to privileged and nonprivileged users. Nonprivileged users can temporarily execute any operation allowed to privileged users when running a privileged program. See Section 2.3 for more information on privileged programs.

| Operation | Nonprivileged User | Privileged User |
|---|---|---|
| File Access | Restricted by file protection system (see section 2.3) | Unrestricted |
| Maximum number of simultaneously open files | Restricted by system manager | Restricted only by system's resources |
| Maximum size of output data file | Restricted by system manager | Restricted only by available free space on the device |

| Operation | Nonprivileged User | Privileged User |
|---|---|---|
| Nonfile-structured open of a file-structured device (see section 2.5.1.2) | Not allowed | Only prohibited when a public device is opened for output in a nonfile-structured manner |
| Execution of system functions (see section 3.4) | Certain functions are not allowed | Unrestricted |

## 1.3 ENHANCEMENTS TO BASIC-11

There are several features in MU BASIC/RT-11 that are not described in the BASIC-11 Language Reference Manual. All new and changed features are summarized in Tables 1-1, 1-2, and 1-3.

The precedence of arithmetic operators is slightly different than that described in the BASIC-11 Language Reference Manual. Specifically, in the absence of parentheses MU BASIC/RT-11 has this order of priority:

a. Exponentiation (proceeds from left to right)
b. Unary minus

For example,

```
X=3 \ PRINT -2^2,-X^2,2^-2
-4             -9              .25

READY
```

The priority described in the BASIC-11 Language Reference Manual produces values of +4 and +9 for the first two operations. The third operation (2↑-2) produces the same values under both sets of priorities. Operations involving both exponentiation and unary minus should, in general, be enclosed in parentheses to avoid confusion. The immediate mode statement should be rewritten as:

```
X=3 \ PRINT -(2^2),-(X^2),2^(-2)
-4             -9              .25

READY
```

Table 1-1
Enhancements to BASIC-11 Commands

| Command | Meaning |
|---|---|
| CTRL/C | Key command; stops execution of the BASIC-11 statement or command, prints the following:<br>↑C<br>STOP AT LINE xxxxx<br>READY<br>If a command or immediate mode statement is interrupted, the AT LINE is not included. CTRL/C can be disabled by special system function calls (see section 3.4.6). The response to CTRL/C may not be immediate if input or output is in progress. |
| CTRL/Q | Key command; continues output to terminal after CTRL/S. May be labeled XON on some terminals. |
| CTRL/S | Key command; suspends all output to terminal until CTRL/Q, CTRL/O or CTRL/C is typed. May be labeled XOFF on some terminals. |
| CTRL/U | Deletes the entire line currently being entered. BASIC prints:<br>↑U |
| ASSIGN device: | Assigns specified device to user if it is available. |
| BYE | Terminates the user's session. |
| DEASSIGN [device:] | Deassigns specified device or all devices if none is specified. |
| HELLO | Special command to gain access to the MU BASIC/RT-11 system. Not available on all systems. |
| KEY | Enables echoing after TAPE command or SYS(1) or SYS(3) system function call. |
| LENGTH | Displays the amount of memory used by the current program, expressed as number of words. |
| SET TTY type | Sets system to allow different terminals; type may be VT05, ASR33, or LA30. |
| TAPE | Disables echoing and enters special mode to allow use of the low-speed paper tape reader. |
| UNSAVE file descriptor | Deletes specified file. |

Table 1-2
Enhancements to BASIC-11 Statements

| Statement | Meaning |
|---|---|
| KILL string | Deletes specified file. |
| LET VF int (exprl)=expr2 | Special form of LET statement for use with virtual array files. |
| NAME stringl TO string2 | Renames specified file. |
| OPEN | Opens data files; has new keywords for device optimization and special form for virtual arrays. |

Table 1-3
Enhancements to BASIC-11 Functions

| Function | Meaning |
|---|---|
| SYS | System function calls; to control output to terminal and perform system operations. |
| TAB | Causes the terminal head to move to specified column. If the column number specified is less than the current position, a carriage return without a line feed is printed and printing starts at the specified position on the same line (instead of resuming printing at the current position as described in the BASIC-11 Language Reference Manual. |

## 1.4 ASSEMBLY LANGUAGE ROUTINES

Any assembly language routines that the system manager has included with MU BASIC/RT-11 can be executed by the BASIC-11 CALL statement. The CALL statement, including the implied call format, is described in the BASIC-11 Language Reference Manual. The system manager can provide further information on assembly language routines included (if any).

# CHAPTER 2

# INPUT AND OUTPUT

## 2.1 MU BASIC/RT-11 FILES

The MU BASIC/RT-11 file capability allows data and BASIC programs to be stored for future use. Programs that access files can manipulate data much faster and in larger amounts than programs that perform terminal input and output only. Saving programs in files makes it possible to restore them at a later time. Another program can run a saved program by means of the CHAIN or OVERLAY statement.

Data is stored either in sequential files or in random-access, virtual array files. Sequential files are treated in the same way as terminal input/output -- data is read by an INPUT statement and written by a PRINT statement. Sequential files are useful for storing data that is to be processed serially. Virtual array files are similar to arrays stored in memory. An element of data in a virtual array can be part of any BASIC expression just the same as an element of a normal array. An element of a virtual array file can be assigned a value by a special form of the LET statement. Virtual array files allow data to be accessed in a random, non-serial manner and are the only BASIC files in which existing data can be updated without rewriting the entire file.

Each user creates and accesses files independently. A user creating a file can allow others to access the file or can restrict access to it. A privileged user can create a public or group library file that can be conveniently accessed by many users. All users can access public files whereas group files are only available to a specific group of users.

BASIC files can be created and accessed on any device supported by the RT-11 Monitor and included by the system manager in the BASIC system. Among the devices that may be included in the system are disk, DECtape, cassette, industry-compatible magtape, card reader, line printer, and high-speed paper tape reader/punch.

NOTE

Files can only be created on cassettes, disks, DECtapes, and magtapes that have been previously zeroed by the RT-11 PIP program. See your system manager or the RT-11 System Reference Manual for information on PIP.

## 2.2 FILE DESCRIPTOR

BASIC files are created and accessed by file control commands and statements. A file is specified by means of a file descriptor in the general form:

$$\text{[device:]}\begin{bmatrix}\$\\ \#\\ @\end{bmatrix}\text{[filename][.[extension]]}$$

where:

| | |
|---|---|
| device: | is a legal abbreviation representing a device and may be any abbreviation listed in Table 2-1. |
| $ | denotes a public file. |
| # | denotes a group file. |
| @ | denotes a private file (default). |
| filename | is the 1- to 6-character name of the file. |
| extension | is the 1- to 3-character extension of the file. |

Table 2-1
Device Name Abbreviations

| Abbreviation | Represents |
|---|---|
| CR: | Card reader. |
| CT[digit]: | Cassette (unit number specified by digit). |
| DK[digit]: | System device (unit specified by digit). |
| DT[digit]: | DECtape (unit specified by digit). |
| KB: | User's terminal. |
| LP: | Line printer. |
| MT[digit]: | Magtape (unit specified by digit]. |
| PP: | High-speed paper tape punch. |
| PR: | High-speed paper tape reader. |
| RF: | Fixed-head disk. |
| RK[digit]: | RK11 cartridge disk (unit specified by digit). |
| SY[digit]: | System device (unit specified by digit). |

In addition to the devices listed in Table 2-1 any permanent device names added to the RT-11 Monitor can be used from BASIC. See the system manager for the devices and device name abbreviations that are available.

It is not necessary to specify the device. When the device is not specified the system device is assumed. The system device can only be disk or DECtape.

NOTE

The optional digit in the device specification represents the device unit number. If the optional digit is omitted from SY or DK, the system device (unit from which system was bootstrapped) is assumed. If the optional digit is omitted from any other device name, unit Ø is assumed.

## 2.2.1 File Classes

There are three classes of files:

Public library files
Group library files
Private files

Public library files are accessible to all users. Programs that will be accessed by many users may be placed in the public library. Including a dollar sign ($) in a file descriptor specifies a file in the public library. Public library files are stored internally as files with only one-character extensions.

Group library files are accessible to a group of all users having the same first character in their user ID. For example, users with the user ID's H9, HF, HO, and HZ belong to the same group (H is the group character). A group library file is specified by including a number sign (#) in the file descriptor. Group library files are stored internally with a two-character extension. The second character of the extension is the group character.

A private file is accessible only to the user who has created it. A private library file can be specified by including the at sign (@) or by not including any of the special file class symbols ($, # and @). A private file is stored internally as a file with a three-character extension. The second and third characters of the extension are the characters in the user ID of the user who created the file.

### 2.2.2 Filename and Extension

When the specified device supports named files (disk, DECtape, cassette, and magtape support named files) the filename and extension specify the individual file on the device. Only one permanent file with a particular filename and extension can exist on a given device. If a new file is created on a device with the same filename and extension as a file already existing on that device, the old file is deleted when the new file is made permanent (closed). But files with the same filename and extension can exist on different devices; they are completely independent.

For the line printer, high-speed paper tape reader/punch, card reader, or terminal, which do not support named files, any specified filename and extension are ignored.

If a filename is not specified in a SAVE, REPLACE, or UNSAVE command, the current program name is the assumed filename. If any of the file descriptors in a KILL or NAME TO statement are missing the filename, the current program name is also assumed. The OPEN statement does not have a default filename. If neither a filename nor an extension is specified, then a nonfile-structured OPEN is attempted (see section 2.5.1.2). All other statements and commands (APPEND, NEW, OLD, RENAME, CHAIN, and OVERLAY) assume the default filename "NONAME" when none is specified.

An extension specification is necessary only when accessing a private file of another user, a group file of another group, or a file that has a nonstandard extension.

The first character of the extension depends on the type of file. If the file is a BASIC program (OVERLAY and CHAIN statements and APPEND, OLD, REPLACE, RUN, SAVE, and UNSAVE commands) the first character of the standard extension is B. If it is a data file (OPEN, NAME TO, and KILL statements) the first character of the standard extension is D.

The extension actually used by BASIC is dependent on:

 the type of operation involved

 the user ID of the current user

 the extension specified (if any)

 the class of the file (determined by $, #, or @)

The type of operation involved determines whether the first character of the extension is D (for data file operations) or B (for BASIC program operations) unless an extension is specified.

The user ID of the current user becomes the second and third characters of the extension actually used unless an extension is specified or a $ or # is specified.

If an extension is specified, then that is the actual extension used unless a $, #, or @ is specified.

If a file class is specified by a $, #, or @, then it determines the second and third characters of the actual extension used. If $ is specified (public library file), then the second and third characters are nulls. If # is specified (group library file), then the third character is a null. And if @ is specified (private file), then the second and third characters of the extension actually used are the user ID of the current user (see examples below).

The following list contains file statements and commands and the device, file name, and extension they will attempt to access. The access may be restricted by the file protection system (see section 2.3).

These examples assume the current user ID is B9, and the current program name is EVENT.

| File Statement or Command | File Specified | Comment |
|---|---|---|
| OPEN "DATA" AS FILE #1 | SY:DATA.DB9 | The system device is assumed, the file name is specified, the first character of the extension is D because OPEN is a data file statement and the last two characters are the current user ID. |
| SAVE DT1: | DT1:EVENT.BB9 | The device is specified, the program name is the file name, the first character of the extension is B because SAVE is a BASIC program file command, and the last two characters are the current user ID. |
| OLD $PROGRM | SY:PROGRM.B | The system device is assumed, the file name is specified, the first character of the extension is B because it is a program file statement, and there are no second and third characters because it is a public library file. The program name becomes PROGRM. |
| OPEN "#MONEY.G" AS FILE #2 | SY:MONEY.GB | The system device is assumed, the filename is specified, the first character of the extension is specified, and because # specifies a group file, the second character is the group character and there is no third character. |
| OLD PR: | PR: | The papertape reader does not need a filename or extension. The program name becomes the default NONAME. |

| File Statement or Command | File Specified | Comment |
|---|---|---|
| OPEN "DT1:ABC.HB9" AS FILE 1<br>OPEN "DT1:@ABC.H" AS FILE 1<br>OPEN "DT1:@ABC.HHH" AS FILE 1 | DT1:ABC.HB9 | These statements are equivalent. The first statement specifies the file descriptor exactly. The second specifies the @, filename, and first character of the extension; the @ causes the second and third characters of the extension to be the current user ID. The third is the same as the second except that a three character extension is specified, but the second and third characters specified are ignored because of the presence of @. |
| OPEN "STORE.R" AS FILE #1 | SY:STORE.R | The system device is assumed and the filename and extension are specified exactly. This is a public library file because the second and third characters of the extension are nulls. |
| OPEN "STORE" AS FILE #1 | SY:STORE.DB9 | The system device is assumed and the filename is specified. The first character of the extension is D because OPEN is a data file statement, and the last two characters of the extension are the current user ID because a private file is assumed. |
| OPEN "STORE." AS FILE #1 | SY:STORE. | The system device is assumed and the filename (STORE) and extension (the null extension) are specified. |
| OLD DTØ:77GAME.B7R | DTØ:77GAME.B7R | Another user's private file can be accessed only by specifying the extension. |

NOTE

When the user ID is AS the default extension for data files is ".DAT" not ".DAS". AT are the last two characters of the extension in a data file statement only when neither the extension nor the file class is specified. The last two characters of the extension in a BASIC program operation or when @ is specified are the normal AS. This feature allows compatibility with the default extensions in the single user BASIC/RT-11.

## 2.3 FILE PROTECTION SYSTEM

There are several degrees of file access allowed:

| | |
|---|---|
| 1 Run | allows access by the RUN command or CHAIN statement. |
| 2 Read | allows access by the OLD or APPEND command or the OPEN FOR INPUT or OVERLAY statement or use of the value of an element of a virtual array. |
| 3 Update | allows access by the LET VF statement -- can only be done to virtual array files. |
| 4 Complete | allows access by all of the above and by the SAVE, REPLACE, or UNSAVE command or the OPEN FOR OUTPUT, NAME TO, or KILL statement. |

A nonprivileged user is allowed complete access only to the user's own private files. A nonprivileged user can have Run and Read access to files in the public library and the user's own group library. No access is allowed (for a nonprivileged user) to other users' private files and files in other groups' libraries. The access allowed a nonprivileged user to all files other than the user's own private files can be modified by the inclusion of a protection code in the filename.

A privileged user has complete access to all files. Group library and public library files can only be created by a privileged user.

When one of the four digits, 6, 7, 8, and 9, occurs in the first or second character position of a filename it has a special file access meaning. The first character determines the access allowed all users in the group, specified by the second character of the extension of the file. The second character of the filename determines the access allowed for all other users. In public library files, the first character position determines the access allowed for all users. The following list describes the meaning of the digit 6, 7, 8, or 9 in the first or second character position.

| Digit | Meaning |
|---|---|
| 6 | Run, read, and update access. Allows all access except creating. 6 is only useful for virtual array files. |
| 7 | Run and read access. |
| 8 | Run access only. Program erases the user's storage area when it terminates execution (except when run by a privileged user). |
| 9 | Run access only. Program erases the user's storage area when it terminates execution (except when run by a privileged user). Same as 8 except that the program can perform any privileged file operation or system function even when run by a nonprivileged user. |

Any other character in the first or second character position of a filename of a private file specifies no access allowed for users other than the file's creator.

Any other character in the first character position of a filename for a group file specifies read and run access (equivalent to specifying 7) to users in the group. Any other character in the second position specifies no access for all users not in the group.

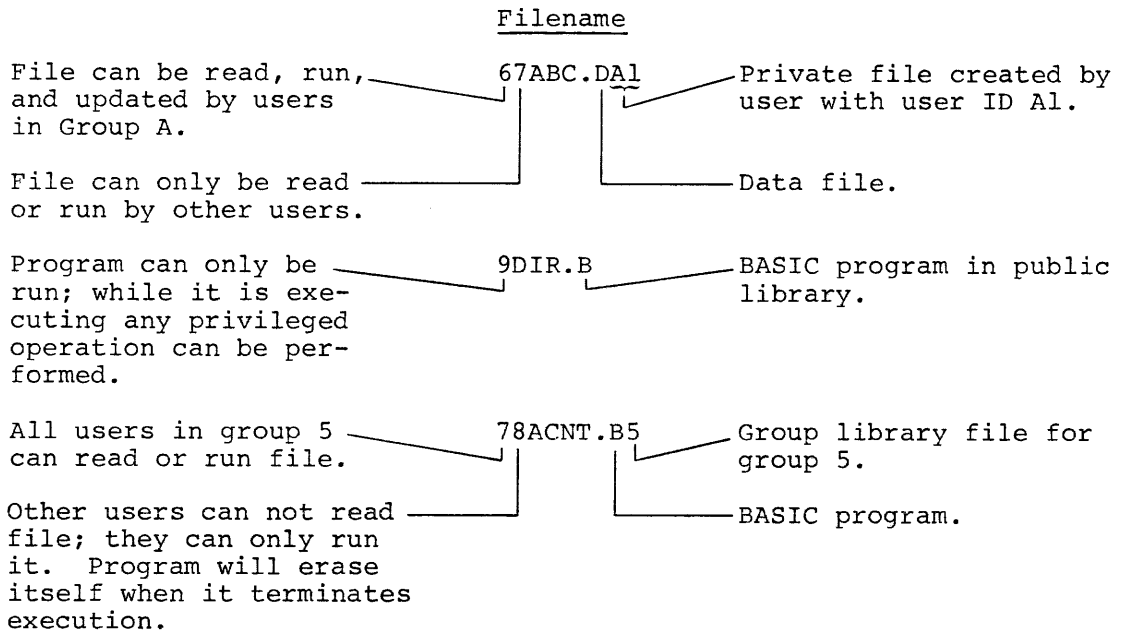
Any other character in the first character position of a filename of a public library file specifies read and run access (equivalent to specifying 7).

A 9 in the first or second character position of a filename allows a nonprivileged user executing the program to perform any operation that a privileged user can. Only a privileged user can create a file with a 9 in the first or second position. It is important that any such programs be carefully coded to ensure privacy of confidential information. Because the program deletes itself on any program termination, including the CTRL/C key command and error conditions, there is no danger that a nonprivileged user can alter it.

NOTE

A privileged user can run, read, update, or create any file. Only a privileged user can create group and public library files (with the exception that a non-privileged user with a one-character user ID can create group library files).

Examples:



Filename

67ABC.DA1

File can be read, run, and updated by users in Group A.

File can only be read or run by other users.

Private file created by user with user ID A1.

Data file.

9DIR.B

Program can only be run; while it is executing any privileged operation can be performed.

BASIC program in public library.

78ACNT.B5

All users in group 5 can read or run file.

Other users can not read file; they can only run it. Program will erase itself when it terminates execution.

Group library file for group 5.

BASIC program.

## 2.4 BASIC PROGRAMS IN FILES -- UNSAVE COMMAND

BASIC program files are stored or updated with the SAVE or REPLACE command, respectively. These files can then be restored by the APPEND, OLD, or RUN command or the CHAIN or OVERLAY statement. These commands and statements are described in the BASIC-11 Language Reference Manual.

Files containing BASIC programs can be deleted by the UNSAVE command. The form of the command is:

```
UNSAVE  file descriptor
```

where

file descriptor    is described in section 2.2.

A file can be deleted only by a user who has complete access to it (see section 2.3).

Examples:

These examples assume the current user ID is GR.

| | |
|---|---|
| UNSAVE TEST | Deletes the file TEST.BGR from the system device. |
| UNSAVE FILE2.DGR | Deletes the data file FILE2.DGR from the system device. |

## 2.5 DATA FILES

### 2.5.1 Sequential Data Files

Sequential files are treated in a manner similar to terminal input/output; however, sequential file data can be manipulated in larger amounts and much more quickly than the equivalent terminal data. Data in sequential files must be accessed serially -- to get to the last data item in a sequential file it is necessary to read the entire file.

The OPEN statement associates a physical file, specified by the file descriptor, with an internal logical unit number (also called channel number). A sequential file can be opened for output or input, not for both. To update an existing sequential file, it is necessary to open it for input and open a new file for output. Then all of the data must be read from the input file and written, including the updates, to the output file.

The PRINT #expr statement outputs data to a sequential file opened for output. The expression after the number sign (#) must have the same value as the logical unit number of the file specified in the OPEN statement. The data stored in the file is the same data that would be printed on the terminal by an equivalent PRINT statement -- this includes space, carriage return, and line feed data.

The INPUT #expr statement reads data from a sequential file opened for input. The value of the expression following # must be the same as the logical unit number specified in the OPEN statement. Data is read from the file in the same way that it is input from the terminal. The INPUT #expr statement reads an entire line of data (up to a carriage return). If only one variable is specified in the INPUT # statement one data item is read from the file data line -- any excess data is ignored. If more than one variable is specified, BASIC reads as many data items from the line as needed. If there is insufficient data, BASIC reads the next data line from the file to get more data. If there is more than one numeric data item on a data line, they must be separated by commas -- spaces alone can not be used to separate data items.

String data is always read to the end of the line so there can only be one string data item on a data line.

NOTE

When creating a sequential file that is to be read later by the INPUT #expr statement, it is necessary to insert the separators required. Commas and carriage returns are valid separators. If there is more than one numeric data item on a file data line, the INPUT #expr must specify a number of variables equal to the number of data items. For example, the data written by:

```
OPEN "DATA" FOR OUTPUT AS FILE #1
PRINT #1, 5, ",", 1Ø
CLOSE
```

would be correctly read by:

```
OPEN "DATA" FOR INPUT AS FILE #2
INPUT #2, A, B
CLOSE
```

Commas must be printed as strings (as in the above example) but carriage returns are output by BASIC after every PRINT #expr statement (unless the list is terminated by a comma or semicolon).

Once a data file has been opened for input, the end-of-file condition can be tested by the IF END #expr statement. An input file can be restored to the beginning (the first data item in the file is read next) by the RESTORE #expr statement.

Output files are made permanent when they are closed. Files are closed by the execution of the CLOSE, CHAIN, or END statement or the execution of the physical end of program (the highest numbered program line). Files are not closed when program execution is terminated by execution of a CTRL/C key command or STOP statement or the occurrence of a fatal program error.

An output file which has been opened but not yet closed is purged by the execution of the SCR, BYE, RUN, OLD, or NEW command. If an output file is purged, all data output to the file is lost and the current contents of the file buffer are discarded.

Sequential files opened for input can be closed or purged. The data stored in the file is unaffected.

NOTE

> The contents of a file buffer are not actually output to the device until the buffer is filled or the file is closed. For example, all output to the line printer is not completed until the file is closed.

The PRINT #, INPUT #, IF END #, RESTORE #, CLOSE, and OPEN statements are described in the BASIC-11 Language Reference Manual. Some additional features of the OPEN statement are described in this section.

The format of the OPEN statement for sequential files is:

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

where:

| | |
|---|---|
| string | is a file descriptor as described in section 2.2. It can be any string expression. |
| FOR INPUT | specifies that an existing file is opened for input. FOR INPUT is assumed if neither FOR INPUT nor FOR OUTPUT is specified. |
| FOR OUTPUT | specifies that a new file is created. Any existing file with the same file descriptor will be deleted when the new file is closed. |
| # | is optional. |
| expr1 | is the logical unit number, an integer in the range 1-127. Other statements access the file by specifying the logical unit number associated with it. The logical unit number is also known as the channel number. |
| DOUBLE BUF | specifies that a second I/O buffer is allocated to the file if there is room in memory. |
| ,FILESIZE expr2 | determines the area allocated for an output file on disk or DECtape. It is ignored on other devices and for input files. |

| | |
|---|---|
| ,RECORDSIZE expr3 | determines the buffer size for nonfile-structured devices or specifies a system buffer and channel (the latter is only available to privileged users). |
| ,MODE expr4 | specifies special handling of magtape or cassette file. See section 2.5.1.1. Any specified MODE is ignored for other devices. |

The FILESIZE option specifies the maximum number of 256-word blocks that the file can occupy. If FILESIZE is not specified, then a default maximum file size is provided by the system manager. The ?IFL (Illegal File Length) error message is printed if a nonprivileged user specifies a FILESIZE less than or equal to zero or greater than the maximum size allowed by the system manager.

For a privileged user a FILESIZE Ø gives the standard RT-11 file size allocation (which is either the larger of the second largest free area or half of the largest free area), a FILESIZE -1 gives the absolute largest free area, and a FILESIZE of less than -1 causes the ?IFL error message.

When a file is closed, the size of the file is reduced to the number of blocks that have actually been used. If output is attempted past the file size allocated, the ?FTS (File Too Short) error message is produced and all data is lost.

When calculating the number of blocks to allocate to a sequential file, the following facts should be considered: there are 512 characters per block, spaces are valid characters, and each line output has two characters (carriage return and line feed) added to it. Once a file has been created it can not be extended.

The RECORDSIZE option can be used to specify the buffer size for a nonfile-structured device. The buffer size is specified in bytes (two bytes equal one word of memory). Output speed can be increased by specifying a buffer size larger than the default. The standard default buffer sizes are given in Appendix D. The ?IRS (Illegal Record Size) error message is produced when a nonprivileged user specifies a buffer size less than 1.

The RECORDSIZE option can be used by a privileged user to request a system buffer and channel. If a RECORDSIZE -1 is specified and no area is available for a buffer in either the system I/O area (a buffer area common to all users) or the user area, the file uses a system buffer. If no system buffer is currently available, the program waits for one to become free. A RECORDSIZE -1 also causes a system channel to be used if no user channel is available. If no system channel is available, the program waits for one. The effect of a RECORDSIZE -1 is that the OPEN statement never fails because, if the channel pool is empty or because no memory is available for a buffer, BASIC waits for a system channel and/or buffer instead of issuing a fatal error message. If a privileged user specifies a RECORDSIZE of 0 or less than -1 the ?IRS error message is printed.

NOTE

If neither filename nor extension is specified in the file descriptor a ?FPV (File Protection Violation) error message is produced for nonprivileged users; but, for privileged users, the device is opened in a nonfile-structured manner. This can happen if a string variable specifying the file descriptor is null. See section 2.5.1.2.

Example:

This example program creates a data file, closes the file, reads it back, and prints the sum on the line printer.

```
10 REM  PROGRAM DEMONSTRATING THE OPEN STATEMENT
20 OPEN "DATA" FOR OUTPUT AS FILE #1
25 REM  CREATES OUTPUT FILE
30 FOR I=1 TO 100
40 PRINT #1,I;",";I^2
50 REM  THIS PRINTS I AND I-SQUARED
60 REM  TO THE FILE
70 REM  NOTE THE STRING CONSTANT ","
80 NEXT I
100 CLOSE 1
110 OPEN "DATA" FOR INPUT AS FILE #2
120 IF END #2 THEN 200
130 INPUT #2,I,I2
140 S=S+I \ S2=S2+I2
150 GO TO 120
200 CLOSE 2
210 OPEN "LP:" FOR OUTPUT AS FILE #1
220 PRINT #1,"THE SUM OF THE NUMBERS";
230 PRINT #1," 1 THROUGH 100 IS";S
240 PRINT #1,"THE SUM OF THEIR SQUARES IS ";S2
250 CLOSE
```

2.5.1.1 Sequential Files on Magtape and Cassette - BASIC operations involving cassette and magtape devices are handled somewhat differently from those involving other devices, because of the sequential nature of cassette and magtape. It is possible to have only one file open at a time on any given cassette or magtape unit. If an attempt is made to open a file on a unit which already has a file open on it, the ?NER (Not Enough Room) error message is produced.

The last file on a cassette or magtape is specially formatted. It marks the end of the existing data and is where new output normally begins. After a new file is created on the device a new specially formatted file is written after it. This file marks the new logical end-of-tape. This specially formatted file is called a double end-of-file on magtape and the sentinel file on cassette.

When a file is opened, the device (CT or MT) is first rewound and then the tape is read until the file specified in the file descriptor is found or the logical end-of-tape is reached. If the file is opened for input, the operation is finished if the file is found, or the ?FNF (File Not Found) error message is produced if the file is not found. If the file is opened for output and the file specified is found, a special empty file descriptor is written there (the old file is deleted) and then the tape is read until the logical end-of-tape is found. In any case, once the logical end-of-tape is reached a file header is written and the operation is completed. This search procedure is employed because there is no inclusive directory at the beginning of the tape and the only way to access a file is to search the tape from the beginning until the file is found.

NOTE

> An existing file on a cassette or magtape is deleted when a new output file with the same filename and extension is opened on the device, not when the file is made permanent.

In conditions when files on disk or DECtape are normally purged (see section 2.5.1), files on cassette or magtape are closed. This is done because it is not possible to create new files on a cassette or magtape if the last output file was not closed. If the tape is physically removed from the drive before the CLOSE operation, there is no logical end-of-tape and it is not possible to create new files on the tape. The MODE option or the RT-11 PIP program (see your system manager) can be used to put a new logical end-of-tape on the device.

When a file on cassette or magtape is deleted a special file header is written to signify that the file is empty. When files are only written after the logical end-of-tape (the normal method of operation), the space freed by the file deletion is not reused. Two methods are available to reuse the empty space. One is to use RT-11 PIP to copy the entire tape and then zero the old tape. The other method is to copy all the files after the empty file to a new tape and then use the mode option to create a logical end-of-tape before the empty file.

MU BASIC/RT-11 does not support multiple volume files on cassette or magtape. If the physical end-of-tape is reached while BASIC is writing a file, the ?DHE (Device Hardware Error) error message is produced and the operation is terminated without being completed. The portion of the file that has been written as well as all the other files on the device can be read but no new files can be created on the device. When reading the last block written on the tape BASIC prints a ?DHE or an ?OOD (Out Of Data) error message, depending on whether the last block was incomplete.

The MODE option in the OPEN statement allows files to be specified by position as well as by the filename and extension. The MODE option can be used when it is not necessary or desirable to rewind the tape before each operation or when the file to be accessed is known to be in a certain position. The MODE option also allows new output to begin before the logical end-of-tape (or when there is no logical end-of-tape).

In an OPEN FOR INPUT statement, the MODE option limits the number of files to be searched and, if the file specified by the file descriptor is not found during the search, causes the last file in the search to be opened for input. If a positive expression is specified after MODE, the tape is not rewound and the search is limited to the integer value of the expression. For example, if the present position is after the 5th file on the tape and a MODE 3 is specified, the file specified by the file descriptor is opened if it is the 6th, 7th, or 8th file on the tape - otherwise the 8th file on the tape is opened. A negative value for the expression following MODE has the same effect except that a rewind operation is done on the tape and then the absolute value of the expression is used to limit the search. For example, a MODE -5 causes the file specified by the file descriptor to be opened if it is the 1st, 2nd, 3rd, 4th, or 5th file on the tape - otherwise it causes the 5th file to be opened. If the logical end-of-tape is reached before the specified number of files have been searched, the ?FNF (File Not Found) error message is produced.

NOTE

> The use of MODE in the OPEN FOR INPUT statement is not restricted by the BASIC file protection system. Any file can be accessed by position including files that have been deleted and files with extensions different from the current user ID. To assure privacy of data on cassettes or magtapes, access to the tape must be restricted and the cassette or magtape unit assigned by the ASSIGN command (see Section 3.1) before the tape is mounted.

The MODE option in an OPEN FOR OUTPUT statement causes output to begin at the position specified by MODE. If MODE is followed by a positive expression, no rewind operation is done and output starts at the file position specified by the integer value of the expression. For example, if the present position on the tape is after the 3rd file, a MODE2 causes the new file to be created as the 5th file. All existing files after the fourth file are lost. If a MODE is followed by a negative expression, a rewind operation is done and output starts the position specified by the absolute value of the expression. If the logical end-of-tape is reached before the specified file position has been reached, output begins at the logical end-of-tape.

All empty as well as active files on cassette and magtape are counted. RT-11 PIP can be used to create a magtape or cassette directory that lists all active and empty files. Magtape sometimes have an empty file inserted by the system at the beginning of the magtape. See your system manager or the RT-11 System Reference Manual (DEC-11-ORUGA-B-D) for more information on PIP and magtape and cassette handling.

Care must be employed when using the MODE option in an OPEN FOR OUTPUT statement. One reason is that if the rewind operation is inhibited (by a positive value of the expression) any file with the same filename and extension before the current position is not deleted. In this case, two files exist on the same device with the same filename and extension. In a subsequent OPEN FOR INPUT statement only the first file is seen unless the MODE option is used and the search begins after the first file. A second danger involving use of the MODE option in an OPEN FOR OUTPUT statement is that all files after the output file are lost. No check of file protection is made on these files; consequently, to prevent the loss of information stored on cassette, the cassette must be WRITE-PROTECTED (by uncovering the hole on the cassette) or access to the cassette must be restricted.

NOTE

> All files on cassette are written in groups of four 64-word blocks. The file header and sentinel file are written in special 32-word blocks.

### 2.5.1.2 Nonfile-Structured OPEN Statement

- A privileged user can open a disk, DECtape, cassette, or magtape in a nonfile-structured manner by not specifying the filename and extension. A disk or DECtape can be opened as either a sequential file or a virtual file (see section 2.5.2). In both cases the entire device is treated as though it were a normal BASIC file. To preserve the integrity of the system, public devices (see section 3.1) cannot be opened for output in a nonfile-structured manner, even by a privileged user. Nonpublic devices can be opened for output but this should not be done to a device that contains any RT-11 files to be preserved because any existing files will be lost.

Care must be employed when opening a cassette in a nonfile-structured manner. Only a cassette that has been written in a nonfile-structured manner can be read in a nonfile-structured manner. Normal RT-11 cassettes can not be read in a nonfile-structured manner and cassettes created in a nonfile-structured manner cannot be read in a normal file-structured manner.

In general, devices opened for output in a nonfile-structured manner will be nonstandard and will not be readable by all RT-11 system programs and should only be used if the maximum data storage capability of a medium is necessary.

## 2.5.2 Virtual Array Files

Virtual array files are special random-access binary files. There are three data types for virtual array files: integer, floating point, and string. A file can only contain one data type.

Virtual array files have several advantages over sequential files:

> They can be accessed in a random, nonsequential manner. The last element of a virtual array file can be accessed as quickly as any other element whereas it is necessary to read an entire sequential file before accessing the last element.

Data conversions are not required because numbers are stored in binary; consequently there is no loss of accuracy in writing a number out to a virtual file and then reading it back. The data conversion required by sequential files causes some loss of precision.

Virtual array files are the only BASIC files that can be updated without copying the entire file.

Virtual array files are treated like normal arrays stored in memory with the following differences:

Virtual array files allow the processing of much larger arrays than could fit in available memory.

They also allow data to be saved from one execution of a program to another.

Elements of a virtual array file are not initialized to zero when a program is run.

Integer virtual files are restricted to integer values; no similar restriction applies to any array in memory.

Strings in virtual files have a fixed maximum length which is determined when the file is opened; elements of string arrays in memory can be any length (up to 255 characters). Elements of string virtual arrays that are shorter than the maximum length have null characters appended to them until they are the maximum size. When an element is retrieved it is stripped of its trailing null characters.

Elements of a virtual array file can be assigned values only by a special form of the LET statement while elements of normal arrays can be assigned values by the LET, READ, INPUT, or CALL statements.

Virtual arrays can only have a single subscript (dimension); arrays in memory can have one or two dimensions.

The special form of the OPEN statement for virtual files is:

```
            ┌FOR INPUT ┐                 $
OPEN string │          │ AS FILE VF int[ ][(exprl)][=expr2][,FILESIZE expr3]
            └FOR OUTPUT┘                 %
```

where:

| | |
|---|---|
| string | is a file descriptor as described in section 2.2 and can be any string expression. |
| FOR INPUT | an existing virtual array file is opened. Both output and input are allowed. If neither FOR INPUT nor FOR OUTPUT is specified, FOR INPUT is assumed. |

| | |
|---|---|
| FOR OUTPUT | specifies the creation of a new virtual array file. Any previous file with the same file descriptor is deleted when the new file is closed. Both output and input are allowed. |
| int | is the virtual array logical unit number, an integer in the range 1 to 127. Elements of the array are accessed by specifying this virtual array logical unit number. This logical unit number is completely independent from sequential file logical unit numbers. |
| % | indicates an integer virtual array file. |
| $ | indicates a string virtual array file. |
| | When neither % nor $ is specified, the virtual array file is floating point. |
| expr1 | is the dimension of the file. It is the maximum subscript that can be used in accessing the array. This is not true when FILESIZE is also specified. |
| =expr2 | is the fixed maximum string length in the range 1 to 255. This should be specified only when $ is also specified. Default value is 32. |
| FILESIZE expr3 | has the same meaning as in the normal OPEN statement (see section 2.5.1). The maximum legal subscript is determined by the number of elements that can fit in the file. When FILESIZE is specified it supersedes the dimension specified in expr2. |

NOTE

When a virtual file is first created, FOR OUTPUT must be specified but to update the data in the file FOR OUTPUT must not be specified. One consequence of this is that the program to create a new virtual file must be changed to update the file.

The actual size of the permanent virtual file is determined only by the OPEN FOR OUTPUT statement that created it. When the file is closed the file size is not reduced. Nor can the file be extended once it has been opened. The only way to extend a virtual array is to open a new, larger virtual array file and then copy all the data from the old file to the new one.

To save the data in a virtual file, the file must be made permanent (closed). Virtual files are made permanent or purged (lost) under the same conditions as sequential files (see section 2.5.1).

Once a virtual array file has been opened its elements can be used the same as any other variables in a BASIC program with one exception: an element of a virtual array can only have a value assigned by a special form of the LET statement:

```
[LET] VF integer(expr1)=expr2
```

where:

| | |
|---|---|
| integer | must be an integer constant (not an expression) and specifies the virtual array logical unit number. |
| expr1 | specifies the element in the array. |
| expr2 | must be the same data type as the array. |

If an integer virtual file element is assigned a value greater than 32767 or less than -32768, the ?IDT (Illegal Data Type) error message is produced.

Table 2-2 describes the number of elements of each data type that can be stored in one 256-word (512-byte) block.

Table 2-2
Storage Requirements of Virtual Arrays

| Data Type | Bytes per Element | Elements per Block |
|---|---|---|
| Floating point | 4 | 128 |
| Integer (%) | 2 | 256 |
| String ($) | string length | 512/string length |
| String ($) - default length | 32 | 16 |

The number of string elements per block can be fractional.

NOTE

Virtual files can exist only on disk or DECtape.

Examples:

The following examples assume the current user ID is B9.

```
OPEN "TEST" AS FILE VF1$ (2000) = 10
```

Opens the existing file SY:TEST.DB9 as virtual array file 1 containing 2001 string elements; each one 10 characters long. This file is now available for input and output operations. A reference to file element 2001 or greater causes an error.

```
OPEN "TEST2" FOR OUTPUT AS FILE VF2$ (500)
```

Creates a new file SY:TEST2.DB9 as virtual array file 2 with 501 string elements, each 32 characters long.

```
OPEN "TEST3" FOR INPUT AS FILE VF3
```

Opens the existing file SY:TEST3.DB9 as virtual array file 3. It consists of floating point numbers because neither $ nor % is specified. Both input and output operations are legal.

```
LET A$="TEST4"
OPEN A$ FOR OUTPUT AS FILE VF4% (50),FILESIZE (10)
```

Creates the file SY:TEST4.DB9 and opens it for input or output as virtual array file 4 with 10 blocks. The FILESIZE specified overrides the dimension (50); consequently, elements 0 to 2559 can be accessed.

These files can then be used in BASIC operations as follows:

```
LET A = B + VF3(I)/2
```

Uses the value of virtual array file element VF3(I) in computing an expression.

```
PRINT "VARIABLE",N,VF4(N)
```

Uses the value of integer virtual array file element VF4(N) in a print list.

```
LET VF3(2*N+1) = (A + B)/2
```

Sets the value of virtual array file element VF3(2*N+1) to the value of the expression (A+B)/2.

LET VF1(101) = "ABCD"

Sets the value of string virtual memory file element VF1(101) to "ABCD" followed by six null characters. When VF1 (101) is evaluated the trailing nulls will be automatically deleted.

Any virtual file opened should be closed by the CLOSE statement.

CLOSE [VF integer1 [,VFinteger2,VFinteger3,...]]

Examples:

| | |
|---|---|
| CLOSE VF3 | closes virtual file 3 |
| CLOSE | closes all files including virtual files |

NOTE

There may be problems when two users attempt to simultaneously assign values to elements of the same virtual array file. See section 2.5.4.

Example:

```
100 PRINT "OCTAL DUMP" \ REM THIS PROGRAM PRINTS AN OCTAL
110 PRINT "FILE NAME"; \ REM DUMP OF THE SPECIFIED FILE
120 INPUT F$
130 PRINT "START BLOCK,#BLOCKS";
140 INPUT B1,B2
190 OPEN F$ FOR INPUT AS FILE VF1%
200 OPEN "LP:" FOR OUTPUT AS FILE #1
210 PRINT #1:"OCTAL DUMP OF FILE ";F$
220 FOR B=B1 TO B1+B2-1
230 PRINT #1
240 PRINT #1:"BLOCK";B
250 FOR L=0 TO OCT'37'
260 LET V=L*16
270 GOSUB 1000
280 PRINT #1:SEG$(V$,4,6);"/";
290 FOR S=0 TO 7
300 LET V=VF1(B*256+L*8+S)
310 GOSUB 1000
320 PRINT #1:" ";V$;
330 NEXT S
340 PRINT #1
350 NEXT L
360 PRINT #1
370 NEXT B
380 CLOSE #1
390 STOP
1000 LET V1=V \ REM          THIS SUBROUTINE CONVERTS
1005 REM                     INTEGER V
1010 LET V$="" \ REM         TO ASCII STRING V$, WHICH
```

```
1020 LET V1$='0' \ REM         IS THE OCTAL VALUE OF V
1030 IF V>=0 THEN 1060 \ REM   USES V1,V2,V3,V1$,V2$
1040 LET V1=V1+2^15 \ REM      V IS PRESERVED
1050 LET V1$='1'
1060 FOR I=1 TO 5
1070 LET V3=INT(V1/8)
1080 LET V2$=STR$(V1-V3*8)
1090 LET V$=V2$&V$
1100 LET V1=V3
1110 NEXT I
1120 LET V$=V1$&V$
1130 RETURN
```

For more information on virtual files see Appendix C.

## 2.5.3 File Deletion and Renaming

The NAME TO statement changes the name of a file on disk or DECtape; it does not alter the contents of the file. The form of the NAME TO statement is:

```
NAME string1 TO string2
```

where:

| | |
|---|---|
| string1 | is a file descriptor as described in section 2.2 that specifies the file to be renamed. |
| string2 | is a file descriptor that specifies the new name and extension. No device should be specified. |

BASIC programs can be renamed by the NAME TO statement by explicitly specifying the extension. A NAME TO is illegal on cassette and magtape and is ignored on devices which do not support named files.

Examples:

These examples assume the current user ID is B9.

| | |
|---|---|
| NAME "DATA" TO "OLDDAT" | Changes file SY:DATA.DB9 to SY:OLDDAT.DB9. |
| NAME "@PROG.B" TO "@PROG2.B" | Changes file SY:PROG.BB9 to SY:PROG2.BB9. |
| F$="PROG"<br>NAME "@"+F$+".B" TO "@"+F$+".A" | Changes file SY:PROG.BB9 to SY:PROG.AB9. |

The KILL statement deletes a file. It is similar in effect to the UNSAVE command except that KILL assumes a data file. The form of the KILL statement is:

```
          KILL string
```

where:

| | |
|---|---|
| string | is a file descriptor as described in section 2.2. |

Examples:

These examples assume the current user ID is BS.

```
     KILL "FILE1"          Deletes the file SY:FILE1.DBS.
     KILL "PROG.BBS"       Deletes the file SY:PROG.BBS.
```

## 2.5.4 Simultaneous File Updating Restrictions

BASIC does not place any restriction on more than one user opening the same file. A user can be prohibited access to a file by the file protection system, but there is no mechanism to restrict access to a file only when another user has opened it. There is no conflict when only one user is writing to a file and all other users are reading it. There are two conditions in which conflict occurs:

Two or more users have opened a file for output with the same filename and extension (on the same device).

Two or more users are updating an existing virtual array file.

If two or more files are opened on a device with the same filename and extension, only the last file closed remains permanent. Ordinarily this problem should not occur since nonprivileged users do not have the ability to create files with extensions whose second and third characters are different from the user ID. It could occur if two or more users have logged into BASIC with the same user ID; consequently, this practice is not recommended. Privileged users do have the ability to create files with any extension and should take care that they do not create files with the same extension used by a nonprivileged user. The same restriction applies to replacing BASIC programs.

The virtual file facility provides a means for maintaining a data base. This data base can be read by more than one user simultaneously. It is important to note that BASIC does not provide any record-locking facility. If two users attempt to write values into the same record, the user who accesses another record (or closes the file) last has the values written. The values written by all other users are lost.

# CHAPTER 3

# SYSTEM COMMANDS AND FUNCTIONS

## 3.1 ASSIGNING DEVICES

BASIC ensures that each user can independently access devices. Some devices, disk and DECtape units, can be accessed safely by more than one user simultaneously. These devices are called public devices. Other devices, cassette, magtape, card reader, high-speed paper tape punch, high-speed paper tape reader, and line printer, can be accessed by only one user at a time. These are called nonpublic or assignable devices. In addition, the system manager can designate any disk or DECtape unit as nonpublic.

NOTE

Each separate unit of a device is considered unique, for example, CTØ: (Cassette unit Ø) and CT1: (Cassette unit 1) are considered separate devices.

Only one user can access a nonpublic device at any one time. The device becomes available to other users only when the user currently accessing the device releases it. There are two ways to obtain access to a device:

1) By means of the ASSIGN command.
2) By means of a file-related command or statement.

When a user enters an ASSIGN command for an available device, BASIC assigns the device to that user. No other user can access the device until the user who has obtained assignment enters a DEASSIGN or BYE command to deassign the device. When a user enters an ASSIGN command for a public or unavailable device, the ?DNA (Device Not Available) error message is produced.

If a user accesses an unassigned device by means of a file-related statement or command, the device is made unavailable to other users until the operation is completed.

File-related statements and commands are the OPEN, KILL, NAME TO, OVERLAY, and CHAIN statements and the SAVE, UNSAVE, OLD, APPEND, and RUN file descriptor commands. (For all statements and commands except the OPEN statement the device is unavailable until the operation is finished.) After the execution of an OPEN statement the device is unavailable to other users until all the user's open files on that device are closed.

Figure 3-1 illustrates the possible states for a nonpublic device.

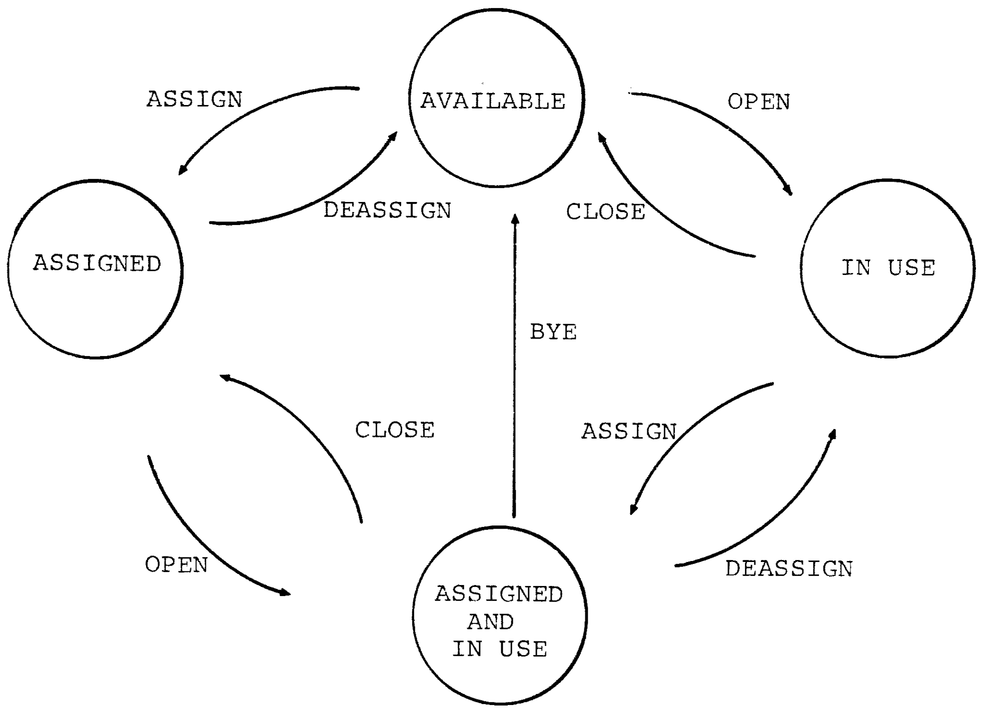


Figure 3-1
Possible States for Nonpublic Devices

ASSIGNED signifies exclusive use and IN USE signifies temporary use. Both conditions can exist concurrently for the same user. OPEN signifies execution of the OPEN statement or the beginning of any other file related command or statement. CLOSE indicates the execution of the CLOSE statement (or of any statement or command that closes files) or the completion of any file related command or statement (other than OPEN). A public device is always AVAILABLE, even when it is being used.

Every file on a device must be closed before the device becomes available. For example, if a user has not assigned a device but has opened two files on it, both files must be closed before the device becomes available to other users.

The form of the command to assign a device is:

```
ASSIGN device:
```

where device: is one of the device symbols described in Table 2-1. If an attempt is made to ASSIGN a device that is already assigned to the same user the command is ignored and no error message is produced.

The form of the command to deassign a device is:

```
DEASSIGN [device:]
```

where device: has the same meaning as in the ASSIGN command.

If no device is specified in the DEASSIGN command, all devices assigned to the user are deassigned. The BYE command also deassigns all devices.

## 3.2 LENGTH COMMAND

The LENGTH command displays on the terminal the amount of storage required by the BASIC program in memory. This information is useful in determining the minimum user area in which a specific program can run. The system manager can use this information in determining the size of the user areas (see MU BASIC/RT-11 System Installation Notes). The form of the command is:

```
LENGTH
```

The LENGTH command produces this message:

```
mmm WORDS USED, nnn FREE
```

where:

| | |
|---|---|
| mmm | is the number of words currently occupied by the user's program |
| nnn | is the number of words remaining free in the user's area |

The number of words in use includes memory currently needed by the BASIC program itself, arrays, string variables, and file buffers in the user's area. To determine the size of the program alone, enter the LENGTH command immediately after an OLD or CLEAR command. Arrays are created after the RUN command and file buffers are created when the OPEN statement is executed. The memory required for string variables and string arrays varies with the current values of the strings; consequently, the LENGTH command returns the current memory requirements, which may be smaller than the maximum memory requirements.

Several error messages can be produced when the program exceeds the amount of memory available. These errors are ?ATL (Array Too Large), ?BSO (Buffer Storage Overflow), ?PTB (Program Too Big), and ?SSO (String Storage Overflow). Program size can be reduced by several procedures:

Eliminate or reduce unnecessary items such as REMark statements, long printed messages, and optional keywords such as LET.

Make maximum use of multiple statement lines.

Make efficient use of program loops, subroutines, user-defined functions, and computed GO TO statements.

Split up large programs into several smaller programs by use of CHAIN or OVERLAY statements.

Reduce the size of arrays in memory to the size required (DIMension statement).

Use virtual array files for arrays that are too large to fit into memory.

Reduce the number of variables and arrays in a program by reusing them when their contents are no longer needed, instead of creating new variables or arrays.

> Reduce the number of simultaneously open files by opening a file just before it is needed and closing it immediately after the last use.

After program lines are deleted, the program can be stored by the SAVE command and restored by the OLD command to further optimize program memory requirements.

File buffers are stored in a system I/O area common to all users if there is no room in the user area. If there is no room in either the user's area or the system I/O area, a ?BSO (Buffer Storage Overflow) error message is printed, except when a privileged user has specified the RECORDSIZE -1 option in the OPEN statement. See section 2.5.1 for a description of the RECORDSIZE option.

For more information on program storage see Appendix D.

## 3.3 TERMINAL CHARACTERISTICS

Many different terminals can be used with BASIC. Several system functions and commands allow use of specific characteristics of the terminals. The SET TTY command specifies the class of terminal; system function 6 sets the width of the terminal; CTRL/S and CTRL/Q key commands stop and then restart output to the terminal, the TAPE and KEY commands, and system functions 1, 2, and 3 facilitate the use of the low-speed paper tape reader on the ASR33 terminal. The system functions are described in section 3.4.

BASIC has a type-ahead feature which allows terminal input to be entered before BASIC is actually ready to process it. For example:

```
OLD PR:
LIST
RUN
4

READY


NONAME  17-MAR-75  MU BASIC/RT-11 V01-06

10 INPUT A
20 PRINT A,A^2,A^3

READY


NONAME  17-MAR-75  MU BASIC/RT-11 V01-06

? 4              16              64

READY
```

While the paper tape is being read the LIST and RUN commands and the data 4 have been entered. The first READY indicates that the OLD command has finished. The program is then listed and run. The question mark is produced by the INPUT statement; however, because the data has already been entered BASIC prints the values on the same line as the question mark.

If the CTRL/C key command is typed at any time, however, it is seen first and any unprocessed type-ahead is discarded.

If type-ahead exceeds thirty characters, the next character is accepted but is not echoed until the type-ahead is processed. If still more characters are entered BASIC rings the terminal bell to indicate that no more characters can be accepted.

NOTE

Extreme care should be used with the type-ahead feature. The printing on the terminal may become difficult to read. This does not affect the processing of the type-ahead.

## 3.3.1 Stopping Output to Terminal

The CTRL/S key command (may be labeled XOFF on some terminals) temporarily suspends all output to the terminal. This allows an alphanumeric display terminal (VT05, VT50, etc.) to be photographed, copied or read without the information on the screen being lost. The CTRL/S key command itself does not cause any character to be printed on the screen (it is not echoed).

The CTRL/Q key command (may be labeled XON on some terminals) causes output to the terminal to resume. The CTRL/Q command itself does not cause any character to be printed (it is not echoed). No characters are lost in a CTRL/S - CTRL/Q combination. In addition to its normal functions, the CTRL/C command also causes output to the terminal to resume after a CTRL/S command.

NOTE

While both the CTRL/O and CTRL/S key commands stop output to the terminal, they are not equivalent. The CTRL/O key command causes all output to the terminal to be lost - the program continues executing but does not print anything on the terminal. The CTRL/S command stops output to the terminal and program execution. When program execution is resumed (by the CTRL/Q command) output resumes. No program output is lost during this CTRL/S and CTRL/Q combination.

### 3.3.2 Using the Low-Speed Paper Tape Reader/Punch

BASIC usually prints on the terminal's printer every character input by the low-speed paper tape reader or the keyboard. This process, called "echoing", provides a visual confirmation of what has been typed; however, when reading tapes from the low-speed reader echoing is often undesirable, because tapes often contain data that is frequently reused.

The TAPE command stops the echoing process. It also causes the RUBOUT key command to be ignored. This allows paper tapes to be input on the low-speed reader without printing the contents of the tape on the terminal. When creating a papertape to be input using the TAPE command, an erroneous character should be deleted by physically backspacing the tape and then typing the RUBOUT key.

The form of the command is:

```
TAPE
```

The TAPE command has the same effect as the system function 1.

After the TAPE command has been executed the echoing process may be resumed by typing the KEY command.

The form of the KEY command is:

```
KEY
```

If the TAPE command has been executed the KEY command itself is not echoed but all following characters typed at the terminal are echoed.

The KEY command also causes RUBOUT to have its usual effect.

Echoing is also resumed after a CTRL/C key command or the execution of system function 2.

NOTE

When creating binary tapes or tapes of BASIC programs (whose lines are longer than the terminal width) on the low-speed paper tape punch use the SYS(6,0) system function to prevent automatic printing of carriage return/line feed combinations. During its normal operation, BASIC occasionally outputs a null character to the low-speed punch. This is ignored when BASIC reads the tape.

## 3.3.3 SET TTY Command

The terminal type for all local terminals should be set by the system manager when the system is initialized; the SET TTY command should not be used on these terminals. However, to allow any type of dial-up terminal to be used, a user can set the correct number of fill characters with the SET TTY command. In this respect, there are three classes of terminals:

| | | |
|---|---|---|
| 1. | ASR33 | Teletype[1] with low-speed paper tape reader/punch |
| | KSR33 | Teletype |
| | ASR35 | Teletype with low-speed paper tape reader/punch |
| | KSR35 | Teletype |
| | LA3Ø-P | Parallel DECwriter |
| | VTØ5 | Alphanumeric display with a data rate less than or equal to 300 baud. |
| | LA36 | DECwriter II |
| | VT5Ø | Alphanumeric display |
| 2. | LA3Ø-C | Serial DECwriter |
| | LA3Ø-E | Serial DECwriter |
| 3. | VTØ5 | Alphanumeric display with a data rate greater than 300 baud. |

For all terminals in the first class the appropriate SET TTY command is:

```
SET TTY ASR33
```

For serial LA3Ø DECwriters the appropriate command is:

```
SET TTY LA3Ø
```

[1]Teletype is a registered trademark of the Teletype Corporation.

For a VTØ5 alphanumeric display with a data rate greater than 300 baud, the appropriate command is:

```
SET TTY VTØ5
```

The SET TTY command has no effect on the terminal margin. See section 3.4.5 for information on setting the terminal margin.

## 3.4 SYSTEM FUNCTION CALLS

System function calls perform a variety of operations. Certain system functions are available to all users. These cancel a CTRL/O typed at the user's terminal, disable echoing, reenable echoing, enter a special single-character input mode, scratch the user's program in memory and then return to the READY message, and return the current user ID. Certain other system function calls can be executed only by a privileged user. These disable the CTRL/C interrupt, set the user ID, terminate the privileged user status, and cause BASIC to exit and return control to the RT-11 monitor.

The system functions can be used in any arithmetic expression, but for reasons of simplicity and compatibility it is recommended that system functions only be used in the LET statement.

The format of the system function call in the LET statement is:

```
[LET] var=SYS (expr1[,expr2])
```

where

| | |
|---|---|
| var | is the target variable. The value returned by the system function is stored in the target variable. |
| expr1 | determines the system function to be performed. |
| expr2 | is an optional argument used in some SYS function calls. |

Table 3-1 describes the system functions that are performed when expr1 is in the range -4 to +8. If the value of expr1 is less than -4 or greater than 8 the ?ARG (Argument Error) message is printed. If expr2 is specified and not expected the ?SYN error message is printed.

Table 3-1
Summary of System Function Calls

| Value of expr1 | Function Executed by the SYS Call |
|---|---|
| 0 | Cancel effect of CTRL/O typed on terminal. |
| 1 | Enter special mode for inputting tapes on the low-speed reader. |
| 2 | Enable echoing. Cancels effects of SYS(1) and SYS(3). |
| 3 | Disables echoing. |
| 4 | Single character input mode. Target variable contains the ASCII value of the next character typed at terminal. Optional argument, expr2, specifies logical unit number of file. |
| 5 | Delete current program, change program name to NONAME, and print the READY message. |
| 6 | Set the terminal margin to the value of expr2. |
| 7 | Re-enable CTRL/C as an interrupt. |
| 8 | Privileged function call; disable CTRL/C as an interrupt. Causes BASIC to ignore any CTRL/C typed at terminal. |
| -1 | Privileged function call; sets the user ID to the value specified by ASCII value in expr2. If expr2 is negative return the current user ID as the value of the function. |
| -2 | Clear privileged user status. User will not be able to execute any privileged functions or file operation. |
| -3 | Privileged function call; return to RT-11 monitor. A SYS(-3) function call should be executed by a privileged user only if he has notified all other users that the system will be taken down and that they should terminate their jobs. |
| -4 | Return privileged status (+1 for privileged or Ø for nonprivileged). |

It is possible to combine two or more system function calls in one statement, for example, the statement:

```
LET A = Ø*SYS(3) + SYS(4) + Ø*SYS(2)
```

turns off echoing, inputs a character from the terminal, and turns echoing back on. The system functions are executed in order of the normal BASIC-11 precedence of operations.

### 3.4.1 System Function Ø - Disabling CTRL/O

The CTRL/O key command stops output to the terminal; execution of the system function call Ø causes output to resume.

Example:

In the following example, data is input from a file. Each value is printed on the terminal and then a sum is printed. A CTRL/O key command typed any time before line 100 is executed suppresses printing of all further data items but does not suppress the printing of the final sum.

```
5 REM PROGRAM TO INPUT DATA
7 REM AND PRINT SUM
10 OPEN "PRI" FOR INPUT AS FILE #1
20 PRINT "DATA IN FILE:"
30 IF END #1 THEN 100
40 INPUT #1,D
50 PRINT D,
60 T=T+D \ GO TO 30
100 A=SYS(0)
110 PRINT  \ PRINT "SUM="T
```

### 3.4.2 System Functions 1, 2, and 3 - Using the Low-Speed Paper Tape Reader/Punch

BASIC usually echoes every character input by the low-speed reader or the keyboard. This provides visual confirmation of what has been typed; however, echoing is undesirable in certain cases. System functions 1 and 3 disable echoing. System function 2 enables echoing.

System function 1 allows compatibility with paper tapes produced by other systems when they are read by the low-speed reader. System function 1 disables echoing and causes all RUBOUTs to be ignored. RUBOUT is the character normally used in BASIC to delete the previous character typed. Some other systems correct an error on paper tape by physically backing up the tape and then typing a RUBOUT to erase a character. System function 1 should only be used to read tapes prepared in this manner. System function 3 should be used for reading all other paper tapes on the low-speed reader. System function 2 cancels the effect of system function 1 or 3. Echoing is resumed and RUBOUT resumes its usual meaning.

Example

```
10 REM PROGRAM TO INPUT DATA
20 REM FROM THE LOW SPEED READER
25 DIM A(100)
30 Z=SYS(3) \ REM DISABLE ECHOING
40 OPEN "KB:" FOR INPUT AS FILE #6
45 FOR I=1 TO 100
60 INPUT #6,A(I) \ REM INPUT FROM READER
70 NEXT I
100 Z=SYS(2) \ REM RESUME ECHOING
```

NOTE

Turning off the echoing does not affect printing caused by PRINT statements and BASIC-11 messages.

## 3.4.3 System Function 4 - Single-Character Input Mode

System function 4 returns the decimal ASCII value of the next character input from the terminal or from a file. (See Appendix A for a list of the ASCII values.) System function 4 is the only method for BASIC programs to process terminal input without waiting for a carriage return to be typed. This allows interactive programs to use single character response and not require a carriage return.

Any key or key combination on the terminal is a valid response to a system function 4 input request except CTRL/S or CTRL/Q. CTRL/C is only valid if the CTRL/C command is disabled.

Example

```
10 PRINT "FOR HELP TYPE H--";
20 A=SYS(4) \ REM SINGLE CHARACTER INPUT
25 PRINT
30 IF A=ASC("H") THEN 200
40 STOP
200 PRINT "INFORMATION THAT YOU SHOULD KNOW"
```

To input a single character from a sequential file, specify the logical unit number of the file as the second argument in the system function 4 call. System function 4 then returns in the target variable the ASCII value of the next character in the file. All characters including nulls are returned. This allows data in any file to be read with no need for separating commas or carriage returns. Binary files can be copied exactly by use of system function 4. When the end of the file is reached system function 4 returns a value of -1. (Successive calls also return -1.)

The file must be opened for input as a sequential file, not as a virtual file.

NOTE

> If a file is opened on the terminal (KB:) a system function 4 acting on the terminal file is the same as a system function 4 with no file specified. CTRL/S and CTRL/Q are not valid responses on a terminal to a system function 4 input request.

Example

This example will copy any RT-11 file exactly.

```
10 PRINT "INPUT FILE";
20 INPUT I$
30 PRINT "OUTPUT FILE";
40 INPUT O$
50 OPEN I$ FOR INPUT AS FILE #1
60 OPEN O$ FOR OUTPUT AS FILE #2
70 A=SYS(4,1)
80 IF A=-1 THEN 200
90 PRINT #2,CHR$(A);
100 GO TO 70
200 CLOSE
210 GO TO 10
```

This program is less efficient and slower than a copy program that uses string variables to copy files. But a copy program using string variables does not copy some characters and requires that carriage returns separate strings.

## 3.4.4 System Function 5 - Return to READY

Execution of a SYS(5) system function call deletes the program in memory, changes the program name to NONAME, and returns to READY. This is a useful method of terminating programs that are not to be rerun.

Example:

```
LIST

DELETE  17-MAR-75  MU BASIC/RT-11 V01-06

10 REM THIS PROGRAM WILL DELETE ITSELF
20 A=SYS(5)

READY

RUNNH


READY

LIST

NONAME  17-MAR-75  MU BASIC/RT-11 V01-06


READY
```

System function 5 is equivalent in effect to the SCR command.

See the BASIC-11 Language Reference Manual for a description of the SCR command.

### 3.4.5 System Function 6 - Terminal Margin

System function 6 sets the maximum number of characters that can be printed on one line. BASIC initially assumes a terminal margin of 72 and outputs a carriage return/line feed combination after printing 72 characters on a line. Execution of system function 6 with a nonzero second expression (between 1 and 255) causes BASIC to output a carriage return/line feed after printing the number of characters specified in the second expression. The margin also affects echoing - if more characters are entered than fit on a line, a carriage return/line feed is printed and the excess characters are echoed on the next line.

If a line is partially full and there is not enough room for an output string (or number), a carriage return/line feed combination is printed and the string (or number) is printed on the next line. An output string (or number) longer than a complete single line is continued on the next line. The process is repeated as many times as necessary to print the entire string (or number). Even if a line is partially filled when a system function 6 is executed, the margin is changed for that line.

NOTE

> BASIC outputs characters to a terminal buffer and continues program execution without waiting for the characters to actually be printed on the terminal. If a system function 6 changes the margin while there are characters in the buffer, a carriage return will be printed when the new margin is reached or exceeded. But BASIC tests if a string or number will fit on a partially filled line before outputting the characters to the buffer and this test is not affected by a later margin change.

Example:

```
10 A=SYS(6,10) \ REM SETS MARGIN TO  10
20 PRINT "1234567";"8901"
30 PRINT "THIS LINE WON'T FIT!"
RUNNH

1234567
8901
THIS LINE
WON'T FIT!

READY
```

With a margin of ten, the string "8901" does not fit on the first line of output, so it is printed on the second line. The string to be printed by line 120 will not fit on one line, so it is printed ten characters per line until the string is exhausted.

A system function 6 with a second expression equal to zero suppresses printing of any automatic carriage return/line feed combinations. This suppression is useful when preparing binary tapes or tapes of BASIC programs (whose lines are longer than the terminal width) on the low-speed paper tape punch or when using special cursor control characters available on some alphanumeric video display terminals. (Cursor control characters determine where on the video display screen the next printed character will appear.)

To set BASIC to use the full width of an LA36 DECwriter II (132 columns) type:

```
A = SYS(6,132)
```

To set BASIC to use the full width of a VT50 alphanumeric display terminal or LA30 (80 columns) type:

```
A = SYS(6,80)
```

To set BASIC to use the full width of a VT05 or ASR33 (72 columns) type:

```
A = SYS(6,72)
```

This returns BASIC to the initial default state.

### 3.4.6 System Functions 7 and 8 - CTRL/C Disable

The system function call 8 is a privileged function call that causes the CTRL/C key command to be ignored. If a nonprivileged user attempts to execute SYS(8), the ?PSF (Privileged System Function) error message is produced. A program should be thoroughly debugged before the SYS(8) function call is inserted.

The system function 7 can be executed by any user and it returns CTRL/C to its normal meaning. If CTRL/C is already enabled, a SYS(7) is ignored.

NOTE

If a program executes a SYS(8) function call and then enters a closed loop, it is not possible to halt the program. All other users are unaffected but it is not possible to use the terminal at which the error has occurred until the MU BASIC/RT-11 system is taken down and then brought up again.

### 3.4.7 System Function -1 - Set User ID

The SYS(-1, expr2) function sets the user ID to the letters specified by a positive ASCII value in expr2. This is a privileged system function call. To set the user ID to a value "AB" the following system function call could be executed:

```
A = SYS(-1, ASC("A")+256*ASC("B"))
```

Only the ASCII values of the characters A through Z and the digits 0 through 9 should be used for the user ID. The second character, however, can be a null. Use of any other values results in illegal RT-11 file descriptors.

If the value of expr2 is negative, the system function call can be executed by any user and it returns the current user ID.

Example:

```
10 A=SYS(-1,-1)
20 U$=CHR$(A)+CHR$(A/256)
30 PRINT "CURRENT USER ID IS ";U$
```

## 3.4.8 System Function -2 - Clear Privileged Status

A SYS(-2) function call clears the privileged status indicator for the current user. After execution of a SYS(-2) function it is impossible to execute a privileged system function or a privileged file operation. It is possible to regain the privileged status only by typing the BYE command and then logging on under a privileged user ID (see Chapter 1).

## 3.4.9 System Function -3 - Return to RT-11 Monitor

SYS(-3) is a privileged system function. It allows BASIC to be taken down to return control to the RT-11 monitor. SYS(-3) causes all users' programs to be terminated and erased and all files open for output to be deleted. After a SYS(-3) function call it is not possible to enter any commands or program lines to BASIC until BASIC is reloaded by the system manager. Consequently, all users should be notified, if possible, before execution of a SYS(-3) function call.

## 3.4.10 System Function -4 - Return Privilege Status

System function -4 returns the privilege status of the current user. This allows a privileged program (see section 2.3) to determine if the user running the program is privileged or nonprivileged. If the user is privileged a value of +1 is returned, but if the user is nonprivileged a value of 0 is returned. For example:

```
10 A=SYS(-4)
20 IF A=1 THEN 100
30 PRINT "ILLEGAL OPERATION FOR NONPRIVILEGED USER"
40 A=SYS(5) \ REM RETURN TO READY
100 REM DO RESTRICTED OPERATION HERE
```

# CHAPTER 4

# ERROR MESSAGES

When BASIC encounters an error, execution of the command or statement is interrupted and an error message is printed. Most errors are fatal and cause BASIC to print the READY message. The condition causing the error must be corrected before execution can be continued.

Certain arithmetic and input errors are nonfatal. BASIC substitutes a default value for a nonfatal arithmetic error and resumes execution. When data in an illegal format is entered in response to an INPUT statement, the request for input is repeated. Nonfatal errors do not cause BASIC to print the READY message.

BASIC detects errors when it executes commands, immediate mode statements, or program lines. Program lines that are typed are not checked for syntax errors until executed. No errors are produced when typing program lines or reading them from a file with these exceptions: ?LTL (Line Too Long), ?TLT (Too Long to Translate), ?PTB (Program Too Big), and ?SYN (SYNtax error - caused when program lines in a file contain illegal characters).

When a program is interrupted by an error, BASIC includes in the message the line number of the statement causing the error. It is often useful to list this line and examine the values of the variables in the line by an immediate mode PRINT statement.

The cause of the error can be corrected and execution of the program continued by the immediate mode GO TO statement. Execution of the program can also be started at the beginning by the RUN command. This will initialize all variables and delete any open files. To save the data in any open files, type an immediate mode CLOSE statement.

## Error Messages

All error messages are printed in one of two formats:

```
          message
```

or

```
          message AT LINE xxxxx
```

where xxxxx is the line number of the statement containing the error. Error messages produced by immediate mode statements or commands are printed in the first format.

Table 4-1 lists all BASIC error messages. The message produced is in the abbreviated form unless BASIC has been assembled with longer error messages specified. All error messages are fatal unless the explanation specifies nonfatal.

Table 4-1
BASIC Error Messages

| Abbreviated Form | Longer Form | Explanation |
|---|---|---|
| ?ADR | ADDRESS CHECK ERROR | Internal system error. If error is reproducible, system manager should submit SPR. |
| ?ARG | ARGUMENT ERROR | Arguments in a function do not match, in number, range, or type, the arguments defined for the function. |
| ?ATL | ARRAYS TOO LARGE | Not enough memory is available for the arrays specified in the DIM statements. If the array cannot be made smaller, then reduce the size of the program (see section 3.2). Alternatively, a virtual array file may be used instead of an array in memory. |
| ?BDR | BAD DATA READ | Illegal characters in data item input from a file or from a DATA statement. |
| ?BLG | BAD LOG | Nonfatal, expression in LOG or LOG10 function is zero or negative, BASIC returns a value of zero and continues execution. |

Table 4-1 (Cont.)
BASIC Error Messages

| Abbreviated Form | Longer Form | Explanation |
|---|---|---|
| ?BRT | BAD DATA - RETYPE FROM ERROR | Nonfatal, item entered in response to an INPUT or INPUT #Ø statement is in wrong format. Retype item and program will continue. |
| ?BSO | BUFFER STORAGE OVERFLOW | Not enough room available for file buffer in user area or system I/O area. Reduce program size (see section 3.2). |
| ?CAO | CHANNEL ALREADY OPEN | OPEN statement specifies a channel (logical unit number) which is already associated with an open file. |
| ?CHN | CHANNEL NUMBER FAULT | Internal system error. If error is reproducible, system manager should submit SPR. |
| ?CNO | CHANNEL NOT OPEN | A PRINT #, PRINT USING #, INPUT #, IF END #, or CLOSE statement specifies a channel (logical unit number) not associated with an open file. |
| ?CPE | CHANNEL POOL EMPTY | All device channels are currently in use. Caused by OPEN statement. Retry operation later. |
| ?DEV | NO DEVICE HANDLER | Handler is not currently available to BASIC users. Notify the system manager. If the handler is present on the system device and has been loaded correctly (if in a F/B environment) and error is reproducible, submit SPR. |
| ?DHE | DEVICE HARDWARE ERROR | A device hardware error has been detected. This is often caused by an off-line or write-locked device. If error is reproducible and not caused by an off-line device or by output to a write-locked device, notify the system manager. |
| ?DIR | DIRECTORY I/O ERROR | An error has been detected during an RT-11 directory operation. This is often caused by a write-locked device. If error is reproducible and not caused by a write-locked device, notify the system manager. |

Table 4-1 (Cont.)
BASIC Error Messages

| Abbreviated Form | Longer Form | Explanation |
|---|---|---|
| ?DRO | DIRECTORY OVERFLOW | There is no room in directory for a new file. Delete old files from device or use another device. |
| ?DNA | DEVICE NOT AVAILABLE | The device requested is currently in use by another user or another job (when the RT-11 Background/Foreground monitor is used). Use another device or try again later. This error message is also produced by an attempt to ASSIGN a public device. |
| ?DNE | DEVICE NOT ENABLED | Device is in LOCAL mode or not properly mounted. Enable device and retry, or use another device. |
| ?ENL | END NOT LAST | END statement is not the highest numbered program line. Caused when END statement is executed and a program line has a line number higher than the END statement line number. |
| ?ETC | EXPRESSION TOO COMPLEX | The expression being evaluated caused stack overflow because it is too complex. This is usually caused by user-defined functions or nested functions. The degree of complexity that produces this error varies according to the amount of space available in the stack at the time. Breaking the statement up into several simpler ones eliminates the error. |
| ?F-EMT<br>?F-OVL<br>?F-SYS<br>?F-TRP | none<br>none<br>none<br>none | Illegal EMT coded.<br>Overlay read error.<br>Fatal system error.<br>Trap to location 4 or 1Ø.<br><br>These are four fatal system errors. The message will be printed on the console terminal and will not include AT LINE xxxxx. Control will return to the RT-11 Monitor. The system manager must reload BASIC. If the error is reproducible and not due to a user programming error in an assembly language routine, submit SPR. |

Table 4-1 (Cont.)
BASIC Error Messages

| Abbreviated Form | Longer Form | Explanation |
|---|---|---|
| ?FAD | FUNCTION ALREADY DEFINED | The user-defined function is previously defined. |
| ?FTC | ERROR IN FETCH | Device contains a bad block or is offline. Notify your system manager if the device is not offline. If error is not due to an offline device or a bad block and is reproducible, submit SPR. |
| ?FNF | FILE NOT FOUND | The file requested is not on the specified device. |
| ?FOV | FLOATING OVERFLOW | Nonfatal, the absolute value of the result of a computation is greater than the largest number that may be stored by BASIC (approximately $10^{38}$). A value of zero is given to the expression and BASIC continues execution. |
| ?FPV | FILE PROTECTION VIOLATION | Restricted file operation has been attempted. |
| ?FRM | FORMAT ERROR | Format string error occurs in PRINT USING statement or an attempt was made to print item in the wrong type of data field. |
| ?FSV | NESTED FOR STATEMENTS WITH SAME CONTROL VARIABLE | A FOR statement is inside a FOR-NEXT loop that specifies the same control variable that the FOR statement specifies. |
| ?FTS | FILE TOO SHORT | The specified or default FILESIZE in a data file opened for output is not large enough to hold the data. The file is not closed and all data is lost. Specify larger FILESIZE in OPEN statement. If this error message is produced by a SAVE or REPLACE command, save the program on cassette, magtape, or paper tape (if available) and notify system manager. |

Table 4-1 (Cont.)
BASIC Error Messages

| Abbreviated Form | Longer Form | Explanation |
|---|---|---|
| ?FUN | FLOATING UNDERFLOW | Nonfatal; the absolute value of the result of a computation is smaller than the smallest number that BASIC can store (approximately $10^{-38}$). A value of zero is given to the expression and BASIC continues execution. |
| ?FWN | FOR WITHOUT NEXT | The program contains a FOR statement without a corresponding NEXT statement to terminate the loop. |
| ?FZD | FLOATING ZERO DIVIDE | Nonfatal; computation includes a division of some quantity by zero. The expression is given a value of zero and BASIC continues execution. |
| ?ICN | ILLEGAL CHANNEL NUMBER | The channel (logical unit number) specified is not in the range 1-127 or the IF END statement specifies a file on a terminal. |
| ?IDM | ILLEGAL DIM | A subscript in a DIM or COMMON statement is not an integer number or an array is dimensioned more than once. |
| ?IDT | ILLEGAL DATA TYPE | The statement assigns a value greater than 32,767 or less than -32,768 to an element in an integer virtual array file. |
| ?IFL | ILLEGAL FILE LENGTH | The FILESIZE specified in the OPEN statement exceeds the maximum size allowed. Error is also produced when FILESIZE specified is less than 1 for nonprivileged users (or less than -1 for privileged users). |
| ?IFS | ILLEGAL FILE SPECIFICATION | The file specification does not conform to the required syntax, or contains illegal characters. Legal characters are the letters A through Z, the digits Ø through 9, and the special symbols ".", ":", " " (blank), "$", "#", and "@" which must be used as described in section 2.2. |

Table 4-1 (Cont.)
BASIC Error Messages

| Abbreviated Form | Longer Form | Explanation |
|---|---|---|
| ?IID | ILLEGAL I/O DIRECTION | An attempt has been made to OPEN FOR OUTPUT a read-only device (high-speed paper tape reader) or to OPEN FOR INPUT a write-only device (line printer, high-speed paper tape punch). For example, the statement: OPEN "LP:" AS FILE 1 will generate this error message because FOR INPUT is assumed when neither is specified. |
| ?IIM | ILLEGAL IN IMMEDIATE MODE | The INPUT statement has been entered in immediate mode. |
| ?INS | ILLEGAL NUMBER OF SUBSCRIPTS | More than two subscripts are specified in a DIM or COMMON statement or the array is dimensioned with one subscript and referenced by two or vice versa. |
| ?IRS | ILLEGAL RECORD SIZE | The OPEN statement specifies a RECORDSIZE less than one for unprivileged users or specifies a RECORDSIZE of zero or less than minus one for a privileged user. |
| ?ISL | ILLEGAL STRING LENGTH | String virtual array OPEN statement specifies a string length outside the range 1-255. |
| ?LTL | LINE TOO LONG | An attempt has been made to enter a line longer than 132 characters; the line is ignored. If this message occurs when BASIC is reading a program from a file, BASIC stops reading the file. |
| ?MSP | MISSING SUBPROGRAM | Occurs when assembly language routines have been included with BASIC and a CALL statement specifies a nonexistent routine name. This error can also be caused by a syntax error in the first element of a line. BASIC interprets this error as an implied call statement. |

Table 4-1 (Cont.)
BASIC Error Messages

| Abbreviated Form | Longer Form | Explanation |
|---|---|---|
| ?NER | NOT ENOUGH FILE SPACE | The device contains insufficient free space to accommodate the requested file. Try again, and specify a smaller FILESIZE in the OPEN statement or a different device. This error message is also produced when one user attempts to open more than one file simultaneously on one cassette or magtape unit. If this message occurs when accessing a public device, notify the system manager. |
| ?NFS | NOT FILE STRUCTURED | An attempt has been made to open a virtual file on a device other than DECtape or disk. |
| ?NGS | NEGATIVE SQUARE ROOT | Nonfatal, the expression in the SQR (square root) function has a negative value. The square root of the absolute value of the expression is returned and BASIC continues execution of the program. |
| ?NOB | NUMBER OUT OF BOUNDS | The absolute value of a numeric constant specified in a statement or in the VAL function is less than the smallest number BASIC can store (approximately $10^{-38}$) or is greater than the largest number BASIC can store (approximately $10^{38}$). |
| ?NRH | NO ROOM FOR HANDLER | Currently the system I/O area has insufficient space for the non-resident handler of the requested device. If possible use another device or retry the operation again later. |
| ?NSM | NUMBERS AND STRINGS MIXED | String and numeric values appear in the same expression or they are set equal to each other; for example, A$=2. |
| ?NVD | NOT A VALID DEVICE | The device name is not valid or is not available to BASIC users. |
| ?NWF | NEXT WITHOUT FOR | A NEXT statement has been executed without a corresponding FOR statement. |

Table 4-1 (Cont.)
BASIC Error Messages

| Abbreviated Form | Longer Form | Explanation |
|---|---|---|
| ?OOD | OUT OF DATA | The data list has been exhausted and a READ statement requests additional data or the end of a file has been reached and the INPUT # statement requests additional data. |
| ?PSF | PRIVILEGED SYSTEM FUNCTION | A nonprivileged user has attempted to execute a privileged system function call. |
| ?PTB | PROGRAM TOO BIG | The line just entered causes the program to exceed the user code area; the line is ignored. Reduce program size (see section 3.2). If this error occurs when BASIC is reading a program from a file, BASIC stops reading the file. |
| ?RPL | USE REPLACE | An attempt has been made to save a program in a file that already exists on the device. The operation does not occur and the original file is not disturbed. Use the REPLACE command if the operation is intended. |
| ?RWG | RETURN WITHOUT GOSUB | A RETURN is encountered before execution of a GOSUB statement. |
| ?SOB | SUBSCRIPT OUT OF BOUNDS | The subscript computed is less than zero or is outside the bounds defined in the DIM statement or outside of the limits of the virtual array file. |
| ?SSO | STRING STORAGE OVERFLOW | Not enough memory is available to store all the strings used in the program. Reduce program size (see section 3.2). |
| ?STL | STRING TOO LONG | The maximum length of a string in a BASIC statement is 255 characters. |

Table 4-1 (Cont.)
BASIC Error Messages

| Abbreviated Form | Longer Form | Explanation |
|---|---|---|
| ?SYN | SYNTAX ERROR | BASIC has encountered an unrecognizable element. Common examples of syntax errors are misspelled commands, unmatched parentheses, and other typographical errors. This message can also be produced by attempting to read in a program from a file containing illegal characters in which case BASIC stops reading the file. |
| ?TLT | TOO LONG TO TRANSLATE | Lines are translated as they are entered and the line just entered exceeds the area reserved for translating; the line is ignored. If this message is produced while BASIC is reading a program from a file, BASIC stops reading the file. |
| ?TMC | TOO MANY CHANNELS | OPEN statement exceeds the maximum number of files that may be opened simultaneously by a nonprivileged user. |
| ?TMG | TOO MANY GOSUBS | More than twenty GOSUBs have been executed without a corresponding RETURN statement. |
| ?UFN | UNDEFINED FUNCTION | A user-defined function has been used and not defined. |
| ?ULN | UNDEFINED LINE NUMBER | The line number specified in an IF, GO TO, GOSUB, ON GO TO, ON GOSUB, CHAIN, or OVERLAY statement does not exist anywhere in the program. |
| ?USR | ILLEGAL USR/EXIT CALL | Internal system error. If error is reproducible, system manager should submit SPR. |
| ?↑ER | ↑ERROR | The program has tried to compute A↑B, where A is less than Ø and B is not an integer. This would produce a complex number which can not be represented in BASIC. This message is also produced when A is less than zero and B is an integer with an absolute value greater than 255. |

## Error Messages

BASIC functions that are called improperly cause error messages to be printed. Table 4-2 describes under what conditions BASIC functions produce errors.

Table 4-2
Error Conditions in BASIC Functions

| Function | Condition | Error Message |
|---|---|---|
| All functions | The argument used is the wrong type. For example, the argument is numeric and the function expects a string expression. | ?ARG |
| All functions | The wrong number of arguments has been used in a function, or the wrong character has been used to separate them. For example, PRINT SIN (X,Y) produces a syntax error because the SIN function has only one argument. | ?SYN |
| ASC(string expr) | String expr is not a one character string. | ?ARG |
| BIN(string expr) | Character other than blank, zero, or one in string or value is greater than $2^{16}$. | ?ARG |
| CHR$(expr) | Expr is not in the range Ø-32767. | ?ARG |
| EXP(expr) | Expression is greater than 87. | ?↑ER |
| FNletter | The function FNletter has not been defined (function cannot be defined by an immediate mode statement). | ?UFN |
| LOG(expr) | Expression is negative or Ø. BASIC returns a value of Ø. | ?BLG |
| LOG1Ø(expr) | Expression is negative or Ø. BASIC returns a value of Ø. | ?BLG |
| OCT(string expr) | Character other than blank or digits Ø-7 in string or value is greater than $2^{16}$. | ?ARG |
| PI | An argument is included. | ?ARG |
| SEG$(string expr, expr1,expr2) | No additional error conditions. | |
| SQR(expr) | Expression is negative; BASIC returns the square root of the absolute value of the expression. | ?NGS |

Table 4-2 (Cont.)
Error Conditions in BASIC Functions

| Function | Condition | Error Message |
|---|---|---|
| SYS(expr1[,expr2]) | A nonprivileged user has attempted to execute a privileged system function call. | ?PSF |
| | The value of the first expression is less than -4 or greater than 8 or a second expression is specified when none is expected. | ?ARG |
| TAB | Expression is not in the range Ø-32767. | ?ARG |
| VAL(string expr) | String expr is not a numeric constant. | ?ARG |

# APPENDIX A

# ASCII CHARACTER SET

The following table shows, with the corresponding octal and decimal codes, the 128-character ASCII (American Standard Code for Information Interchange) character set. These codes are used to store ASCII data in files and to store them internally.

The BASIC user can convert an ASCII value to the corresponding string character with the CHR$ function and can convert a string character to the corresponding ASCII value with the ASC function. These functions are described in the BASIC-11 Language Reference Manual. A special system function (see section 3.4.3) returns the ASCII value of characters input from the terminal or a file.

BASIC also uses the ASCII values of the characters in string comparisons. See the BASIC-11 Language Reference Manual for a description of string relational operators.

BASIC converts to upper case all lower case letters entered at the terminal. No conversion is done on terminal output or input and output with any other device.

The octal code is provided for reference. BASIC does not support octal numbers except through the OCT function (see the BASIC-11 Language Reference Manual).

ASCII characters are stored internally and in files in 8 bits. The eighth (high order) bit is normally zero.

## ASCII Character Set

Table A-1
ASCII Character Set

| ASCII Decimal Code | ASCII 7-Bit Octal Code | Character |
|---|---|---|
| 0 | 000 | NUL (CTRL/@) |
| 1 | 001 | SOH (CTRL/A) |
| 2 | 002 | STX (CTRL/B) |
| 3 | 003 | ETX (CTRL/C) |
| 4 | 004 | EOT (CTRL/D) |
| 5 | 005 | ENQ (CTRL/E) |
| 6 | 006 | ACK (CTRL/F) |
| 7 | 007 | BEL (CTRL/G) |
| 8 | 010 | BS (CTRL/H) |
| 9 | 011 | HT (CTRL/I or TAB) |
| 10 | 012 | NL (NEW LINE or LINE FEED) |
| 11 | 013 | VT (Vertical TAB) |
| 12 | 014 | FF (Form Feed) |
| 13 | 015 | RT (Return) |
| 14 | 016 | SO (CTRL/N) |
| 15 | 017 | SI (CTRL/O) |
| 16 | 020 | DLE (CTRL/P) |
| 17 | 021 | DC1 (CTRL/Q) |
| 18 | 022 | DC2 (CTRL/R) |
| 19 | 023 | DC3 (CTRL/S) |
| 20 | 024 | DC4 (CTRL/T) |
| 21 | 025 | NAK (CTRL/U) |
| 22 | 026 | SYN (CTRL/V) |
| 23 | 027 | ETB (CTRL/W) |
| 24 | 030 | CAN (CTRL/X) |
| 25 | 031 | EM (CTRL/Y) |
| 26 | 032 | SUB (CTRL/Z) |
| 27 | 033 | ESC (ALTMODE) |
| 28 | 034 | FS (CTRL/\) |
| 29 | 035 | GS (CTRL/]) |
| 30 | 036 | RS (CTRL/^) |
| 31 | 037 | US (CTRL/_) |
| 32 | 040 | SP (space bar) |
| 33 | 041 | ! |
| 34 | 042 | " |
| 35 | 043 | # |
| 36 | 044 | $ |
| 37 | 045 | % |
| 38 | 046 | & |
| 39 | 047 | ' |
| 40 | 050 | ( |
| 41 | 051 | ) |
| 42 | 052 | * |
| 43 | 053 | + |
| 44 | 054 | , |
| 45 | 055 | - |
| 46 | 056 | . |
| 47 | 057 | / |
| 48 | 060 | 0 |
| 49 | 061 | 1 |
| 50 | 062 | 2 |
| 51 | 063 | 3 |
| 52 | 064 | 4 |
| 53 | 065 | 5 |

Table A-1 (Cont.)
ASCII Character Set

| ASCII Decimal Code | ASCII 7-Bit Octal Code | Character |
|---|---|---|
| 54 | 066 | 6 |
| 55 | 067 | 7 |
| 56 | 070 | 8 |
| 57 | 071 | 9 |
| 58 | 072 | : |
| 59 | 073 | ; |
| 60 | 074 | < |
| 61 | 075 | = |
| 62 | 076 | > |
| 63 | 077 | ? |
| 64 | 100 | @ |
| 65 | 101 | A |
| 66 | 102 | B |
| 67 | 103 | C |
| 68 | 104 | D |
| 69 | 105 | E |
| 70 | 106 | F |
| 71 | 107 | G |
| 72 | 110 | H |
| 73 | 111 | I |
| 74 | 112 | J |
| 75 | 113 | K |
| 76 | 114 | L |
| 77 | 115 | M |
| 78 | 116 | N |
| 79 | 117 | O |
| 80 | 120 | P |
| 81 | 121 | Q |
| 82 | 122 | R |
| 83 | 123 | S |
| 84 | 124 | T |
| 85 | 125 | U |
| 86 | 126 | V |
| 87 | 127 | W |
| 88 | 130 | X |
| 89 | 131 | Y |
| 90 | 132 | Z |
| 91 | 133 | [ |
| 92 | 134 | \ |
| 93 | 135 | ] |
| 94 | 136 | ↑ or ^ |
| 95 | 137 | ← or _ |
| 96 | 140 | ` |
| 97 | 141 | a |
| 98 | 142 | b |
| 99 | 143 | c |
| 100 | 144 | d |
| 101 | 145 | e |
| 102 | 146 | f |
| 103 | 147 | g |
| 104 | 150 | h |
| 105 | 151 | i |
| 106 | 152 | j |
| 107 | 153 | k |

Table A-1 (Cont.)
ASCII Character Set

| ASCII Decimal Code | ASCII 7-Bit Octal Code | Character |
|---|---|---|
| 108 | 154 | l |
| 109 | 155 | m |
| 110 | 156 | n |
| 111 | 157 | o |
| 112 | 160 | p |
| 113 | 161 | q |
| 114 | 162 | r |
| 115 | 163 | s |
| 116 | 164 | t |
| 117 | 165 | u |
| 118 | 166 | v |
| 119 | 167 | w |
| 120 | 170 | x |
| 121 | 171 | y |
| 122 | 172 | z |
| 123 | 173 | { |
| 124 | 174 | \| |
| 125 | 175 | } |
| 126 | 176 | ~ |
| 127 | 177 | RUBOUT |

# APPENDIX B

# SUMMARY OF BASIC STATEMENTS, FUNCTIONS, AND COMMANDS

This appendix summarizes the statements, functions, and commands in MU BASIC/RT-11. These summaries supersede the summaries provided in the BASIC-11 Language Reference Manual.

## B.1 DOCUMENTATION CONVENTIONS

Certain conventions are used to describe the format of the BASIC language throughout this document. Tables B-1 and B-2 describe these conventions. See the BASIC-11 Language Reference Manual for a more complete description of the terms used.

Table B-1
Documentation Conventions

| Convention | Meaning |
|---|---|
| Items in lower-case letters | Elements to be supplied by user according to rules explained in the text. Table B-2 provides a description of some frequently used elements and abbreviations. |
| Items in capital letters and special symbols | BASIC keywords, must appear exactly as shown because they form the vocabulary of the BASIC-11 language. For example, LET, IF, OPEN, RUN, #, and /. |
| Braces | A choice of one element among two or more possibilities, for example: $\left\{\begin{array}{l}\text{THEN statement}\\\text{THEN line number}\\\text{GO TO line number}\end{array}\right\}$ |

Table B-1 (Cont.)
Documentation Conventions

| Convention | Meaning |
|---|---|
| Square brackets [] | Optional element or a choice among optional elements, for example:<br>[LET] variable = expression<br>OPEN string $\begin{bmatrix}\text{FOR INPUT}\\\text{FOR OUTPUT}\end{bmatrix}$ AS FILE #expr |
| Ellipsis . . . | Preceding elements may, at the user's option, be repeated, for example:<br>CLOSE #expr1, #expr2, #expr3, . . .<br>DEF FNletter (var1[,var2,...,var5])=expr |

Lower-case words that appear in a description of the format of a BASIC statement, function, or command represent elements that must be supplied by the user according to the rules provided. When one of these words is repeated a number or letter appendage serves to identify each separate element in the explanations. Table B-2 describes the meaning and abbreviations used for some elements.

Table B-2
Lower-Case Words Used in Format Descriptions

| Word | Abbreviation | Meaning |
|---|---|---|
| expression | expr | Any valid BASIC expression. |
| string expression | string expr | Any valid BASIC string expression. |
| integer | int | Any positive integer (must be a numeric constant). |
| line number | | Any legal line number. Must be a one to five digit number within the range 1 to 32,767. |
| relational operator | rel-op | An arithmetic or string relational operator. |
| variable | var | A name representing a numeric or string variable. |

## B.2 SUMMARY OF BASIC STATEMENTS

The following summary of BASIC statements defines the general format of each statement and gives a brief explanation of its use. Square brackets [] indicate optional elements.

[CALL "]routine name["] (argument list)

Used to call assembly language routines from a BASIC program. The word CALL and the pair of quotation marks should either both be excluded or both be included.

CHAIN string [LINE line number]

Terminates execution of the program, loads the program specified by string, and begins execution at the lowest line number or, when a line number is present in the CHAIN statement, at the specified line number. The string may be any string expression and is a file descriptor as described in section 2.2.

CLOSE [[#]expr1, [#] expr2, [#] expr3, ...,VFint1,VFint2,...]

Closes the file(s) associated with the logical unit number(s) and virtual file logical unit number(s) specified. If no logical unit number is specified, closes all open files.

COMMON var1[(int1[,int2])], var2[(int1[,int2])],...

Preserves values and names of specified variables and arrays when the CHAIN statement is executed. Both string and arithmetic variables and arrays can be passed. The statement also dimensions the specified arrays.

DATA number [,"string",number,...]

Used in conjunction with READ to input listed data into an executing program. Can contain any mixture of strings and numbers.

DEF FNletter (var1[,var2,...,var5])=expression

Defines a user function. Letter may be any single letter A through Z.

DIM var(int1[,int2]),var2(int1[,int2]),...

Reserves space in memory for arrays according to the subscript(s) specified after the variable name.

END Optional, placed at the physical end of the program to terminate execution.

FOR var = expr1 TO expr2 [STEP expr3]

Sets up a loop to be executed the specified number of times.

GOSUB line number

Unconditionally transfers control to specified line of subroutine.

GO TO line number

Unconditionally transfers control to specified line number.

IF expr1 rel-op expr2 $\left\{\begin{array}{l}\text{THEN statement}\\ \text{THEN line number}\\ \text{GO TO line number}\end{array}\right\}$

Conditionally executes the specified statement or transfers control to specified line number. When the condition is not satisfied, execution continues at the next sequential line. The expressions and the relational operator must all be string or all be numeric.

IF END #expr $\left\{\begin{array}{l}\text{THEN statement}\\ \text{THEN line number}\\ \text{GO TO line number}\end{array}\right\}$

Tests for end-of-file condition of input sequential file associated with logical unit expr.

INPUT [#expr$\left\{\begin{array}{l},\\:\end{array}\right\}$ ] var1[,var2,...]

Inputs data from the file associated with the logical unit specified by expr or from the user's terminal. Variables may be arithmetic or string. #expr can be followed by a comma or a colon.

KILL string expr

Deletes file specified by string expr.

[LET] variable = expression

Assigns value of expression to the specified variable. Variable and expression must be of the same type - either numeric or string.

[LET] VFinteger(expr1) = expr2

Assigns value of expr2 to the expr1 element of the virtual file VF integer. The data type of the virtual file and of expr2 must be the same - either numeric or string.

LINPUT [#expr$\left\{\begin{array}{l},\\:\end{array}\right\}$] var1[,var2,...]

Equivalent to INPUT (for compatibility only).

NAME string expr1 TO string expr2

Renames file specified by string expr1 to name specified by string expr2.

NEXT variable

Placed at end of FOR loop to return control to FOR statement.

ON expression GOSUB line number1 [, line number2, line number3,...]

Conditionally transfers control to subroutine at one line number specified in list. Value of expression determines the line number to which control is transferred.

```
ON expression GO TO line number1 [, line number2,line number3,...]
```

Conditionally transfers control to one line number in the list. Value of expression determines the line number to which control is transferred.

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

Opens a file specified by string for input or output as specified (assumes input if neither specified) and associates file with the logical unit expr1. String expr is a file descriptor as described in section 2.2.

```
            [FOR INPUT ]                [%]
OPEN string [FOR OUTPUT] AS FILE VF int [$] [(expr1)][=expr2][,FILESIZE expr3]
```

Opens a virtual array file specified by string. FOR OUTPUT creates a new file; FOR INPUT (or neither) allows either output or input to an existing file. Elements may be assigned a value by the LET VF statement. When % (percent sign) is specified, the data type is integer; when $ (dollar sign) is specified the data type is string; and when neither is specified, the data type is floating point. expr1 specifies the dimension and expr2 specifies the string length (used with $ only).

```
OVERLAY string expr [LINE line number]
```

Overlays or merges the program currently in memory with the program in the file specified by string, and when overlay is completed, transfers control to either the next sequential BASIC line number or the line number specified. String expr is a file descriptor as described in section 2.2

```
             {,}
PRINT [#expr {:} ][expr1,expr2,expr3,...]
```

Prints values of expressions on the terminal or, when specified, to the file associated with logical unit expr. Expressions can be numeric and string. The TAB function can also be included. Elements can be separated by either commas or semicolons. #expr can be followed by a comma or a colon.

```
             {,}
PRINT [#expr {:} ] USING string, [expr1,expr2,expr3,...]
```

Prints values of expressions on the terminal or, when specified, to the file associated with logical unit expr in the format determined by string. Both numeric and string expressions can be used. Elements must be separated by commas.

```
RANDOMIZE
```

Causes the random number generator (RND function) to produce different random numbers every time the program is run.

```
READ var1[,var2, var3,...]
```

Assigns values listed in DATA statements to specified variables. Variables may be string or numeric.

REM [comment]

No effect on execution of program. Contains explanatory comments in a BASIC program.

RESET [[#]expr]

Equivalent to RESTORE.

RESTORE [[#]expr1[,[#]expr2,[#]expr3,...]]

Resets either the data pointer or, when specified, the input file(s) associated with logical unit number(s) specified to the beginning. File(s) must be on file structured devices.

RETURN

Terminates a subroutine and returns control to the statement following the last executed GOSUB statement.

STOP

Terminates execution of the program. Placed at logical end(s) of the program.

## B.3 SUMMARY OF BASIC FUNCTIONS

### ARITHMETIC FUNCTIONS

The following functions perform standard mathematical operations in BASIC.

| Name | Explanation |
|---|---|
| ABS(expr) | Returns the absolute value of expr. |
| ATN(expr) | Returns the arctangent of expr as an angle in radians in the range + or - pi/2. |
| COS(expr) | Returns the cosine of expr radians. |
| EXP(expr) | Returns the value of e↑expr where e is (approximately) 2.71828. |
| INT(expr) | Returns the greatest integer less than or equal to expr. |
| LOG(expr) | Returns the natural logarithm of expr. |
| LOG10(expr) | Returns the base 10 logarithm of expr. |
| PI | Returns the value of pi = 3.141593 (approximately). |
| RND[(expr)] | Returns a random number between 0 and 1. |
| SGN(expr) | Returns a value indicating the sign of expr. |

| Name | Explanation |
|---|---|
| SIN(expr) | Returns the sine of expr radians. |
| SQR(expr) | Returns the square root of expr. |
| TAB(expr) | Causes the terminal type head to tab to column number expr (valid only in PRINT statements). |
| SYS(expr1[,expr2]) | Special system function calls; control terminal input/output and perform special functions. |

| When expr1 is: | System Function |
|---|---|
| 0 | Cancels effect of a CTRL/O typed at terminal. |
| 1 | Disables echoing and enters special mode for inputting tapes on the low speed reader. |
| 2 | Enables echoing; cancels effect of SYS(1) and SYS(3). |
| 3 | Disables echoing. |
| 4 | Returns the ASCII value of the next character typed at the terminal; if specified, expr2 is the logical unit number associated with the file from which the next character should be read. |
| 5 | Deletes current program; changes program name to NONAME; and returns to the READY message. |
| 6 | Set the terminal margin to the value of expr2. |
| 7 | Enables CTRL/C as interrupt. |
| 8 | Privileged; disables CTRL/C as interrupt. |
| -1 | Privileged function; sets the user ID to that specified by the ASCII value in expr2; if expr2=ASC("X")+256*ASC("Y") then the ID will be "XY". Unprivileged if expr2 is negative, in which case it returns current user ID. |
| -2 | Clears privileged user bit; user will not be able to execute any privileged functions or file operations. |
| -3 | Privileged function; returns to RT-11 Monitor. |
| -4 | Returns privilege status; zero for nonprivileged and one for privileged. |

STRING FUNCTIONS

The string functions are:

| | |
|---|---|
| ASC(string expr) | Returns as a decimal number the 8-bit internal code (ASCII value) for the 1-character string expr. |
| BIN(string expr) Conv bin t dec | Converts a string expression containing a binary number to a decimal value. Blanks are ignored. |
| CHR$(expr) | Generates a 1-character string whose ASCII value is the low-order 8 bits of the integer value of expr. |
| DAT$ | Returns the date as a string in the form dd-mon-yr (for example Ø7-FEB-75). |
| LEN(string expr) | Returns the number of characters in the string expr. |
| OCT(string expr) CONV OCT TO DEC | Converts a string expression containing an octal number to a decimal value. Blanks are ignored. |
| POS (string expr1, string expr2, expr) | Searches for and returns the position of the first occurrence of string expr2 in string expr1. The search starts at the expr character position in string expr1. |
| SEG$(string expr,expr1,expr2) | Returns the string of characters in positions expr1 through expr2 in string expr. |
| STR$(expr) | Returns the string which represents the numeric value of expr. |
| TRM$(string expr) | Returns string expr without trailing blanks. |
| VAL(string expr) | Returns the value of the decimal number contained in the string expr. |

## B.4 SUMMARY OF BASIC COMMANDS

| Command | Explanation |
|---|---|
| APPEND [file descriptor] | Merges the program in core with the program specified by the file descriptor. |
| ASSIGN device: | Assigns specified device to the user if it is available. |
| BYE | Terminates the session of the user issuing the command; deletes all open output files and deassigns all devices. |
| CLEAR | Initializes all variables to zero, and all string variables to nulls and deletes arrays. |
| DEASSIGN[device:] | Deassigns the specified device or all assigned devices. |

## Summary of BASIC Statements, Functions, and Commands

| Command | Explanation |
|---|---|
| HELLO | Special command to get started with BASIC. |
| KEY | Enables echoing after TAPE command or SYS(1) or SYS(3) function call. |
| LENGTH | Displays on the terminal the size of the program in memory and the size of the remaining free memory. |
| LIST[NH] [line number] [-line number / -END] | Prints on the terminal the specified line(s) of the program currently in memory. NH suppresses the printing of the header line. |
| NEW[program name] | Erases the entire storage area of user and sets the current program name to the one specified. |
| OLD[file descriptor] | Erases the entire storage area of user and inputs the program from the specified file. |
| RENAME [program name] | Changes the current program name to the one specified. |
| REPLACE [file descriptor] | Replaces the specified file with the current program. |
| RUN[NH] | Executes the program in memory. NH suppresses the printing of the header line. |
| RUN[NH] file descriptor | Erases the entire storage area of user, inputs the program from the specified file, and then executes the program. Does not print header line in any case. |
| SAVE[file descriptor] | Outputs the program in memory as the specified file (can be used to list a program on the line printer or punch it on the high-speed paper tape punch). |
| SCR[ATCH] | Erases the user's entire storage area, but preserves the program name. |
| SET TTY type | Sets system to allow different terminals; type may be VT05, ASR33, or LA30. |
| TAPE | Disables echoing for entering tapes from the low-speed reader. |
| UNSAVE[file descriptor] | Deletes specified file. |

Key Commands

| Key | Explanation |
|---|---|
| CTRL/C | Interrupts execution of a command or program and causes BASIC to print the READY message. The execution of SYS(8) function call disables CTRL/C. Echoes as "↑C". |
| CTRL/O | Causes all further terminal output to be discarded. If an INPUT statement is encountered, a SYS(Ø) function call is executed, or CTRL/O is retyped, printing resumes. Occasionally CTRL/O suppresses the printing of the READY message. Echoes as "↑O". |
| CTRL/Q | Continues output to the terminal; cancels effect of CTRL/S. May be XON on some terminals. Does not echo. |
| CTRL/S | Temporarily suspends all output to terminal until CTRL/Q is typed; allows alphanumeric display terminals (VT05) to be read or photographed before data is moved off screen. May be XOFF key on some terminals. Does not echo. |
| CTRL/U | Deletes the entire current input line (provided the RETURN key has not been typed). BASIC displays ↑U at the end of the line. For example: `10 BLET A ↑U` ↑ CTRL/U typed here. |
| RUBOUT | Deletes the last character typed and echoes as a backarrow (underscore on some terminals) on the terminal. For example, `FOR N = 3←1 TO 3` ↑ RUBOUT typed here. RUBOUT can be repeated to delete any character up to the beginning of the line. Spaces are considered valid characters and are deleted by RUBOUTs. |

# APPENDIX C

# VIRTUAL ARRAY FACILITY

The virtual array facility allows a BASIC program to operate on data structures that are too large to be accommodated in memory at one time. To accomplish this, BASIC uses the disk file system for storage of data arrays, and maintains only portions of these files in memory at any given time.

An essential difference between real arrays and their virtual counterparts is the time required to reference array elements. In real arrays, the referencing order has no effect on the time required to reference an element. In virtual arrays, this order can have a significant effect on the program execution time. This appendix describes the algorithms used in the virtual array processor, so that users may optimize their use of this facility.

Each MU BASIC/RT-11 disk file is a contiguous sequence of 256-word records. Any position in a file can be specified internally with a 2-component address; the first component is the relative block within the file, the second is the position of the item within the block. One of the functions of the virtual array processor is to transform each virtual array reference into its corresponding file address. This is called mapping.

Virtual arrays are stored as unformatted binary data. This means that no I/O conversions need be performed in storing or retrieving elements in virtual storage. Thus, there is no loss of precision in these arrays, and no time wasted performing conversions.

All references to virtual arrays are ultimately located via file addresses relative to the start of the file. No symbolic information concerning dimensions or data types is stored within the file. Thus, different programs may use different data types to refer to the data contained within a single virtual array file. The user must be cautious in such operations, since it is the user's responsibility to ensure that all programs referencing a given set of virtual arrays are referencing the correct data. Consider the following example:

Program ONE contains

```
10  OPEN "FILE" AS FILE VF1%
     .
     .
```

Program TWO contains

```
10  OPEN "FILE" AS FILE VF2
```

Whenever program TWO references the array VF2, it is using the data known to program ONE as array VF1. VF2 contains floating-point data while VF1 contains integer data. These two arrays do not correspond in data type and the data program ONE creates may be meaningless to program TWO and vice versa.

NOTE

A virtual file should not be simultaneously opened under two or more different channels (by one user or two) and have data changed on more than one channel. For example:

```
50  OPEN "VALUES" AS FILE VF1 (100)
60  OPEN "VALUES" AS FILE VF2 (100)
70  VF1(1) = 10
80  VF2(2) = 20
```

Only one of these two assignment statements is effective. The other value is lost, because two buffers have been created and the last buffer to be written out destroys the changed data in the first buffer written out.

## C.1 ARRAY STORAGE

Strings in virtual storage occupy pre-allocated (determined by OPEN statement) space in the virtual file, and thus differ from strings in memory storage, where space is allocated dynamically. A virtual file containing strings can be considered to be a succession of fields, each of the maximum string length. When a string in a virtual file is assigned a new value, it is stored left-justified in the appropriate field. If the new string is shorter than the maximum length, the remainder of the field is filled with null characters. When the string is retrieved, the trailing null characters are removed.

Table C-1 describes the number of elements of each data type that can be stored in one 256-word (512-byte) block.

Table C-1
Virtual Array Storage Capabilities

| Data Type | Number Bytes per Element | Number Elements per Block |
|---|---|---|
| Floating Point | 4 | 128 |
| Integer (%) | 2 | 256 |
| String ($) | 32 | 16 |
| String ($=string length) | string length | 512/string length |

The string length must be in the range 1-255. The number of string elements per block can be fractional.

## C.2 TRANSLATION OF ARRAY SUBSCRIPTS INTO FILE ADDRESSES

In order to translate an array subscript into a file address, BASIC computes the relative distance from the specified item to the first item in the array. This is computed from the array subscript and the number of elements per block, as shown in Table C-1. The relative distance is added to the starting address of the file (determined by the OPEN statement processor) to find the address of the block containing the item.

Since the OPEN statement contains the only information used to define the structure of a file, it is possible for the user to specify different accessing arrangements for the same file in one or more programs. For example, the user can reference the same data as a series of either 32-byte strings or 16-byte strings, with the following statements:

```
10  OPEN "ABC" AS FILE VF1$ (1ØØØ) = 16
30  OPEN "ABC" AS FILE VF2$ (5ØØ) = 32
```

The user should keep in mind that in MU BASIC/RT-11, as in most BASICs, array subscripts begin with 0, not 1. An array with dimension n actually contains n+1 elements.

## C.3 ACCESS TO DATA IN VIRTUAL ARRAYS

Only a portion of a virtual array is in memory at any given time. This data is transferred directly between the device and an I/O buffer, created when the OPEN statement is executed. This buffer must be 256 words (one block) long, and may not be specified as several blocks with the RECORDSIZE or DOUBLE BUF option in the OPEN statement. For each virtual array file, BASIC notes

the block of the file currently in the buffer

whether the data in the buffer has been modified since it was read into memory

After BASIC translates a virtual array address into a file address, it checks whether the block that contains the referenced item is currently in the buffer. If the necessary block is present the reference proceeds; but if not, another portion of the file is read into the buffer. If the current data in the buffer has been altered, it is necessary to rewrite this data on the device prior to reading new data into the buffer.

The virtual array accessing algorithm is flow charted in Figure C-1. Users should design the order in which virtual file elements are referenced in their programs to minimize the reading and writing of blocks.

## C.4 ALLOCATING DEVICE STORAGE TO VIRTUAL FILES

The FILESIZE or dimensions indicated in an OPEN statement set maximum allowable values for subscripts and are used to compute the initial size of the virtual file to be allocated on the device. The contents are not initialized to zero. The data previously recorded in a block (when it was part of another file) is available to the new owner of

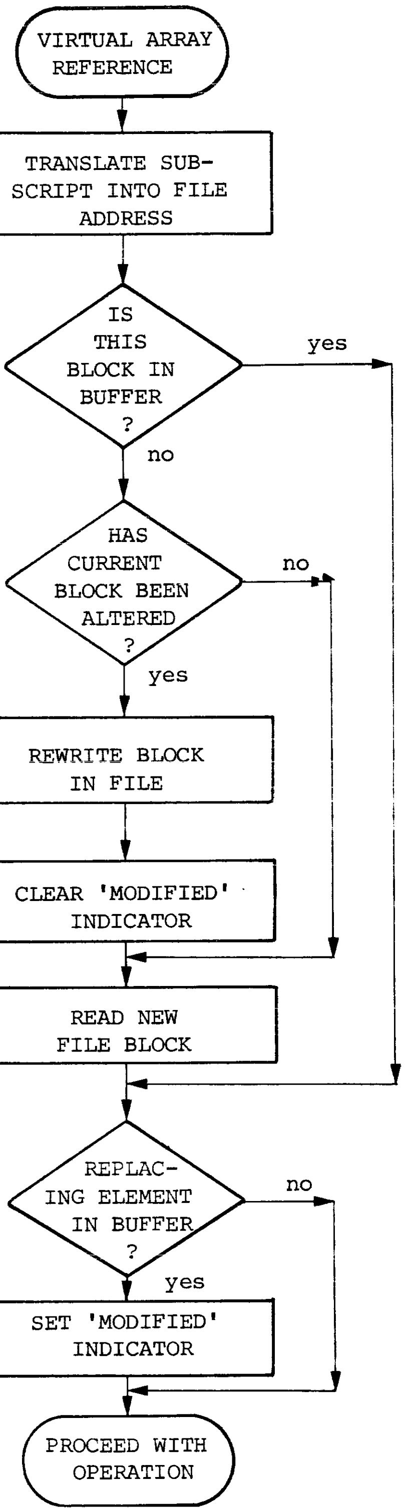

VIRTUAL ARRAY REFERENCE
TRANSLATE SUB-SCRIPT INTO FILE ADDRESS
IS THIS BLOCK IN BUFFER ?
yes
no
HAS CURRENT BLOCK BEEN ALTERED ?
no
yes
REWRITE BLOCK IN FILE
CLEAR 'MODIFIED' INDICATOR
READ NEW FILE BLOCK
REPLACING ELEMENT IN BUFFER ?
no
yes
SET 'MODIFIED' INDICATOR
PROCEED WITH OPERATION

Figure C-1
Virtual Array Accessing Algorithm

the block. Users whose files contain confidential information should explicitly overwrite all data in such files, prior to file deletion, in order to protect data contained therein.

# APPENDIX D

## MU BASIC/RT-11 PROGRAM STRUCTURE

MU BASIC/RT-11 stores each user's program in memory in the following format:

| | |
|---|---|
| Arrays | high address |
| Buffers | |
| Strings | |
| Symbol Table | |
| User Code | low address |

The symbol table and user code area are created when the program is entered. When the RUN command is given, the user program is scanned and arrays are set up. The string area is created during program execution.

The SCRatch command clears all the user code, symbol table, strings, and arrays from memory. The CLEAR command clears the arrays and strings but does not affect the user code or symbol table.

A symbol table entry is created for each distinct line number (four bytes) or variable name (ten bytes) referenced in the program. These entries are not deleted, however, even when all references in the program to a particular line number or variable are removed. Thus, if the program in memory is heavily modified, it may be desirable to save it with the SAVE command and then restore the program with the OLD command to obtain the largest possible user area.

User-entered blanks that are not in REM statements or string constants and blanks produced by BASIC when listing or saving a program do not contribute to the size of the program in memory. The total amount of memory storage required to store a BASIC program (user code and symbol table) depends on the parameters (2 bytes = 1 word) listed in Table D-1.

Table D-1
Parameters in Memory Storage of BASIC Programs

| Parameter | Contribution (bytes) | Definition |
|---|---|---|
| L | 7*L | Total number of lines in the BASIC program. |
| T | T | Total number of tokens in the program (see below for a description of BASIC tokens). |
| V | 10*V | Total number of distinct variable names used in the program (in this context, a scalar and an array variable with the same name are considered to be the same variable name). |
| R | 2*R | Total number of occurrences of variable names and references to line numbers in the program (not including the line number at the beginning of each line). |
| I1 | 2*I1 | Total number of integer constants whose absolute value is in the range $0 \leq x \leq 255$. |
| I2 | 3*I2 | Total number of integer constants whose absolute value is in the range $255 < x \leq 32767$. |
| F | 5*F | Total number of noninteger numeric constants and integer constants not in above ranges. |
| N | 10*N | Total number of NEXT statements in the program. |
| U | 2*U | Total number of references to the name of a user-defined function; e.g., FNA (including the definition itself). |
| S | 2*S | Total number of REM statements, implied CALL statements, and string constants. |

Table D-1 (Cont.)
Parameters in Memory Storage of BASIC Programs

| Parameter | Contribution (bytes) | Definition |
|---|---|---|
| C | C | Total number of characters in all REM and implied call statements and string constants (number of characters following REM, number of characters in an assembly language routine name, number of characters between (but not including) the quotes in a string constant). |
| none | 1 | End of program token. |

When programs are entered, BASIC converts certain words and special symbols (both are called keywords) to 1-byte tokens to conserve memory. Some BASIC keywords are: PRINT, IF, and THEN, functional references such as PI, SIN(, and SEG$( (the left parenthesis following a function name is considered to be part of the name), and special characters such as

```
+
-
(
)
'
"
\
,
:
```

The + or - preceding a numeric constant is considered to be a separate entity from the number.

Every character in a REM statement or string constant requires one byte. To reduce program size minimize the number of characters in each.

Each use of the multiple statement line saves six bytes. The program:

```
10  A = 3
20  B = 4
```

takes six bytes more memory than the equivalent program

```
10  A = 3\B = 4
```

When the BASIC program is running, the following additional array and string storage is required. For each numeric array, the number of bytes allocated is

```
          4*(SS1MAX+2)
```

for a singly-dimensioned array,

or

```
          4*[(SS1MAX+1) * (SS2MAX+1) +1]
```

for a doubly-dimensioned array, where SS1MAX and SS2MAX are the maximum values of the first and second array subscripts, respectively. For each string array, the number of bytes allocated is

```
          2*(SS1MAX+2)
```

for a singly-dimensioned array or

```
          2*[(SS1MAX+1) * (SS2MAX+1)+1]
```

for a doubly-dimensioned array, where SS1MAX and SS2MAX are the maximum values of the first and second array subscripts, respectively.

For each non-null string scalar or array element of length N currently defined in the BASIC program, N+4 bytes of string storage are required. Null strings are not stored. The symbol table or array entry has a special indicator to specify a null string.

In addition to these parameters, when an OPEN statement is executed a buffer is allocated from the user area if there is currently room. If there is no room in the user area, the buffer is allocated from a system I/O area common to all users (if room is available there). Table D-2 contains the standard buffer sizes. A nonstandard buffer size may be specified on nonfile-structured devices by means of the RECORDSIZE option in the OPEN statement. Specifying DOUBLE BUF causes allocation of two equal size buffers.

Table D-2
Standard File Buffer Sizes

| Device | Buffer Size (words) |
|---|---|
| Disk | 256 |
| DECtape | 256 |
| Magtape | 256 |
| Cassette | 64 |
| High-speed paper tape punch | 12 |
| High-speed paper tape reader | 12 |
| Line printer | 16 |
| Card reader | 8 |

INDEX

Please cut along this line.

MU BASIC/RT-11 User's Manual
DEC-11-LIBRA-A-D
March, 1975

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form

Did you find errors in this manual? If so, specify by page.

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

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

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
- ☐ Non-programmer interested in computer concepts and capabilities

Name ______________________ Date ______________

Organization ______________________

Street ______________________

City ______________ State ______________ Zip Code or Country ______________

If you do not require a written reply, please check here. ☐

------------------------------------------------------------ **Fold Here** ------------------------------------------------------------

---------------------------------------------- **Do Not Tear - Fold Here and Staple** ----------------------------------------------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

```
Software Communications
P. O. Box F
Maynard, Massachusetts  01754
```

digital

DIGITAL EQUIPMENT CORPORATION
MAYNARD, MASSACHUSETTS 01754