DataGeneral

Software Documentation

Extended BASIC Reference

093-000065-10

NOTICE

Extended BASIC
Reference
093-000065-10

Revision History:

Original release: November 1971
First revision: May 1972
Second revision: September 1972
Third revision: March 1973
Fourth revision: September 1973
Fifth revision: October 1974
Sixth revision: February 1975
Seventh revision: April 1977
Eighth revision: August 1978
Ninth revision: August 1979
Tenth revision: August 1983

Effective with:

Extended BASIC 1.60 (AOS and AOS/VS)
Extended BASIC 5.60 (RDOS, DOS, and DG/RDOS)

Extended BASIC Reference

093-000065-10

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.

Ordering No. 093-000065

Printed in the United States of America
Revision 10, August 1983

Summary of Changes

The *Extended BASIC Reference* is a revision of the *Extended BASIC User's Manual*. The manual has been reorganized to promote ease of reference. Former chapters 4 (Extended BASIC Functions), 5 (Array Manipulation), and 6 (File Input and Output) of the *User's Manual* have been assimilated into a dictionary of commands, statements, and functions (new chapter 3). The introductory material from these chapters appears now in chapter 1. Former chapter 7 (SOS Extended BASIC) has been eliminated. The appendices, however, remain the same.

In addition, the RDOS and DOS "privileged" commands previously documented in the *Extended BASIC System Manager's Guide* are described in chapter 3 of this edition. These commands include ALL, CDIR, CPART, DISABLE, ENABLE, FALL, FMSG, FREE, INIT, KILL, MAX, and RELEASE.

Several previously undocumented commands also have been added to chapter 3: ESC, NOESC, ECHO, NOECHO, and NOMSG.

Also new to this revision is a section in chapter 1 on screen handling.

Appendix E summarizes similiarities and differences of Extended BASIC on the different operating systems.

The new material and the technical revisions and corrections to previously documented material are marked with revision bars (|) in the margins.

Contents

Chapter 2 - Arithmetic and Strings

Chapter 3 - BASIC Statements, Commands, and Functions

Preface

Data General's Extended BASIC is a powerful, straightforward language. Its interactive qualities enable beginning programmers to develop their skills; its advanced features allow experienced programmers to handle complex and diverse applications. These attributes have made BASIC very popular, and in recent years it has become a model language for multiuser systems.

This manual presumes that you have experience with the BASIC language. If you do not, the introductory manual, *basic BASIC,* gives you the background you need.

This manual begins with a brief description of Extended BASIC, explains terminology and symbols, and outlines the steps to follow to log on to the BASIC system, create and run a program, and log off. Also included in chapter 1 are brief introductions to matrices, file I/O, and screen handling. Chapter 2 explains BASIC arithmetic and strings; it includes numbers, variables, arrays, and arithmetic operations. In chapter 3, all interactive BASIC commands, statements, and functions are listed alphabetically; each is explained in detail, and illustrated by example. Chapter 4 explains how to call assembly language subroutines from Extended BASIC.

Appendix A lists all BASIC error messages; Appendix B outlines BASIC programming on mark-sense cards. Other appendixes include Hollerith and ASCII character sets and a BASIC keyword summary.

See the following manuals for related information:

069-000003 *basic BASIC*

093-000119 *Loading and Managing Extended BASIC*

069-000002 *Introduction to the Real Time Disk Operating System*

093-000075 *Real Time Disk Operating System Reference Manual*

093-000093 *Introduction to the Real-Time Operating System (RTOS)*

093-000056 *Real-Time Operating System Reference Manual*

093-000201 *Disk Operating System Reference Manual (DOS)*

093-000087 *BATCH User's Manual*

069-000016 *Introduction to the Advanced Operating System (AOS)*

093-000122 *AOS and AOS/VS Command Line Interpreter User's Manual*

069-000018 *Learning to Use Your Advanced Operating System*

Keyword Descriptions

This manual uses the following format to describe each BASIC keyword:

- Grids with check marks to indicate the operating systems on which the term may be used and whether it is a statement (S), command (C), or function (F). A keyword may be available on one, two, or all four systems.

AOS, AOS/VS	
RDOS, DOS	

S	
C	
F	

- Keyword, followed by a brief description of its purpose. Certain keywords (e.g., MSG) are available in more than one version of Extended BASIC, but differ significantly in description. Each version is documented separately.
- Format for using the word. Insert parentheses as shown. Brackets indicate optional arguments. Other conventions are described below.
- Arguments, with definitions.
- Remarks, which include rules, cautions, and other pertinent information for using the keyword.
- Examples, showing typical uses and illustrating the format.

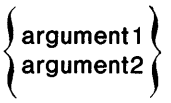
Note: Reserved filenames under DOS and RDOS Extended BASIC use the $ symbol as a prefix, whereas under AOS and AOS/VS the prefix is an @ symbol. Most of the examples in this manual reflect an RDOS system;

AOS and AOS/VS users, please note the difference. See chapter 1 for detailed information on reserved filenames.

Format Conventions

The following general format illustrates the typographical conventions used in the keyword formats:

KEYWORD argument *[option]...*

Where	Means
KEYWORD	Enter the word (or its accepted abbreviation) as shown.
argument	You must enter the given argument (such as a filename). In the following case enter one of the arguments, but not the braces, which merely set off the choices: 
[option]	You have the option of entering the given argument. Don't enter the brackets, which merely set off the optional argument.
...	You may repeat the preceding entry or entries. The explanation tells you what you may repeat.

Also Note

All numbers are decimal unless shown otherwise; e.g., 35_8.

In examples of system interactions, the manual uses

THIS TYPEFACE TO SHOW YOUR ENTRY
ITALIC TYPEFACE FOR SYSTEM QUERIES AND RESPONSES

) is the AOS and AOS/VS CLI prompt.

R is the RDOS/DOS CLI prompt.

* is the Extended BASIC prompt.

□ indicates a space in an entry or in a system response. It is used only as necessary for clarity.

End of Preface

Chapter 1
Introduction

Extended BASIC is an interactive programming language that operates under Data General's Advanced Operating System (AOS), Advanced Operating System/Virtual Storage (AOS/VS), Real-time Disk Operating System (RDOS), Real-Time Operating System (RTOS) as a feature of RDOS, and Disk Operating System (DOS).

RDOS supports certain commands and features that DOS does not. Generally, if you try to use an RDOS-only command or feature in a DOS system, it is nonoperational, and you receive an error message.

DOS does not support cassette use or BATCH mode. Disks created under DOS are entirely compatible with RDOS. Diskettes created under RDOS may not be compatible with DOS when certain RDOS features not supported under DOS are used. See the *Disk Operating System Reference Manual* for more information.

General Information

Data General's Extended BASIC includes the following features:

- String manipulation
- Matrix operations
- Program and keyboard modes
- Fixed- and variable-length file manipulation
- Format control
- Assembly language subroutines

Terminology

BASIC uses words, sometimes called *keywords,* as instructions. When written in appropriate formats, they act as *statements* or *functions* in a program. Many keywords can also be used as keyboard *commands.*

Some BASIC words alone perform an operation. Others require one or more *arguments* to be properly executed, for example:

```
INPUT A,B
```

In this case, A and B are arguments to the INPUT instruction.

The following abbreviations appear in the descriptions of arguments used with BASIC keywords:

Abbreviation	Meaning
var	numeric variable
expr	numeric expression
rel-expr	relational expression
str lit	string literal
val	numeric value
line no.	line number
col	column
control var	control variable
svar	string variable
mvar	matrix variable
filename	a disk filename or a device

The BASIC Program

A BASIC program consists of several BASIC *statements.* Each statement includes a properly formatted BASIC word preceded by a *line number* in the range 1 to 9999. The line number of a statement determines the order in which BASIC executes it. Program execution proceeds from the lowest numbered line to the next higher numbered line, unless directed elsewhere by statements such as GOTO or GOSUB.

You can use the exclamation point (!) in place of a REM statement, or at the end of a line to add comments about the program step. Anything to the right of the ! has no effect on the program. See AUDIT in chapter 3 for an example of this in a program.

As in the following example, write each program statement on a separate line; terminate each line with a carriage return (CR) or new line (NEW LINE).

```
* 05 PRINT "SAMPLE PROGRAM"
* 10 LET A=5
* 15 LET B=2
* 20 PRINT A*B
* 25 END
```

The asterisk (*) prompt at the beginning of each line indicates that BASIC is ready for you to enter a command or a program statement. In RDOS/DOS/RTOS environments, a system manager can create any prompt string of up to 10 characters, or use the default asterisk prompt.

The asterisk prompt appears in the examples in this manual.

BASIC Commands

BASIC commands do not include line numbers; the system executes them immediately after you terminate the command with a carriage return or new line. In the following case the command LIST instructs BASIC to display the current program statements. The command RUN instructs BASIC to execute the program.

```
* LIST
0005 PRINT "SAMPLE PROGRAM"
0010 PRINT 5*2
   .
   .
   .
0025 END
* RUN
SAMPLE PROGRAM
10

END AT 0025
*
```

See "Uses of Commands" below for additional uses of commands.

Blank Spaces

As an aid to typing, BASIC ignores blank spaces unless they are part of a string literal; however, this requires some care. For example, in the following statement S and P are intended to be variables for initializing a loop:

```
* 10 FOR X=S TO P
```

However, BASIC ignores the spacing and interprets the statement as

```
10 FOR X=STOP
```

and displays a syntax error message.

Using Extended BASIC

The following sections describe the procedures for invoking Extended BASIC on the various operating systems; for creating, executing, and correcting programs; for interrupting execution; and for leaving Extended BASIC.

Logging On to RDOS or DOS Extended BASIC

For single-user systems, logon procedures do not apply. Execute Extended BASIC from the RDOS CLI and come immediately to the BASIC prompt (*). The directory BASIC.DR must be initialized before execution, or it must be a subdirectory in the partition in which you are executing BASIC.

On multiuser systems, you can log onto the system as soon as you see the BASIC prompt message on your terminal:

BASIC xx.xx
Ready

Begin the log-on procedure by pressing either the ESC or DEL key. Use the ESC key on hard-copy terminals, on which DEL echoes a backarrow (←). Use the DEL key on display terminals, on which DEL backspaces the cursor and erases a character.

The system requests your account name. Respond by typing your assigned account name (4-20 characters), and a carriage return. BASIC checks all the characters before allowing you access, but saves only the first four. It denies you access if that 4-character name belongs to an active user. The system asks for your assigned password. Type the password and a carriage return. To protect the confidentiality of your password, it is not echoed at your terminal.

You have three chances to correctly enter your account name and password. If your third attempt fails (no match is found in the BASIC.ID file), the message TOO MANY ATTEMPTS appears, the ready message reappears, and modem lines disconnect. You must reinitiate the sign-on sequence. If you encounter sign-on errors other than UNKNOWN ACCOUNT NAME or UNKNOWN PASSWORD, the ready message reappears, modem lines disconnect, and you must reinitiate the sign-on sequence.

If your account name and password are valid, the system outputs your account name, the date, time, and assigned terminal number and displays a prompt. A summary of the log-on procedure and format follows:

BASIC xx.xx *Ready*	(Screen display)
ESC	(Press ESC or DEL.)
ACCOUNT NAME: aaaa	(Enter a 4-20 character account name.)
PASSWORD: yyyy	(Enter a 0-20 character password--not echoed.)

aaaa	*mm/dd/yy*	*hh:hh*	*SIGN ON,*	*zz*
(Account name)	(Date)	(Time)		(Terminal no.)

* (Prompt)

Logging On to AOS, AOS/VS Extended BASIC

Log onto an AOS or AOS/VS system according to the instructions in the *Command Line Interpreter User's Manual* for the given system.

Once you have logged on to the operating system, you may go directly into BASIC or to the command line interpreter (CLI), depending on how the system manager has built your system. If you go directly into BASIC, the message below appears; and you can begin to work. If you log into the CLI, enter the following command command to execute BASIC; BASIC displays logon information and its prompt:

```
) EXECUTE BASIC

AOS BASIC Revision xx.xx 14-Sep-83 1-Oct-83 09:14:22

*
```

Creating a New Program

Having successfully logged onto the system, you may enter a new program, or modify and run an old program. It is generally good practice to type the NEW command before entering a new program. The NEW command clears your work area in memory and thereby prevents the interspersion of lines from an old program into your new program. NEW is described in chapter 3.

When typing a new program, begin each line with a line number of not more than four digits; end each line with a carriage return. Before pressing the CR key, you can correct typing errors with the DEL key. Press DEL once for each character you want erased; then continue by typing the correct characters.

On a CRT terminal, each DEL erases the last character on the screen. Note that on some terminals the RUBOUT key replaces DEL. If you are using a hard-copy terminal, a backarrow (←) echoes at your terminal each time you press RUBOUT. For example:

```
* 10 PRINT "CONV ← ← RRECTION BY RUBOUTS"
```

In line 10, two characters (N and V) are "rubbed out" and then the line is completed. Use a LIST command to inspect the corrected line.

```
* LIST 10
0010 PRINT "CORRECTION BY RUBOUTS"
*
```

In addition, in RDOS/DOS you can delete an entire current statement or command line by entering a backslash character (\). BASIC echoes a backslash and a carriage return. On AOS and AOS/VS, the ESC key has the same effect.

```
* 10 PRINT "HELLO\ (Backslash deletes line.)
* 10 PRINT "WELCOME, MEREDITH" (Reenter line.)
* LIST
0010 WELCOME, MEREDITH
```

Programs can be documented by using the REM statement or by an exclamation mark (!) following a statement. Enter comments after REM or the exclamation mark, for example:

```
0010 REM THIS PROGRAM PERFORMS ORDER ENTRY
0020 DIM A$(25) ! DIMENSION STRING VARIABLE
0030 REM
0040 DIM B(100) ! DIMENSION NUMERIC ARRAY
...
```

The REM on line 0030 merely aids readability.

Running a Program

After you have finished typing a program, execute it by entering the RUN command. BASIC runs the program, starting from the lowest numbered statement. If there are no runtime errors (see Appendix A), BASIC outputs all results that you requested in PRINT statements.

The system translates lowercase letters in BASIC to uppercase if they are part of a variable name, keyword, or numeric literal, but not if they are part of a string literal. Commands that accept string literals as arguments (e.g., CHAR, ACL, CHATR) accept either uppercase or lowercase letters. The edit commands, .E and .C, do not translate lowercase letters to uppercase, so you must count lowercase and uppercase letters separately when you edit.

Programs that were previously written and saved can be executed in one of the following two ways:

```
* LOAD "program-name"
* RUN
```

or

```
* RUN "program-name"
```

Saving a Program

There are two methods for saving your BASIC programs: save a memory (or core) image of the program using the SAVE command, or save an ASCII copy using the LIST command.

To save and recover a memory image of your program, use the following commands:

```
* SAVE "program-name"
* LOAD "program-name"
* RUN
```

To save and recover an ASCII list file, use the following commands:

```
* LIST "program-name"
* ENTER "program-name"
* RUN
```

The LIST command is useful for maintaining an updated version of your code in ASCII format. If you are entering source code under control of the BASIC interpreter, use the LIST command to copy the code to a specified file. If you are copying to the same file each time you revise your code, you are prompted to press the NEW LINE key to rewrite the file, for example:

```
* (Write a program.)
* LIST "TEST.PGM"
* ENTER "TEST.PGM"
* (Revise the program.)
* LIST "TEST.PGM"
PRESS NEW LINE KEY TO DELETE OLD: <NL>
*
```

Correcting the Program

After running a program, you may need to change it because of error messages or incorrect results. Correct the program by using any of the following procedures:

1. Substitute a new statement for a statement containing errors by retyping the entire line, including line number and carriage return.

2. Eliminate a statement from the program by typing its line number followed by a carriage return, or by using the ERASE command:

   ```
   * 125 <CR> (Deletes line 125.)
   * ERASE 200,300 (Deletes all lines from 200 to
                    300.)
   ```

3. Insert new statements between existing statements by typing the new statements with intermediate line numbers. If the number of new statements exceeds the number of line numbers available between the existing statements, use the RENUMBER command (see chapter 3) to change the increment between line numbers. (It is generally good practice to number your lines by increments of 10 to allow for program expansion and correction.)

4. Use editing commands to correct statements held in the edit buffer. If Extended BASIC encounters faulty syntax in a statement, it inserts the statement into the edit buffer. Several editing commands can then be used to correct the statement, for example:

   ```
   * 0010 LET B$ = A
   ERROR 2 - Illegal statement syntax
   * .C/A/A$
   0010 LET B$ = A$
   *
   ```

   In this example the .C command changes A to A$, and sends the corrected statement to working storage. In many instances, editing commands can prevent you from having to retype an entire line of code.

Interrupting Program Execution

To stop a running program, the listing of a program, or any other task which BASIC is performing, press the ESC key. BASIC then outputs a prompt to signal that you can enter a new command.

```
* RUN
  .
  .
  .
<ESC>
STOP AT 0110
*
```

The line number that is output is the last line that completed execution.

Certain BASIC statements or commands--INPUT, LREAD, ENTER, ERASE, MAT PRINT, DELAY, MAT INPUT, LIST, FILE, and LIBRARY--may require a long time to execute. You can abort them, but further execution that depends on their completion will be affected.

The statement MAT INV and all file I/O statements may require considerable execution time, but cannot be interrupted by ESC. If you enable ESC on the console and get no response, check to see if one of these noninterruptable statements is executing or if there is a system failure.

Logging Off RDOS or DOS Extended BASIC

On RDOS or DOS you may leave Extended BASIC by entering the BYE command. The BASIC system then outputs a summary of usage information (see below) and puts the terminal into an idle state:

```
* BYE
aaaa mm/dd/yy hh:mm SIGN OFF, zz
aaaa mm/dd/yy hh:mm CPU USED, qq
aaaa mm/dd/yy hh:mm I/O USED, rr, ss

BASIC xx.xx
R
```

where: represents:

aaaa	Your account name
mm/dd/yy	Today's date
hh:mm	The current time
zz	The terminal port number
qq	The number of CPU seconds you used during the terminal session, to the nearest tenth of a second
rr	The number of file input and output statements executed (OPEN, CLOSE, READ, WRITE, etc.)
ss	The number of BASIC I/O statements executed (LIST, LOAD, ENTER, etc.)

Logging Off AOS and AOS/VS Extended BASIC

When you want to leave AOS or AOS/VS Extended BASIC, enter the BYE command. The BASIC process then terminates and the process (e.g., CLI) that was in effect when BASIC was invoked continues.

Telephone Line Interruption (RDOS, DOS, RTOS)

If your terminal is attached to the computer by a Bell 103 modem or compatible hardware, and line transmission fails for any reason, BASIC saves your current program and data in a memory image file and does an implicit BYE command. You can retrieve the file--named aaaa$.CI, where aaaa is your account name--by using the LOAD command (see Figure 1-1).

BASIC Program Example

The following example shows an entire BASIC session under RDOS or DOS: logging in, communicating with the system operator, running the program, and logging off.

```
BASIC xx.xx
Ready
<ESC>                    (Press ESC or DEL key.)
ACCOUNT-NAME: DREW
PASSWORD: EASY           (Password EASY not echoed.)

DREW 1/23/83 10:32 SIGN ON, 2

* MSG OPER PLS MOUNT TAPE #1255 (NO RING)
FROM OPER: DONE-TAPE ON MT12
* MSG OPER THANX
* LOAD "PRODUCTION"

* LIST
0010 DIM A$ (10)
0020 INPUT "TAPE MOUNTED ON",A$
0030 A$=A$,":0","<0>"
0040 OPEN FILE (0,3),A$
0050 READ FILE (0),A,B,C$
0060 IF EOF (0)=1 GOTO 200
0070 PRINT A,B,C$
0100 GOTO 50
0200 CLOSE FILE (0)
0210 PRINT "END OF JOB"
0220 STOP

* RUN
TAPE MOUNTED ON MT12
   .
   .
   .
END OF JOB

STOP AT 0220
* MSG OPER PLS RELEASE MT12
FROM OPER: TAPE REMOVED FROM MT12
* BYE
DREW 1/23/83 10:40 SIGN OFF, 2
DREW 1/23/83 10:40 CPU USED, .3
DREW 1/23/83 10:40 I/O USED, 4,2

BASIC xx.xx
Ready
```

Notice that this program provides device independence. That is, the assignment of the magnetic tape drive number is deferred until program execution time, thereby allowing the system operator to assign any available unit.

Uses of Commands

You can use most BASIC statements as keyboard commands. However, certain statements have meaning only within the context of a program and cannot be used as commands. These statements are DATA, DEF, END, FOR, GOSUB, GOTO, NEXT, ON, REM, RETURN, RETRY, and STOP. All other BASIC statements can act as commands to help you

- Perform file I/O
- Perform calculations
- Dynamically debug programs

File I/O

With BASIC commands you can open and close files; you can also input or output programs and data from files and devices. These commands are derived from the file I/O statements described in chapter 3. For example:

```
* OPEN FILE (1,3), "$PTR"
* READ FILE (1), A, B, C, D, E, F, G (5)
```

```
ACCOUNT NAME: MARY                    (User signs on)
PASSWORD:

MARY 10/13/83 06:35 SIGN ON, 12

* 20  FOR I = 0 TO 19                 (User enters and runs program)
* 30  ;I, SYS(I)
* 40  NEXT I
* LIST
0020  FOR I = 0 TO 19
0030     PRINT I, SYS(I)
0040  NEXT I

* RUN
0          23746
1          26
2          10
3          1977
4          2
5          .2
6          6
7          0
8                                     (Disconnect occurs here)

ACCOUNT NAME: MARY                    (User repeats sign-on)
PASSWORD:

MARY 10/13/83 06:47 SIGN ON, 03

* WHATS "MARY$.CI"                    (Identifies disconnected program)

MARY$.CI DW 379 10/13/83 06:38 (10/13/83) 00

* LOAD "MARY$.CI"                     (Retrieves it)

* LIST)                               (Examines it)
0020  FOR I = 0 TO 19
0030     PRINT I, SYS(I)
0040  NEXT I
* ...                                 (Session continues)
```

DG-25447

Figure 1-1. Telephone Line Interruption

Calculations

With the PRINT command, you can obtain immediate results of arithmetic computations. A semicolon (;) can be used for the word PRINT.

```
* ;EXP(SIN(3.4/8))
1.5103188
* LET A=EXP(SIN(3.4/8))
* PRINT USING "+####.####□□□□",A
+□□□1.5103
*
```

You also can interrupt a running BASIC program and use the assigned values of program variables to make calculations.

```
* LIST
0010 FOR J = 1 TO 1000000
0020 LET X = X + 1
0030 NEXT J
* RUN
  .
  .
  .
<ESC>
STOP AT 0010
* PRINT J, 10 * X
100      990
*
```

Dynamic Program Debugging

You can interrupt a running program (using ESC or programmed STOP statements) at a number of different program points. You can check the current values of the variables at those points and make corrections to statements or variables in the program, as necessary.

Then use either "RUN line number" or CON to restart the program at the point of interruption without losing the values of the variables or the newly inserted values and statements. Note the following four examples.

1.
```
* RUN
  .
  .
  .
<ESC>
STOP AT 1100
* IF A<>B THEN PRINT B,A
```
(Command conditionally provides for examination of A and B.)
```
.025 .5
*
```

2. The program performs a series of calculations and prints the results.

```
* RUN
  .
  .
  .
2.33333
5.41234
8.99999
<ESC>
STOP AT 0570
* READ X1, X2, X3
```
(Space over the next 3 values in the data block. Resume program execution at the next statement.)
```
* CON
3.16524
  .
  .
  .
```

3.
```
* RUN
  .
  .
  .
<ESC>
STOP AT 1100
* ;A
```
(Print value of variable A.)
```
0
* A = -1
* C$ = "% OF LOSS"
```
(Change the value of arithmetic variable A and string variable C$.)
```
* RUN 505
```
(Resume running at statement 505.)
```
  .
  .
  .
```

4.
```
* DIM A (14,4)
* RUN
  .
  .
  .
<ESC>
STOP AT 0500
* DIM A (3,5)
```
(Redimension array A.)

Dimensioning Arrays

One-dimensional arrays are called *vectors;* two-dimensional arrays are called *matrixes.* Matrix statements also work for vectors wherever the row argument is optional.

You can dimension arrays with any of the following three methods:

- Use a DIM statement to declare the number of elements for a vector or rows and columns for a matrix.
- Include the dimensions in a matrix statement.
- Allow a default size of 10 elements, or 10 rows and 10 columns, by not specifying dimensions in a DIM or matrix statement.

Note: A matrix does not have row 0 or column 0; and as in all BASIC arrays, BASIC stores matrix elements by row in ascending locations in memory.

A matrix that is dimensioned in a DIM statement is automatically initialized to all zeros.

Matrix statements allow dimensioning and redimensioning as long as the total number of elements in the new dimensions does not exceed the total number of elements of the matrix declared in the original DIM or other matrix statement.

In the following example, matrix statements 40 and 60 legally redimension matrix A as well as perform matrix operations (see chapter 3). In each case the total number of elements does not exceed the number declared in statement 20.

```
* 20 DIM A(15,14)          (210 elements in matrix A)
* 40 MAT A=CON(20,7)       (140 elements)
* 60 MAT A=ZER(10,10)      (100 elements)
```

File Concepts

The following sections briefly define files, file devices, and records; they also present several conventions that govern record I/O. See chapter 3 for detailed explanations of the file file I/O statements.

Definition of a File

A file is a collection of related data treated as a unit. Each file has one or more names (called *filenames*) which enable both the user and the system to address it. Related to the manipulation of files are devices.

File Devices

Devices are the physical means for storing and retrieving information. There are two distinct types of devices:

- Unit record devices. These include card readers, terminals, and line printers, which usually transmit and/or receive only single records. They are used for I/O, data storage, and retrieval.
- Multifile devices. These include magnetic tape units and disks, which enable you to read and write more than one file per device. In particular, the flexibility of disks enables you to organize a file so you can randomly read or write individual records.

Creating File Records

A *record* is a discrete unit of a file; it is what is transmitted in file I/O.

A single record is the result of a READ FILE or WRITE FILE statement. You may specify a record size, in bytes, with an OPEN FILE statement (see the description in chapter 3). The output of a WRITE FILE statement with fewer than the specified number of bytes is padded with nulls to fit the size of the record. Output having greater than the specified number of bytes causes an error. If a READ FILE statement transfers fewer bytes than the record size, the remaining bytes in the record, including any nulls, are passed over.

If you specify no record size, WRITE FILE does not pad with nulls, and READ FILE does not pass over any bytes.

Filenames

The following sections present the rules governing the composition and use of filenames according to the operating system under which Extended BASIC is running.

DOS and RDOS Disk Filenames

Each disk file created under BASIC in a DOS or RDOS operating system may have a filename made up of 1-10 characters. Legal characters include

- A-Z
- a-z (converted to uppercase by the operating system)
- 0-9
- $ (dollar sign)

In addition, you may append an optional one- or two-character alphanumeric extension to the filename by separating it from the filename with a period in the form, filename.ex.

Unlike RDOS utility programs such as MAC and RLDR, BASIC does not recognize any special two-character alphanumeric extensions. You can create extensions to suit your needs, for example:

TEST.SR can be a SouRce file.

TEST.CI can be the Core Image file obtained by saving TEST.SR.

TEST.LS can be a LiSting file output from the program.

DOS and RDOS Reserved Filenames

Unit record devices and magnetic tape devices have special names without extensions. Devices with reserved names are listed as follows:

$TTI and $TTI1	Input consoles
$TTO and $TTO1	Output consoles
$CDR and $CDR1	Punched card readers
CTn	Cassette units ($0 < n < 17_8$)
$LPT and $LPT1	Line printers
MTn	Magnetic tape units ($0 < n < 17_8$)

$PLT and $PLT1	Incremental plotters (access via assembly language subroutines; see Appendix B)
$PTP and $PTP1	Paper tape punches
$PTR and $PTR1	Paper tape readers
QTY:n	Multiplexor consoles

For a complete list of RDOS reserved filenames, see the *RDOS CLI User's Manual.*

AOS and AOS/VS Disk Filenames

Each disk file created under BASIC in an AOS or AOS/VS operating environment has a filename of 1-31 characters. Legal characters include

- A-Z
- a-z (converted to uppercase by the operating system)
- 0-9
- . (period)
- $ (dollar sign)
- _ (underscore)

You can select any combination of legal characters to create a filename. A few examples of legal filenames are as follows:

```
FILE_NAME.NEW
SAVE.FILE$.SR
LIFE.JAN.5
```

Unlike operating system utility programs such as MASM (the macroassembler), BASIC does not recognize any special alphanumeric extensions. You can create filename extensions to suit your needs, for example:

TEST.SR can be a SouRce file.

TEST.CI can be the Core Image file obtained by saving TEST.SR.

TEST.LS can be a LiSting file output from the program.

AOS and AOS/VS Reserved Filenames

Unit record devices and magnetic tape devices have special names without extensions. Devices with reserved names are listed as follows:

@CDR,@CDR1, @CDR2,...@CDRn	First and succeeding card readers
@CON0,@CON1, @CON2,...@CONn	First and succeeding console display/keyboards or asynchronous communications lines
@DPn-DPn17	Moving-head disk units 0 through 7 on the first controller, and 10 octal through 17 octal on the second controller. n is a single alphabetic character indicating the disk type. These types are described in *Managing AOS* and *Managing AOS/VS.*
@LPT,@LPT1, @LPT2,...@LPTn	First and succeeding line printers
@MTA0-@MTA17	Magnetic tape units 0 through 7 on the first controller, and 10 octal through 17 octal on the second controller.
@MCA, @MCA1	First and second multiprocessor communications adaptor controllers
@PLT,@PLT1, @PLT2,...@PLTn	First and succeeding digital plotters
@PTP,@PTP1, @PTP2,...@PTPn	First and succeeding paper tape punches
@PTR,@PTR1, @PTR2,...@PTRn	First and succeeding paper tape readers

Screen Handling

Extended BASIC offers several methods for explicitly controlling the position of the cursor. The TAB(X) function, used in conjunction with the PRINT statement, enables you to position the cursor to the column designated by X on the current line. The LWRITE statement provides both column and line cursor positioning by allowing the embedding of control characters in strings. Used together with the PRINT statement, LWRITE with the appropriate arguments enables full control of the screen.

For DG-type terminals, the syntax of the LWRITE statement for controlling the cursor is

```
LWRITE "<16><column><row>"
```

where column is a numeric expression or variable in the range 0-79, and row is a numeric expression or variable in the range 0-23. The argument <16> is a decimal value indicating that the next two decimal values are to be interpreted as row and column positions. (Non-DG terminals may require other values than <16>.) For example, the statement

```
LWRITE "<16><0><0>"
```

positions the cursor to the leftmost column of the top row of the screen.

The entire string argument to the LWRITE statement can itself be a string variable. For instance, the statement

```
LWRITE "<16><39><11>"
```

positions the cursor to the approximate center of the screen. The equivalent of this statement, using a string variable as the argument, is:

```
0010 DIM A$(3)
0020 LET A$(1,1)=CHR$(16)
0030 LET A$(2,2)=CHR$(39)
0040 LET A$(3,3)=CHR$(11)
0050 LWRITE A$
```

If you add the statement

```
0060 PRINT "HELLO"
```

to this example, the string HELLO will be printed at the approximate center of the screen.

The following five examples further illustrate cursor positioning.

1. This program prompts you for a string, prints it at the approximate center of the screen, then prompts you for another string. Exit from the program by pressing ESC.

```
* LIST
0005 PRINT "<12>"          ! CLEAR SCREEN
0006 PRINT "<7>"           ! SOUND THE
                           TERMINAL TONE
0010 INPUT "→ ",X$
0030 LWRITE "<16><36><11>"
                    ! POSITION THE CURSOR

0040 PRINT X$
0045 DELAY =5
0050 GOTO 0005
```

2. This program clears the sceen, prompts you for a number, and prints that number at column 0, row 10. The program continues to print the number you enter at the same column and row position until you press ESC.

```
* LIST
0001 LET C$=CHR$(0)
0002 LET R$=CHR$(10)
0003 LET A$="<16>",C$,R$
0005 PRINT "<12>"
0010 INPUT "→ ",X
0020 LWRITE A$
0030 PRINT X
0040 GOTO 0010
```

3. This program allows you to specify dynamically the row and column position. Note how the string argument to the LWRITE command is built. If you enter 0,0 to the prompt, the string HELLO is printed at the top row, leftmost column. Exit from this program by pressing ESC.

```
* LIST
0010 DIM A$(3)
0015 LET A$(1,1)=CHR$(16)
0016 PRINT "<12>"
0020 INPUT "ROW AND COLUMN? ",R,C
0030 GOSUB 9000
0050 PRINT "HELLO"
0060 DELAY=3
0070 GOTO 0016
9000 REM POSITION CURSOR SUBROUTINE
9001 REM R = ROW, C = COLUMN
9005 LET A$(2,2)=CHR$(C)
9010 LET A$(3,3)=CHR$(R)
9020 LWRITE A$
9030 RETURN
```

4. This program is a more practical application of cursor positioning. Here a data entry screen labeled NAME AND ADDRESS FILE is created. As each item of information is entered and NEW LINE is pressed, the cursor jumps to the next input field. At the end the data is displayed. The program can be altered to loop at the end of each data entry cycle and to output the data to a file; in this way the programmer can code useful routines for capturing and displaying data at specific screen locations.

```
* LIST
0001 DIM N$(25)
0002 DIM L$(15)
0003 DIM S$(15)
0004 DIM C$(15)
0005 DIM A$(50)
0010 PRINT "<12>"
0020 LWRITE "<16><28><1>"
0030 PRINT "<20>NAME AND ADDRESS
     FILE<21>"  ! UNDERSCORED
0040 LWRITE "<16><0><6>"
0050 PRINT "LAST NAME:"
0060 LWRITE "<16><25><6>"
0070 PRINT "FIRST NAME:"
0080 LWRITE "<16><55><6>"
0090 PRINT "MIDDLE INITIAL:"
0100 LWRITE "<16><0><10>"
0110 PRINT "STREET:"
0120 LWRITE "<16><25><10>"
0130 PRINT "CITY:"
0140 LWRITE "<16><45><10>"
0150 PRINT "STATE CODE:"
0160 LWRITE "<16><65><10>"
0170 PRINT "ZIP:"
```

```
0180 REM DATA ENTRY
0190 LWRITE "<16><11><6>"
0200 LREAD "",L$,B$
0210 LWRITE "<16><37><6>"
0220 LREAD "",F$,B$
0230 LWRITE "<16><71><6>"
0240 LREAD "",M$,B$
0250 LWRITE "<16><8><10>"
0260 LREAD "",S$,B$
0270 LWRITE "<16><31><10>"
0280 LREAD "",C$,B$
0290 LWRITE "<16><57><10>"
0300 LREAD "",S1$,B$
0310 LWRITE "<16><70><10>"
0320 LREAD "",Z$,B$
0330 PRINT
0340 PRINT
0350 REM OUTPUT
0360 LET N$=F$,"□",M$,"□",L$
0370 PRINT N$
0390 PRINT S$
0400 LET A$=C$,",□",S1$,"□□",Z$
0410 PRINT A$
0420 STOP
```

5. This program demonstrates the versatility of Extended BASIC's cursor positioning capability. Try it.

```
* LIST
0005 PRINT "<12>"
0010 LET A=0
0020 LET B=79
0030 LET C=0
0040 LET D=21
0050 LET Y=0
0060 FOR I=1 TO 11
0070     FOR X=A TO B
0080         GOSUB 1000
0090     NEXT X
0100     LET X=X-1
0110     FOR Y=C TO D
0120         GOSUB 1000
0130     NEXT Y
0140     LET Y=Y-1
0150     FOR X=B TO A STEP -1
0160         GOSUB 1000
0170     NEXT X
0180     LET X=X+1
0190     FOR Y=D TO C STEP -1
0200         GOSUB 1000
0210     NEXT Y
0220     LET Y=Y+1
0230     LET A=A+1
0240     LET B=B-1
0250     LET C=C+1
0260     LET D=D-1
0270 NEXT I
0280 LWRITE "<16><37><10>"
0282 PRINT "<14>The End<15>"
         ! ENABLE BLINKING
0300 END
1000 REM
1010 LET C$=CHR$(X)
1020 LET R$=CHR$(Y)
1030 LET A$="<16>",C$,R$
1040 LWRITE A$
1050 PRINT "*"
1060 RETURN
```

End of Chapter

Chapter 2
Arithmetic and Strings

Numbers

An Extended BASIC number may range from + or - 5.4 * 10^{-79} to + or − 7.2 * 10^{75}. Numbers may be expressed in integer, floating-point, or in exponential form (E-type notation).

BASIC provides either all single-precision or all double-precision floating-point calculations.

The format of converted numeric data (for example, as converted by a PRINT statement) depends upon the BASIC system generated (see *Loading and Managing Extended BASIC*). The least significant digit of any printed number is always rounded.

Single-Precision Print Representation

BASIC does not use exponential format for any floating-point or integer number of six digits or less. BASIC prints a floating-point or integer number that requires more than six digits in the following E-type notation:

(sign)n.nnnnnE(sign)XX

where:	represents:
n.nnnnn	An unsigned number carried to five decimal places with trailing zeros suppressed
E	Times 10 to the power of
XX	An unsigned exponential value

The following table of values illustrates E-type notation for single precision:

Number	Single-Precision Output Format
2,000,000	2E+06
108.999	108.999
.0000256789	2.56789E-05
24E10	2.4E+11

Double-Precision Print Representation

BASIC does not use exponential form for any floating-point or integer number of eight digits or less. BASIC prints a floating-point or integer number that requires more than eight digits in the following E-type notation:

(sign)n.nnnnnnnE(sign)XX

where:	represents:
n.nnnnnnn	An unsigned number carried to seven decimal places with trailing zeros suppressed
E	Times 10 to the power of
XX	An unsigned exponential value

The following table of values illustrates E-type notation for double precision:

Number	Double-Precision Output Format
.666666666	.66666667
108.999868	108.99987
111111111.99	1.1111111E+08

Internal Number Representation

Internally, BASIC stores numbers in a format compatible with other Data General Corporation software such as FORTRAN IV and the relocatable assemblers. Single-precision floating point numbers are stored in two consecutive 16-bit words of the form:

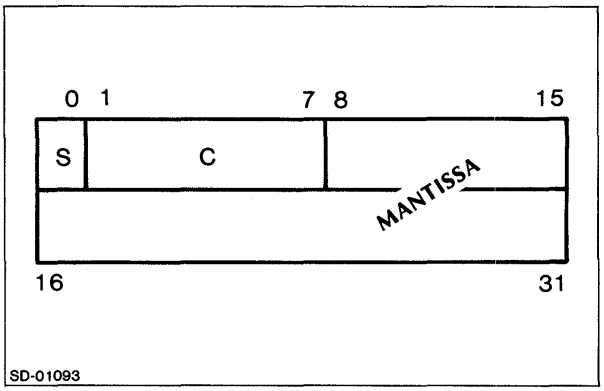


Figure 2-1.

where:	represents:
S	The sign of the mantissa: 0 is positive, 1 negative
C	The characteristic: an integer expressed in excess-64 code
Mantissa	A normalized six-digit hexadecimal fraction.

Double-precision floating-point numbers add two words of precision to the mantissa, which can be represented as:

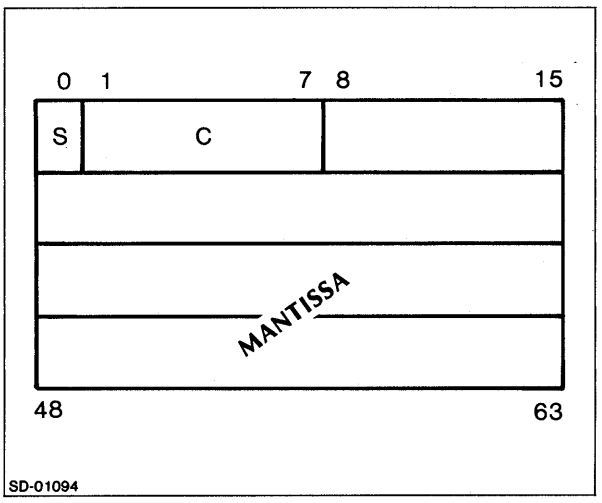


Figure 2-2.

The internal floating-point precision is 6 to 7 decimal digits for single precision and 13 to 15 decimal digits for double precision.

There may appear to be a BASIC rounding problem because of the way floating-point numbers are converted from their ASCII representation and stored internally. This restriction is noticed more frequently in single-precision BASIC. With the following code, single precision yields 4960.71, double precision 4960.72.

```
* 10 LET X=6179.92
* 20 LET Y=1219.2
* 30 PRINT X-Y
```

Floating point numbers are stored internally in binary with a normalized hexadecimal mantissa. When, for example, converting ASCII .4 to binary, a slight error is introduced: .4 is stored as a number slightly smaller than .4, since the fraction 4/10 cannot be represented exactly in binary. PRINT USING can be used to override the PRINT format and display the extra digits, for example:

```
* 10 INPUT X          ! X=.4
* 20 Y=X*10000
* 30 Z=INT(Y)         ! INT function truncates fractional
                       portion
* 40 W=Z/10000
* 50 PRINT X,Y,Z,W    ! yields .4 4000 3999 .3999
* 60 PRINT USING "###.###",W   ! yields 0.400
* 70 STOP
```

Since the internal representation of noninteger numbers may not be exact, it is advisable to test for a range of values when testing for a noninteger. For example, if the result of computation A was to be 1.0, a reliable test for 1 is:

```
IF ABS(A-1.0) < 1.0E-6 THEN...
```

If this test succeeds, A is equal to 1 within 1 part in 10↑6. This is approximately the accuracy of single-precision floating-point calculations.

Because of the method of storage of real numbers, the PRINT statement prints the value of A=(25.1*10) as 251, even though the actual internal representation of 25.1 * 10 is slightly less than 251. However, the command PRINT INT(25.1*10) truncates the final value to 250. This is due to the truncation of the fractional part by the INT function.

Numeric Variables

Express the name of a numeric variable (var in formats) as either a single letter or a single letter followed by a digit. For example:

Acceptable Variable Names	Unacceptable Variable Names
A	6A
A3	AZ
Z	B14
Z6	

BASIC also permits string variables (svar); see "String Variables" later in this chapter.

Arrays

An array represents an ordered set of values. Each member of the set is called an array element. An array can have either one or two dimensions. An array name may be a single letter or a single letter followed by a digit.

Array Elements

Each of the elements of an array is identified by the name of the array followed by a parenthesized subscript.

A subscript value must be a positive 1 or greater. Array B3 in the figure below illustrates a one-dimensional array with six elements.

Each element in a two-dimensional array (or matrix) has two subscript values. The first subscript number identifies the row; the second number identifies the column. Array C in the following figure is a two-dimensional array with six elements.

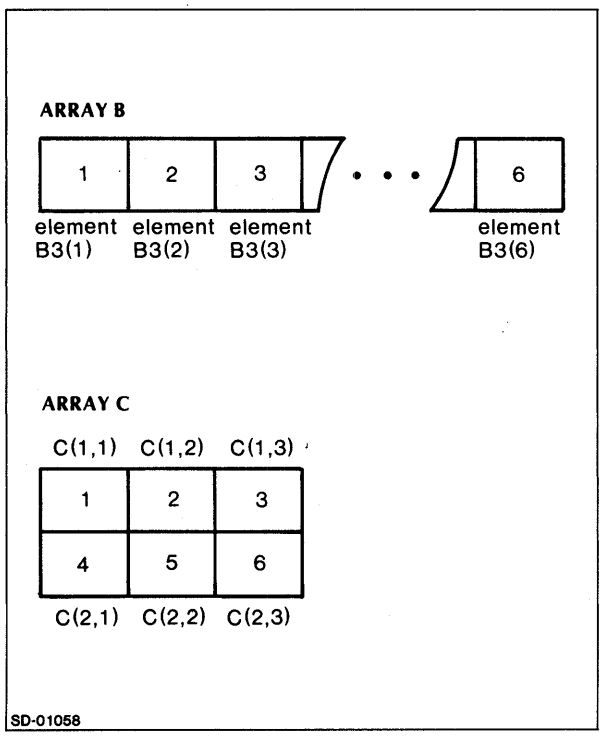


Figure 2-3.

Declaring an Array

An array is typically declared in a DIM statement, which names the array and sets its dimensions. You can also dimension an array in a MAT INPUT, MAT READ, MAT READ FILE, or MAT assignment statement. ("MAT" is an abbreviation for *matrix*.)

The *bounds* of an array are the values that determine the lowest and highest subscript values. The lower bound of a dimension is always 1; the upper bound is set in a DIM statement, as in this example:

```
* 10 DIM A(15), B1(2,3)
```

The upper bound of array A is 15; the upper bounds of matrix B1 are 2 and 3. These statements will also declare arrays:

```
* 50 MAT A = CON (4,5)
* 60 MAT READ A (4,5)
* 70 MAT A = B*C
```

The upper bounds of array A are determined by the expressions in the MAT assignment statements and by the subscripts in MAT READ.

If you do not declare an array, a default upper bound of 10 is assigned to each dimension of the array. Thus, if you have not dimensioned C, the following statement creates array C with 10 elements, with the fifth element set to 1:

```
* 100 C(5) = 1
```

If you try in this way to create an array with dimensions greater than 10 without declaring the array, BASIC restricts the upper bound(s) to the default and displays the message ERROR 31- SUBSCRIPT. Thus with the following statement, C becomes a matrix of 100 elements, the error message is displayed, and the value 273 is not stored:

```
* 200 C(9,11) = 273
```

There is no limit on the number of elements in a given array other than restrictions of available memory. When you declare storage for an array, it is permanently allocated. You can redimension an array, which merely rearranges the initially allocated space. If you try to redimension an array to a larger size, the error message, ERROR 28-DIM OVFL appears.

Redimensioning an array using a DIM statement does not disturb the data in the array, as shown in the following example:

```
* LIST
0010 DIM A(12)
0020 FOR I=1 TO 12
0030     LET A(I)=I
0040 NEXT I
0050 DIM A(3,4)
0060 MAT PRINT A
0070 END
* RUN
 1   2   3   4
 5   6   7   8
 9   10  11  12

END AT 0070
*
```

A numeric scalar variable and a numeric array variable may share the same name. BASIC treats them as distinct variables, provided you declare the array in a DIM statement before referring to the scalar variable, as in the following example. Line 20 refers to the scalar variable A; line 50 refers to the array A.

```
* 10 DIM A(3,3)
* 20 A=5067
* 30 MAT A=CON
* 40 PRINT A
* 50 MAT PRINT A;
* RUN
 5067
 1   1   1
 1   1   1
 1   1   1

 END AT 0050
*
```

Arithmetic Operations

A *numeric expression* (expr in statement formats) is any combination of numbers, numeric variables, array variables and functions linked together by arithmetic operators. The operators used in writing numeric expressions are

Operator	Meaning	Example
+	Unary plus	A+(+B)
-	Unary minus	A+(-B)
↑	Exponentiation	A↑B (A to the B power)
*	Multiplication	A*B
/	Division	A/B
+	Addition	A+B
-	Subtraction	A-B

Note: You cannot raise a negative number to a fractional exponent.

Priority of Arithmetic Operations

BASIC evaluates a numeric expression in the following order, proceeding from left to right:

1. BASIC evaluates any expression within parentheses before any unparenthesized expression. When parenthesized expressions are nested, BASIC evaluates the innermost expression first.
2. Unary plus and minus
3. Exponentiation. In a series of exponentiations, evaluation proceeds from left to right; that is, A↑B↑C = (A↑B)↑C.
4. Multiplication and division (equal priority)
5. Addition and subtraction (equal priority)
6. When two operators have equal priority, evaluation proceeds from left to right.

The following example illustrates the priorities of operators in evaluating an expression.

Z+(-A)+B*C↑D

1. A is negated.
2. C↑D is evaluated.
3. B is multiplied by the result of step 2.
4. Z is added to the result of step 1.
5. The result of step 4 is added to the result of step 3.

Parentheses

Since BASIC evaluates parenthesized expressions first, parentheses can change the order of evaluation of an expression. The following example uses the same variables as the previous example. Note the differences in the evaluation steps:

Z-((A+B)*C)↑D

1. A + B is evaluated.
2. The value from step 1 is multiplied by C.
3. The value from step 2 is raised to the D power.
4. The value from step 3 is subtracted from Z.

Parentheses can clarify the readability of an expression without affecting its final value. For example, the following expressions are equivalent, although their orders of evaluation differ somewhat:

A * B ↑ 3/4 + B/C + D ↑ 3

((A*B↑3)/4) + ((B/C) + D ↑ 3)

Relational Operators and Expressions

Relational operators compare two expressions in a relational expression (rel-expr in statement formats). A relational expression has the form:

expr1 relational operator expr2

The relational operators used in BASIC are:

Symbol	Meaning	Example
=	Equal	A = B
<	Less than	A < B
<=	Less than or equal	A <= B
>	Greater than	A > B
>=	Greater than or equal	A >= B
<>	Not equal	A <> B

Relational operators also may compare strings instead of numeric expressions (see the following section).

String Data

String Literals

A *string* is a sequence of characters that may include letters, digits, spaces, and special characters. A *string literal,* or *constant,* is a string enclosed within quotation marks. String literals are often used in PRINT and INPUT statements as illustrated in the following sample:

```
* 050 REM THE NEXT STATEMENT PRINTS A STRING
* 100 PRINT "THIS IS A STRING LITERAL"
* 150 REM STATEMENT 200 INCLUDES A
* 160 REM STRING PROMPT
* 200 INPUT "X= ",X
```

BASIC does not print the enclosing quotation marks when the string is output to a terminal. You can include special and nonprinting ASCII characters in string literals by enclosing the decimal equivalent of the character in angle brackets (< >). (See appendix D for the decimal equivalents of ASCII character codes.) The following program uses the decimal equivalent of a quotation mark to print it as part of a string:

```
* 10 PRINT "USE DECIMAL 34 TO PRINT <34>"
* RUN
USE DECIMAL 34 TO PRINT "

END AT 0010
*
```

String Variables

Extended BASIC allows string variables as well as string literals. A *string variable name* consists of a letter or a letter and a digit, followed by a dollar sign ($).

Legal String Variables	Illegal String Variables
A$	A14$
A2$	AA$
D6$	2$
	2C$
	A1

BASIC assigns string values to string variables with the LET, INPUT, READ, and LREAD statements.

Dimensioning String Variables

You may declare the length of a string variable with a DIM statement. The length of a string must be in the range 1-32767 characters.

In the following DIM statement, the string A$ has a maximum length of 25 characters, B3$ a maximum length of 200.

```
* 10 DIM A$ (25), B3$ (200)
```

If you do not declare a string variable in a DIM statement, BASIC assumes a maximum length of 10 characters and truncates values longer than 10 characters. Good programming practice suggests that you dimension all string variables, regardless of size.

In the following example, note that BASIC has truncated the string to its assigned dimension: 15 characters, including spaces.

```
* 10 DIM A2$(15)
* 20 LET A2$="PRINT A2$ IS TOO LONG"
* 30 PRINT A2$
* RUN
PRINT A2$ IS TO

END AT 0030
*
```

Substrings

You can select portions of strings, or *substrings,* in program statements and functions by using subscripts. Subscripted variables have the form:

svar $\left[\left\{\begin{matrix} x \\ y,z \end{matrix}\right\}\right]$

where:	represents:
svar	A string variable name
x	The xth through last character of svar
y,z	The yth through zth characters of svar

For example, given string variable A$:

A$	Refers to the entire string.
A$(2)	Refers to the second through last character in the string, inclusively.
A$(I)	Refers to character I through the last character in the string, where I evaluates to a character position less than or equal to the string length.
A$(3,7)	Refers to characters 3 through 7, inclusively.
A$(I,J)	Refers to characters I through J, where I <= J.
A$(1,1)	Refers to only the first character in the string.

When referring to substrings, *y* must not be greater than *z,* except in the special case where *z* equals 0.

Assignment	Extraction
If either *x* or *y* is 0, it defaults to the current length of the string plus 1.	If either *x* or *y* is 0, it defaults to 1.
If *z* is 0 and there are fewer characters provided than would fill the string to its current or dimensioned length, no blanks are padded.	If *z* is 0, *z* defaults to the current length of the string.
x,y,z must not be greater than the dimensioned length.	*x* and *y* must not be greater than the current length.

When an assignment provides too many characters, BASIC truncates the extra characters. When an assignment provides too few characters, blanks are added to the remaining positions. When *x* or *y* is more than one character beyond the current length in an assignment, BASIC generates an error.

The following examples illustrate the use of substrings:

```
* LIST
0005 DIM A$(20)
0010 LET A$(1,3)="SUB"
0020 LET A$(4,10)="STRING "
0030 LET A$(11,17)="EXAMPLE"
0040 PRINT A$
* RUN
SUBSTRING EXAMPLE

END AT 0040
*
```

You can use substrings to change the value of a string variable during a program, for example:

```
* LIST
0010 LET A$="ABCDEF"
0020 PRINT A$
0030 LET B$="1"
0040 LET A$(3,3)=B$
0050 PRINT A$
0060 LET A$(4)=B$
0070 PRINT A$
* RUN
ABCDEF
AB1DEF
AB11

END AT 0070
*
```

Assigning Values to String Variables

READ and DATA statements can assign values to string variables. When you include string data in a DATA list, enclose the string elements in quotation marks.

```
* LIST
0005 DIM A1$(2),B$(10),D$(5)
0010 READ A,A1$,B$,C,D$
0015 PRINT A,C,D$
0020 DATA 5,"ABCD","EFGH",10,"IJKL"
* RUN
5   10   IJKL

END AT 0020 *
```

As this example demonstrates, you may mix string data and numeric data in a DATA list. However, each variable in the READ statement must correspond to the type (numeric or string) of its matching element in the DATA list, or an error message results.

INPUT statements can also input string data to a program. The response at the INPUT prompt (?) may be enclosed in quotation marks, which are optional. If the INPUT statement requests data for more than one string variable, separate the data entered for each string with a comma or a carriage return. You may include commas in an INPUT string by enclosing the entire string in quotation marks, as in the following example:

```
* 10 INPUT A$,B$,C,D,E$
   .
   .
   .
* RUN
?ABCD,"EF,GH",2,4,"SIX"
```

To include quotation marks, enclose 34 in angle brackets. Exercise caution when you include NULL <0>, FORM FEED <12>, or CR <13> for RDOS/DOS systems, or NL <10> for AOS and AOS/VS systems, since these characters are string delimiters.

If you want to assign exactly what is typed, use the LREAD statement. LREAD does not strip leading or trailing blanks, does not use commas for delimiters, and does not process angle brackets.

Strings in IF-THEN Statements

You can also use strings in the relational expression of an IF-THEN statement. In this case, BASIC compares strings character by character on the basis of the ASCII character value (see appendix D) until it finds a difference. If a character in a given position in one string has a higher ASCII code than the character in that position in the other string, the first string is greater. If the characters in the same positions are identical, but one string has

more characters than the other, the longer string is greater. For example:

```
* 200 LET A$="ABCDEF"
* 300 LET B$="25 ABCDEFG"
   .
   .
   .
* 310 IF A$>B$ GOTO 500(True: transfer occurs.)
* 320 IF A$>B$(4) GOTO 500(False: no transfer.)
* 330 IF A$(1,4)=B$(4,7)GOTO 500
                          (True: transfer occurs.)
```

String Concatenation

You may concatenate string variables and string literals on the righthand side of LET statements, using a comma as the concatenation operator. For example:

```
* 10 DIM A$(50),B$(50)
* 15 LET A$="@$2.50,PROFIT MARGIN IS 15%"
* 20 LET B$=A$(1,4),"25",A$(7,26),"1%"
* 30 PRINT B$
* RUN
@$2.25,PROFIT MARGIN IS 11%
```

String concatenation, therefore, allows the following statement:

```
* 10 A$=A$,B$
```

where A$ is concatenated with B$ to yield a new value of A$.

Note: When concatenated strings are assigned to a string variable, the result of the concatenation is calculated *after* the assignment is made. Be careful when you use the same string variable on both sides of an assignment. For example, the statement

```
* 10 A$=B$,A$
```

does not use a temporary string to do the concatenation; therefore, the value of A$ on the right side of the equal sign is not the original value, but rather the value after B$ has been assigned. For example:

```
* LIST
0003 A$ = "12345"
0006 B$ = "A"
0010 A$=B$,A$
0020 PRINT A$
* RUN
AA

END AT 0020
*
```

String Arithmetic

You can perform arithmetic on string variables and string literals. BASIC executes the arithmetic operation, provided the strings (or substrings that begin at the first character of the string) have legal numeric values. BASIC ignores any alphanumerics that follow the numeric substring. If the substring is not a legal number, an error condition occurs.

Valid String	Invalid String
"123"	"FRED"
"123."	"123.E+FRED"
"-123"	"-+123"
"-123.E5"	"FRED123"
"-123.E-5FRED"	

Notice that BASIC permits decimal points, signs, and exponential format in the substring as long as they conform to the numeric representation described at the beginning of this chapter.

You may use the operators +,–,*, and / to link strings and to create an expression to be evaluated numerically. Do not use the concatenation character (,) in a string arithmetic expression.

```
* LIST
0010 LET A$="1234GEARS"
0020 LET B$="5678GEARS"
0030 PRINT A$+B$+"10"
* RUN
6922.

END AT 0030
*
```

BASIC returns 18 digits of precision when string arithmetic calculations are made. If any precision is lost, an error message is output. For example:

```
* 10 PRINT "123E27"*"1"
```

causes an error since the result cannot be represented by 18 digits.

End of Chapter

Chapter 3
BASIC Statements, Commands, and Functions

This chapter is an alphabetical reference for the keywords of Extended BASIC. Each description indicates the operating system(s) on which the word may be used; whether the word can be a statement, command, or function; the word's general syntactic format; remarks about the word's use; and one or more examples.

Table 3-1 groups the keywords according to their general purposes.

Table 3-1 Extended BASIC Keywords: Categories of Use (continues)

RDOS Privileged Commands (Execute only at the system terminal.)				
ALL	DIR	FALL	INIT	MAX
CDIR	DISABLE	FMSG	KILL	RELEASE
CPART	ENABLE	FREE		
Editing Commands				
.(period)	.C	.P		
.A	.E			
Functions				
ABS(x)	DEF FNa(d)	LOG(x)	SGN(x)	SYS(x)
ATN(x)	EOF(x)	ORD(X$)	SIN(x)	TAB(x)
CHR$(A)	EXP(x)	POS(x$,y$,z)	SQR(x)	TAN(x)
COS(x)	INT(x)	RND(x)	STR$(x)	VAL(x$)
CPU(x)	LEN(x$)			
Matrix Manipulation				
Addition & subtraction	DET	MAT INPUT FILE	MAT READ FILE	Multiplication
	IDN	MAT PRINT	MAT TINPUT	TRN
Assignment	INV	MAT PRINT FILE	MAT WRITE FILE	ZER
CON	MAT INPUT	MAT READ		
File I/O				
CLOSE FILE	LWRITE FILE	MAT WRITE FILE	READ FILE	
GPOS FILE	MAT INPUT FILE	OPEN FILE	RESET	
INPUT FILE	MAT PRINT FILE	PRINT FILE	SPOS FILE	
LREAD FILE	MAT READ FILE	PRINT FILE USING	WRITE FILE	

Table 3-1 Extended BASIC Keywords: Categories of Use (concluded)

General Commands and Statements				
ACL	ECHO	LEVEL	ON GOTO	RUN
AUDIT	END	LIBRARY	ON GOSUB	SAVE
BYE	ENTER	LIST	PAGE	SEARCHLIST
CALL	ERASE	LOAD	PRINT	SHARE
CARDS	ESC	LOCK	PRINT USING	SIZE
CHAIN	FILE	LREAD	PUNCH	STOP
CHAR	FN	LWRITE	RANDOMIZE	TAB
CHATR	FOR-NEXT	MSG	READ	TIME
CLI	GDIR	NEXT	REM	TINPUT
CON	GOSUB	NEW	RENAME	UNLOCK
DATA	GOTO	NOECHO	RENUMBER	UNSHARE
DELAY	HELP	NOESC	RESTORE	USERS
DELETE	IF-THEN	NOMSG	RETRY	WHATS
DIM	INPUT	ON ERR	RETURN	WHO
DISK	LET	ON ESC		

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	✓
F	

. (period)

Sends the line in the edit buffer to working storage.

Format

Remarks

1. The period command is a keyboard edit command. The editing commands work by manipulating statements in the edit buffer. BASIC allows you to put one statement into the buffer, change it, and reinsert it into the program. You may use the LIST command at any time to put a line into the edit buffer.
2. The system always inserts the last LIST statement into the buffer. If a statement merged with ENTER causes a syntactic error, the system automatically inserts it into the buffer. If you type a line that causes a syntactic error, BASIC puts that line into the edit buffer. Note that BASIC can hold only one line at a time in the edit buffer.
3. You need only the period command when you use the .A or .E commands. The .C command automatically sends the line back to working storage).

Example

```
* 10 INPUT WHAT IS YOUR CAPITAL?",A
ERROR 2 - Statement or command syntax is invalid
* .E/WHAT/"WHAT/
10 INPUT "WHAT IS YOUR CAPITAL?",A
* .
10 INPUT "WHAT IS YOUR CAPITAL?",A
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	✓
F	

.A

Appends to the line in the working storage edit buffer.

Format

.A string

Argument

string — A string literal, without quotation marks, that you want to append to the line

Remarks

1. The .A command is a keyboard edit command. The editing commands work by manipulating statements in the edit buffer. BASIC allows you to put one statement into the buffer, change it, and reinsert it into the program. You may use the LIST command at any time to put a line into the edit buffer.
2. The system always inserts the last LIST statement into the buffer. If a statement merged with ENTER causes a syntactic error, the system automatically inserts it into the buffer. If you type a line that causes a syntactic error, BASIC puts that line into the edit buffer. Note that BASIC can hold only one line at a time in the edit buffer.
3. .A appends the string argument to the buffer line, but it does not send that line into working storage to be interpreted. Use the period command (.) to send the line to working storage.
4. .A appends only to the line currently in the edit buffer. Use the .P command to see what is currently in the edit buffer.

Example

```
* 10 LET A=
ERROR 2 - Statement or command syntax is invalid
* .A3
10 LET A=3
* . (Period sends edited line 10 to working storage.)
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	✓
F	

.C

Changes a string in a line in the edit buffer and passes the line to working storage.

Format

./*n*/C/string1/string2

Arguments

string1 — A string literal, without quotation marks, in the line in the edit buffer. It is the string you want to change.

string2 — The string literal, without quotation marks, that is to replace string1

n — A non-negative number that designates which string1 you wish replaced by string2. If n=0, all string1's in the buffer are replaced with string2's

Remarks

1. The .C command is a keyboard edit command. The editing commands work by manipulating statements in the edit buffer. BASIC allows you to put one statement into the buffer, change it, and reinsert it into the program. You may use the LIST command at any time to put a line into the edit buffer.
2. The system always inserts the last LIST statement into the buffer. If a statement merged with ENTER causes a syntactic error, the system automatically inserts it into the buffer. If you type a line that causes a syntactic error, BASIC puts that line into the edit buffer. Note that BASIC can hold only one line at a time in the edit buffer.
3. .C changes the first occurrence of string1 to string2 and automatically passes the edited line to working storage to be interpreted.
4. .C changes only the line currently in the edit buffer. Use the .P command to see what is currently in the edit buffer. Use the LIST command to put a line in the edit buffer.
5. The system does not translate lowercase letters into uppercase letters in string1 or string2; it considers lowercase letters distinct from uppercase letters.

.C (continued)

Examples

```
* LIST
0010 LET A=1
0020 FOR I=1 TO 31
0030      LET A1=A
0040      LET A2=A+A1
0050      PRINT A2
0060      LET A1=A2
0070 NEXT I
* LIST 60
0060 LET A1=A2
* .C/A1/A
0060 LET A=A2
*
```

You can use .C to change several mistakes in the same line:

```
* LIST 40
0040 LET A=A+A
* .C/A/A2/
0040 LET A2=A+A
* LIST 40
0040 LET A2=A+A
* .3C/A/A1/
0040 LET A2=A+A1
```

You can also use .C to make several changes at one time, if they are all the same:

```
* 0010 REM "THIS IS A CORRECT LINE'
* LIST 10
0010 REM "THIS IS A CORRECT LINE'
* .C/'/"/"
0010 REM "THIS IS A CORRECT LINE"
* .0C/R/Z/
* LIST 10
0010 ZEM "THIS IS A COZZECT LINE"
ERROR 2 - Statement or command syntax is invalid
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	✓
F	

.E

Changes a string in a line in the edit buffer.

Format

.E/string1/string2/

Arguments

string1	A string literal, without quotation marks, in the line in the edit buffer. It is the string you want to change.
string2	The string literal, without quotation marks, that is to replace string1

Remarks

1. .E is a keyboard edit command that performs the same function as a .C command, without passing the modified line to working storage.
2. .E changes only the line currently in the edit buffer. Use the period command (.) to send the edited line to working storage to be interpreted.
3. The system does not translate lowercase letters into uppercase letters in string1 or string2; it considers lowercase letters distinct from uppercase letters.

Example

```
* LIST 25
0025 DIM A$(35),B$(39)
* .E/35/40/
0025 DIM A$(40),B$(39)
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	✓
F	

.P

Displays the contents of the edit buffer.

Format

.P

Remarks

1. The .P command is a keyboard edit command. The editing commands work by manipulating statements in the edit buffer. BASIC allows you to put one statement into the buffer, change it, and reinsert it into the program. You may use the LIST command at any time to put a line into the edit buffer.
2. The system always inserts the last LIST statement into the buffer. If a statement merged with ENTER causes a syntactic error, the system automatically inserts it into the buffer. If you type a line that causes a syntactic error, BASIC puts that line into the edit buffer. Note that BASIC can put only one line at a time in the edit buffer.
3. Once a line is in the edit buffer, you can edit it using the .A, .C, .E, or period (.) commands.
4. .P displays only the contents of the edit buffer.

Example

```
* THIS IS MONDAY
ERROR 2 - INVALID SYNTAX
* .P
THIS IS MONDAY
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	
F	✓

ABS(X)

Returns the absolute (positive) value of expr.

Format

ABS(expr)

Argument

expr A numeric expression

Example

```
* LIST
0010 PRINT ABS(-30)
* RUN
30

END AT 0010
*
```

AOS, AOS/VS	✓
RDOS, DOS	

S	✓
C	✓
F	

ACL

Prints a report of, or changes the Access Control List for a file in your directory.

Format

ACL "filename" *[, "userID", "attributes"]...*

Arguments

filename A string literal or string variable that evaluates to a filename in your directory

userID A string literal or string variable that evaluates to your identification

attributes One or more letter designations for the attributes listed below. An attribute controls the specified user's access to the named file, in the manner described below.

- R Read Access. The user may only examine the data in the file.
- W Write Access. The user may modify data in the file.
- O Owner Access. The user may change the Access Control List for the file, delete the file, or rename the file.
- E Execute Access. Without E access, BASIC treats the file as an execute-only file. BASIC generates an error message if the user tries to open the file, or attempts a LIST or SAVE after using ENTER or LOAD on the file.
- A Append Access. This attribute has no meaning for nondirectory files. For directory files, append access permits the user to make entries in the directory.

Remarks

1. When you create a file in your own directory, the file automatically has all five attributes, OWARE, for your userID.
2. The ACL command allows you to change the Access Control List to allow others to have full or partial access to your file, or to change your own access privileges.
3. String the file attributes together in the attributes argument without any delimiting spaces or punctuation. You may express them as either string literals or string variables.
4. The ACL command, followed only by the filename argument prints the current Access Control List for filename.
5. For RDOS and DOS systems the CHATR statement provides a similar facility.

Example

```
* ACL "PAGE2.2","JOE","REWAO","MARK","RE"
* ACL "PAGE2.2"
JOE,OWARE MARK,RE
```

In this example, JOE has all access privileges to PAGE 2.2, and MARK is limited to read and execute privileges.

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

ALL

Transmits a message to all active users who have not disabled message reception with the CHAR or NOMSG command.

Format

ALL message

Argument

message The text of the message. Quotation marks are unnecessary.

Remarks

1. ALL must be executed from the master terminal. In a single-user system, the master terminal and user terminal are the same. In multiuser environments, programs using ALL as a statement do not work when run from terminals other than the master terminal.
2. All active users on the system who have not disabled message reception receive the transmitted message on their terminals. The message appears as follows:

   FROM OPER:message

   OPER is the ID associated with the master terminal.
3. No error message appears at the master console if a user has disabled message reception.
4. Message reception is limited to one line per ALL command.

Example

If the system operator types

```
* ALL LINE PRINTER OUTPUT IS READY
```

the users see displayed on their screens

FROM OPER: LINE PRINTER OUTPUT IS READY

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	
F	✓

ATN(X)

Calculates in radians the angle whose tangent is: expr (-pi/2 < result < pi/2).

Format

ATN(expr)

Argument

expr A numeric expression

Example

```
* LIST
0010 REM-CALCULATE ANGLE WHOSE TAN=2
0020 PRINT ATN(2)
* RUN
 1.1071487

END AT 0020
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

AUDIT

Copies console input and output to a file named by argument.

Format

AUDIT *["filename"]*

Argument

filename An optional string literal or string variable to represent the audit file that would contain console input and output

Remarks

1. Using AUDIT as a command allows copying of both terminal input and output.
2. Only one audit file can be in effect at a time.
3. AUDIT without a filename shuts off the AUDIT operation.
4. If the audit filename already exists, the new information is appended to the file.

Example

```
* LIST
0010 DELETE "COPY.DT" ! DELETE OLD FILE
0020 LET A$ = "COPY.DT" ! FORM AUDIT ARG
0030 AUDIT A$ ! OPEN "COPY.DT" AS AUDIT FILE
0040 LET A = 3 ! ASSIGN A
0050 PRINT -A ! OUTPUT -3 TO CONSOLE
0060 AUDIT ! CLOSE AUDIT FILE
0070 END
```

When this program is run, the file "COPY.DT" will contain the three ASCII bytes "-3 <012>", which are output to the terminal.

AOS, AOS/VS	✓
RDOS, DOS	

S	✓
C	✓
F	

BYE

Depending on how your system is configured, signs off from BASIC and returns one level, or returns to the CLI, or logs off.

Format

BYE

Remarks

You can use BYE as a console command or as a program statement to exit from BASIC.

Example

```
* BYE
)
```

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

BYE

Signs off from the BASIC system and makes the terminal available to others.

Format

BYE

Remarks

1. You can use BYE as a console command or as a program statement to log off BASIC.
2. When issued from the master terminal, BYE exits to the operating system CLI, unless one or more users are still logged on. In this case, the following error message is displayed at the master terminal, and you are not be able to exit from BASIC:

   ERROR 64 - System active

   Use FALL to inform all active users that you want to terminate BASIC, then try the BYE command again.
3. BASIC displays accounting information after you enter the BYE command.
4. BYE severs telephone connections.
5. The system does not recognize the ESC key until the messages that follow BYE have been output.
6. Do not disconnect your terminal until the BYE sequence is complete.

Example

```
* BYE
xxxx 01/02/83 10:06 SIGN OFF, 04
xxxx 01/02/83 10:06 CPU USED, 206
xxxx 01/02/83 10:06 I/O USED, 11, 137

BASIC xx.xx
Ready
```

The first two lines show that you are using terminal 04 and that you used 206 seconds of CPU time. The third line shows that you made 11 file I/O calls and 137 BASIC I/O calls.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

CALL

Calls a subroutine written in assembly language from an Extended BASIC program.

Format

CALL subr *[,expr]...*

Arguments

subr A positive integer representing an assembly language subroutine number

expr As many as eight optional arguments to be passed to the subroutine. Arguments may be arithmetic or string variables, or expressions.

Remarks

1. You must initiate all variable arguments passed to an assembly language subroutine before using them in a CALL statement; otherwise an error message occurs.
2. CALL subroutines do not accept arrays, including dimensioned variables, as arguments, unless they include subscripts that indicate the one element to be passed to the CALL.
3. Chapter 4 describes creating assembly language subroutines that may be called from Extended BASIC programs.

Example

```
0005 LET A = 12
0010 LET B = A * 2
0015 CALL 33,A,B
  .
  .
  .
```

Statement 15 calls subroutine 33, with the values of A and B as arguments to the subroutine.

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

CARDS

Transfers and merges BASIC statement lines in DGC mark-sense card format from the card reader, other device, or disk file into your current program storage area.

Format

CARDS "filename"

Argument

filename A device or disk file, expressed as a string literal or variable

Remarks

1. If the filename is a disk file, BASIC searches for it in your directory first. If it is not found, BASIC searches the library directory for the filename.
2. When a statement line from the device or file has the same line number as a line in the current program, BASIC replaces the current statement.
3. CARDS provides a convenient method of entering statements from Extended BASIC mark-sense cards. (Use ENTER for statements not in mark-sense card format.)
4. CARDS permits both 80-column punch cards and 37-column DGC mark-sense cards in the same BASIC system: on machines with two card readers or one reader equipped to read both types of cards. However, you cannot mix 37- and 80-column cards in the same deck. You may enter mixed programs in two steps, as shown in the example.

Example

```
* NEW
* ENTER "$CDR"
* CARDS "$CDR"
```

BASIC merges statements on 80-column cards with statements on special DGC mark-sense cards.

AOS, AOS/VS	
RDOS DOS	✓

S	✓
C	✓
F	

CDIR

Creates a subdirectory.

Format

CDIR name

Argument

name A string literal or string variable of up to 10 characters

Remarks

1. CDIR must be executed from the master terminal. In a single-user system, the master terminal and user terminal are the same. In multiuser environments, any programs using CDIR as a statement do not work when run from terminals other than the master terminal.
2. Each user must be assigned to subdirectory, or to a secondary partition.
3. You may create the subdirectory in the master directory, or in any other primary or secondary partition.

Examples

```
* CDIR "SMITH"
*
```

Creates SMITH.DR in the current directory

```
* A$="JOE"
* CDIR A$
```

Creates JOE.DR in the current directory

AOS, AOS/VS	✓
RDOS, DOS	

S	✓
C	✓
F	

CHAIN

When encountered by the system, runs the specified separate program.

Format

CHAIN "filename" *[THEN GOTO line no.]*

Arguments

filename A string variable or string literal that evaluates to a disk filename or a device

line no. A line number in the program specified by the filename

Remarks

1. When BASIC encounters a CHAIN statement in a program, it stops execution of that program, retrieves the program named in the CHAIN statement from the specified device or file, and begins execution of the chained program.
2. If the program is on disk, the system searches your directory for the filename; if it is not found, the system searches the library disk directory.
3. If BASIC finds the filename, it clears your current program from memory and loads the file into memory. If BASIC does not find the filename, your current program remains in memory, and an error message occurs.
4. CHAIN does not change the status of files. Open files remain open, and current file position pointers are maintained.
5. A program must be in core image format before it can be chained, and may have been partially executed before it was saved.
6. By default, BASIC clears all variables from the new program and runs it from the lowest numbered statement. If you use CHAIN with THEN GOTO *line no.*, variables in the main program maintain the values they had when the program was saved, and the program runs from the specified line number. If the line number does not exist in the new program, BASIC loads the new program but an error message occurs. Also, when THEN GOTO *line no.* is used, the chained program must have been run, then saved, in order to establish the variable expansions.

Example

```
0010 READ A
0020 IF A > 5 THEN GOTO 0060
0030 IF A = 5 THEN GOTO 0070
0040 DATA 4,1,6,3,5
0050 GOTO 0010
0060 CHAIN "SERVICE"
0070 CHAIN "SUBR" THEN GOTO 0050
```

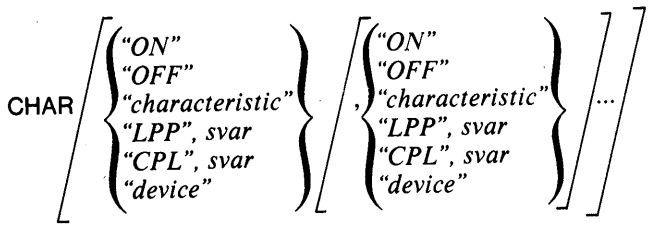
AOS, AOS/VS	✓
RDOS, DOS	

S	✓
C	✓
F	

CHAR

Changes or prints a report of the current device characteristics.

Format



Arguments

svar — A string variable or string literal that represents a numeric value

device — The name of a terminal expressed as a string variable or string literal

characteristic — A device characteristic, expressed as a string variable or string literal. See "Remarks" for a list of device characteristics.

Remarks

1. If you type the CHAR command without a list of arguments, BASIC prints a report listing the current device characteristics on your terminal.
2. The ON and OFF arguments apply only to the characteristics arguments, not to the device, LPP, or CPL arguments.
3. When you use the CHAR command, the keyword ON is in effect until BASIC encounters the keyword OFF. OFF then remains in effect until BASIC encounters ON.
4. LPP sets lines per page; CPL sets characters per line.
5. The CHAR command only changes the device characteristics specified in the command. It does not replace the existing device characteristics.
6. The characteristic FKT enables the user function keys of 6052/6053 DASHER TM display terminals. Each key generates a two-code sequence (see *DASHER Display Terminal 6052, 6053 Technical Reference,* 014-000077). If FKT is on, the function keys are recognized as AOS delimiters.
7. Available device names include the following:

   4010I (Infoton) Hardcopy
   6012 605x
   CRT4-15
8. You cannot set the MOD characteristic (device on a modem line) on or off with the CHAR command; it is displayed for your information only.
9. The device characteristics that you can turn ON or OFF are listed below. Not all characteristics apply to all devices.

ST	Simulate tab settings (eight columns).
SFF	Simulate form feed.
EPI	Require even parity on input.
WRP	Device wraps around when line too long.
SPO	Set even parity on output.
RAF	Send 21 rubouts after each form feed.
RAT	Send 10 rubouts after each tab.
RAC	Send 10 rubouts after each carriage return or NEW LINE.
NAS	Device is non-ANSI standard.
OTT	Old TTY. Convert 175 and 176 to 33 octal.
EOL	Do not execute automatic carriage return-line feed if CPL is exceeded.
UCO	Convert lowercase output to uppercase.
LT	Output 55 nulls upon open and close.
FF	Output a form feed upon open.
EB0	Echo all characters.
EB1	Echo all characters except control characters.
ULC	Input both uppercase and lowercase.
PM	Device is in page mode.
NRM	Disable message reception via ?SEND.
TO	Enable time-outs on reads and writes.
TSP	Do not suppress trailing blanks (card reader).
PBN	Use packed format on binary read (card reader).
ESC	ESC character produces an interrupt.
FKT	Enable function keys as delimiters.
CM4	Allow conversion of RDOS BASIC 4.XX programs to AOS BASIC 1.XX. (See "Remarks" in the next entry for cases.)

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

Examples

```
*CHAR
CRT2 LPP 24 CPL 80
ON STEP1 NAS EB0 ESC
OFF SFF SPO RAF RAT RAC OTT EOL UCO LT FF
EB1 ULC PM NRM MOD TU TSP PBN WRP
*CHAR OFF,ST
*CHAR
CRT2 LPP 24 CPL 80
ON EPI NAS EB0 ESC
OFF ST SFF SPO RAF RAT RAC OTT EOL UCO LT
FF EB1 ULC PM NRM TO MOD TSP PBN WRP
```

CHAR

Changes or prints a report of the current device characteristics.

Format

CHAR [[{ "ON" / "OFF" }] "characteristic" [,] ...]

Argument

characteristic A device characteristic, expressed as a string variable or string literal. See "Remarks" for a list of device characteristics.

Remarks

1. If you type the CHAR command without a list of arguments, BASIC prints a report listing the current device characteristics on your terminal.
2. When you use the CHAR command, the keyword ON is in effect until BASIC encounters the keyword OFF. OFF then remains in effect until BASIC encounters ON.
3. The CHAR command only changes the device characteristics specified in the command. It does not replace the existing device characteristics.
4. You cannot set the MOD characteristic (device on a modem line) on or off with the CHAR command; it is displayed for your information only.
5. The device characteristics which you can turn ON or OFF are listed below.

   NCR Do not echo carriage return.
   DSP Disable spooling.
   DLC Disable line feed after carriage return.
   XON XON/XOF protocol for $TTR (multiplexor terminals only)
   DNF Disable 20 nulls after form feed.
   NOE Do not echo input.
   BSP Backspace for rubout.
   DTS Disable tab simulation.
   ESC Escape character produces interrupt.
   NRM Disable message reception from other users.
   CM4 Allow conversion of 4.XX programs to 5.XX programs. (See remark 9 below.)

CHAR (continued)

6. For the CLI system console, the NCR, DLC, DNF, and DTS characteristics are valid only in input mode.
7. Only multiplexor consoles support the XON characteristic.
8. It is recommended that CHAR be used rather than NOECHO, ECHO, NOESC, and ESC.
9. Until you convert your revision 4.XX RDOS Extended BASIC programs to revision 5.XX RDOS Extended BASIC—or to revision 1.XX AOS Extended BASIC—you can use CHAR "CM4" to run your old programs in the following cases:

   a. FOR/NEXT

   Revision 5.XX or Revision 1.XX

   When the FOR/NEXT loop ends, the value of the control variable is the first value not used.

   Revision 4.XX

   When the FOR/NEXT loop ends, the value of the control variable is the last value used.

   b. TAB(X)

   Revision 5.XX or Revision 1.XX

   The first column of a line is column 1.

   If X is less than 1, then 1 is substituted.

   If X is less than the present column number, printing proceeds at column X on the next line.

   If X is 0, and the current column position is 1, the system does not generate a carriage return. Each file has its own column counter.

   Revision 4.XX

   The first column of a line is column 0.

   Negative tab arguments are treated as unsigned integers and evaluated modulo the page size.

   If X is less than the present column number, the TAB(X) function is ignored.

   If X is 0, the system always generates a carriage return.

   If X is 0, the system carriage return is always returned.

   The file column counter is reset to 0 at the beginning of each PRINT FILE statement.

   c. Substrings

   Revision 5.XX or Revision 1.XX

   Assignments with too few characters are padded with blanks in the remaining character positions.

   When the first argument in a substring assignment is 0, it defaults to current length plus 1.

   An error is generated when the first argument of a substring extraction is the current length plus 1 and the second argument is beyond the current length.

   Revision 4.XX

   If an assignment has too few characters, BASIC truncates the remainder of the string and updates the current length.

   When the first argument in a substring assignment is 0, it defaults to 1.

   A null character is returned when the first argument of a substring extraction is the current length plus 1, and the second argument is beyond the current length.

To return to revision 5.XX or revision 1.XX, use the command CHAR "OFF","CM4".

Example

```
* CHAR
ON XON ESC
OFF NCR DSP DLC DNF NOE BSP MOD DTS NRM
* CHAR "ON","BSP"
* CHAR
ON XON BSP ESC
OFF NCR DSP DLC DNF NOE MOD DTS NRM
*
```

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

CHATR

Changes, adds, or removes the resolution file attributes assigned to a file that already exists in your directory.

Format

CHATR "filename", attributes

Arguments

filename	A disk file in your directory, expressed as a string literal or string variable
attributes	File attributes described under "Remarks"

Remarks

1. Attributes that you can set with the BASIC CHATR command are compatible with those of the RDOS CHATR command.
2. You can string together file attributes in the attributes argument without delimiting spaces or punctuation.
3. The attributes given in the BASIC CHATR command replace existing attributes, unless you specify otherwise.
4. Any attempt to use CHATR on an open file generates an error condition.
5. The attributes that may be added or removed by the BASIC CHATR command are:

   P Permanent file. You cannot delete or rename the file.

   R Read-protected. You cannot access the file for reading.

   W Write-protected. You cannot alter the file.

   H Special BASIC argument to CHATR that specifies attributes H, P, and W. Special attribute in CLI is ?.

   E Special BASIC argument to CHATR that makes a file an execute-only file with attribute E. BASIC generates an error message if an execute-only file is opened, or if a LIST or SAVE is attempted on an execute-only file has been entered or loaded.

   O Special BASIC argument to CHATR that makes a file sharable with attribute H. Other users may access the file if they know its name and directory.

   0 Zero removes the current file attributes and adds the attributes to the right of 0. For example, 0RW removes all current file attributes and replaces them with RW.

   * Special BASIC argument to CHATR that preserves the current file attributes and adds those specified. The asterisk may be used only in conjunction with other attributes in the argument.

   + Plus preserves the current file attributes and adds those following the plus sign. It is identical to *.

   - Minus removes only those attributes following the minus sign.

Example

```
* WHATS "TESTFILE"
TESTFILE.  D  260
* CHATR "TESTFILE","WP"
* WHATS "TESTFILE"
TESTFILE.  WPD  260
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	
F	✓

CHR$(A)

Generates the character represented in the ASCII collating sequence by a number.

Format

CHR$(expr)

Argument

expr A numeric expression

Remarks

1. The argument of the character string function may be any numeric expression.
2. BASIC truncates the expression to an integer and computes the modulus 256 before returning a character.
3. If the expression is negative, BASIC returns an error message.

Example

```
* LIST
1000 REM CONVERT LOWER - TO UPPERCASE
1010   FOR I-1 TO LEN(A$)
1020   LET A=ORD(A$(I,I))
1030   IF A > =97 THEN IF A < =122 THEN LET
       A$(I,I)=CHR$(A-32)
1040 NEXT I
```

AOS, AOS/VS	✓
RDOS, DOS	

S	✓
C	✓
F	

CLI

Provides access to the command line interpreter without terminating the BASIC process.

Format

CLI *[command]*

Argument

command Any valid CLI command

Remarks

1. If you execute CLI with an argument, the BASIC process is temporarily suspended until the CLI command is completed; you are then returned to BASIC.
2. If you execute CLI without an argument, you remain at the command line interpreter, where you may execute a series of commands. To return to BASIC, type BYE.
3. For CLI to execute, CLI.PR must be in your searchlist.

Example

```
* CLI
)TIME
13:25:12
)DUMP @MT0:0 -.SR
)BYE
* CLI TIME
13:26:25
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

CLOSE FILE

Dissociates a filename and a file number so that the file is no longer referred to and the file number can be reused.

Format

CLOSE [{ FILE / # } (file)]

Argument

\# A synonym for the keyword FILE

file A numeric expression that evaluates to a file number previously associated with a filename in an OPEN FILE statement.

Remarks

1. You can use the CLOSE FILE statement to close a file, and then reopen it with a new mode argument.
2. Used without arguments, CLOSE closes all open files.

Examples

```
* 100 CLOSE FILE (1)
* 200 CLOSE FILE (X+3)
* 300 CLOSE
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	✓
F	

CON

Continues the execution of a program after a STOP statement in the program has been executed, after the ESC key has been pressed, or after an error has occurred.

Format

CON

Remarks

1. The CON command enables you to continue to execute a program from the point that it was interrupted.
2. If you interrupt a program by pressing the ESC key, CON continues the program with the statement immediately following the one that was executing when you pressed ESC.
3. If the system encounters a runtime error within the program, you may correct the error and issue the CON command to begin execution from the statement following the one in which the error occurred.

CON (continued)

Example

```
* LIST
0010 PRINT "PRINCIPAL INT(%) ";
0020 PRINT "TERM(YRS) TOTAL"
0030 READ P,I,T
0035 IF T=0 THEN GOTO 0080
0040 LET A=P*(1+I/100) ^ T
0050 PRINT P; TAB(12);I;
0055 PRINT TAB(21);T; TAB(32);A
0060 GOTO 0030
0070 DATA 1000,5,10,0,0,0
0080 PRINT
0090 PRINT "CHANGE DATA AT LINE 70"
0100 STOP
0110 GOTO 0010
* RUN
PRINCIPAL   INT(%)   TERM(YRS)   TOTAL
1000        5        10          1628.8946

CHANGE DATA AT LINE 70

STOP AT 0100
* 70 DATA 2500,3,10,1459,6,12,0,0,0
* CON
PRINCIPAL   INT(%)   TERM(YRS)   TOTAL
2500        3        10          3359.7909
1459        6        12          2935.7947

CHANGE DATA AT LINE 70

STOP AT 0100
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	
F	✓

COS(X)

Calculates the cosine of an angle that is expressed in radians.

Format

COS(expr)

Argument

expr A numeric expression specified in radians

Remarks

SYS(15), which is assigned the value of pi (3.1416), may be used in the argument. For more information see the SYS(X) function.

Example

```
* LIST
0010 REM- PRINT COSINE OF 30 DEGREES
0020 LET P=SYS(15)/180
0030 PRINT COS(30*P)
* RUN
.8660254

END AT 0030
*
```

AOS, AOS/VS	
RDOS only	✓

S	✓
C	✓
F	

CPART

Creates a secondary partition, *name*.DR.

Format

CPART name,size

Arguments

name A string literal or string variable of up to 10 characters

size The number of contiguous blocks—at least 48

Remarks

1. CPART must be executed from the master terminal. In a single-user system, the master terminal and user terminal are synonymous. In multiuser environments, any programs using this command as a statement do not work when run from terminals other than the master terminal.
2. Each user must be assigned to a secondary partition, or to a subdirectory.
3. You may create the secondary partition in the master directory, or in any other primary partition.
4. If the size is not an integer multiple of 16, the system truncates the size to the next lower multiple.
5. The name argument may be expressed as either a string variable or a string literal in quotes.

Examples

```
* CPART "JOE",64
*
```

Creates a secondary partition named JOE.DR.

AOS, AOS/VS	
RDOS, DOS	✓

S	
C	
F	✓

CPU(X)

Returns a value equal to the status of a CPU data switch or the numeric value of all 16 console switches.

Format

CPU(expr)

Argument

expr An expression that evaluates to the number of a CPU console switch or -1.

Remarks

1. The value returned is:

   0 if console switch is down
   1 if console switch is up

2. If the expression is -1, the CPU function returns the decimal value of all 16 console switches. This value is in the range 0-65535.
3. Each switch corresponds to a bit in a 16-bit word with the bits numbered 0-15, from left to right.

Examples

```
* 10 IF CPU(0) THEN GOTO 85
```

Proceed to statement 85 if console switch 0 is up.

```
* PRINT CPU(-1)
33
```

Switches 11, 12, 14, and 15 are up.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	
F	

DATA

Provides values for variables specified in a READ statement.

Format

DATA { val / "str lit" } [{ ,val / ,"str lit" }]...

Arguments

val, str lit Elements that form a list of numeric values and/or string literals

Remarks

1. You can use more than one DATA statement in a program.
2. The DATA statement is a nonexecutable statement. The values appearing in DATA statements form a single list. The first item in the lowest numbered DATA statement is the first element of this list. The last item in the highest numbered DATA statement is the last item in this list.
3. Both numbers and string literals may appear in a DATA statement; separate values from each other with commas.
4. Enclose string literals in quotes.

Example

```
* 100 DATA 1, 17, "AB,CD", -1.3E-13
```

See the READ and MAT READ statements for usage and additional examples.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	
F	✓

DEF FNa(d)

Permits you to define as many as 26 different functions that can be referred to repeatedly throughout a program.

Format

DEF FNa(d)=expr

Arguments

a A single letter from A to Z

d A dummy arithmetic variable that may appear in expr

expr An arithmetic expression that may contain a variable d

Remarks

1. Each function returns a numeric value.
2. BASIC does not relate the dummy variable named in the DEF statement to variables in the program with the same name; the DEF statement simply defines the function and does not cause any calculation to be carried out.
3. In the function definition, the expression may include other user-defined functions. In AOS and AOS/VS environments, functions may be nested to a depth of 10. In RDOS/DOS, functions may be nested to a depth specified by the system manager.
4. Function definition is limited to a single line DEF statement. Complex functions that require more than one program statement should be constructed as subroutines.

Examples

```
* LIST
0010 DEF FNE(J)=(J↑2)+2*J+1
0020 LET Y=FNE(5)
0030 PRINT Y
* RUN
 36

END AT 0030
*
```

Line 10 defines the FNE function. Line 20 refers to the FNE function and evaluates it using numeric argument 5.

You can also redefine a function, as in the following example:

```
* LIST
0010 DEF FNA(X)=X^2
0020 PRINT FNA(2)
0030 DEF FNA(Z)=Z^3
0040 PRINT FNA(2)
* RUN
4
8
END AT 0040
*
```

The following example illustrates the nesting of user-defined functions:

```
* LIST
0005 TAB =16
0010 LET P=SYS(15)
0020 DEF FNR(X)=X*P/180
0030 DEF FNS(X)=SIN(FNR(X))
0040 DEF FNC(X)=COS(FNR(X))
0050 FOR X=0 TO 45 STEP 5
0060      PRINT X,FNS(X),FNC(X)
0070 NEXT X
* RUN

0   0              1
5   8.7155743E-02
10  .17364818      .98480775
15  .25881905      .96592583
20  .34202014      .93969262
25  .42261826      .90630779
30  .5             .8660254
35  .57357644      .81915204
40  .64278761      .76604444
45  .70710678      .70710678

END AT 0070
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

DELAY

Delays program execution for a specified amount of time.

Format

DELAY = expr

Argument

expr A numeric expression that represents time in seconds, to the nearest tenth of a second (e.g., 10 = 10 seconds, 35.2 = thirty-five and two-tenths seconds).

Remarks

1. You can use DELAY to postpone program execution on an error condition before attempting a RETRY.
2. In AOS and AOS/VS, the maximum number for the expression is approximately 4,000,000. In RDOS and DOS, the maximum is approximately 65,000.

Example

```
0005 ON ERR THEN 100
0010 OPEN FILE (0,2), "THISFILE"
0020 LET I=0
   .
   .
   .
0100 IF SYS (7) <> 48 THEN 200
(Is anyone else using this file?)
0105 I = I + 1
0110 IF I > 10 THEN GOTO 200
(Allows 10 RETRY attempts.)
0120 DELAY=1.0
(One-second delay before retry.)
0125 RETRY
(Returns to statement which caused error.)
0200 STOP
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

DELETE

Removes a file from your directory.

Format

```
DELETE "filename"
```

Argument

filename A string literal or string variable that identifies a file in your directory that is not protected (see CHATR)

Remarks

1. If DELETE is executed, BASIC searches your directory and deletes the directory entry for the filename.
2. BASIC returns an error message if BASIC cannot find the file, if the file is delete-protected, or if it is not in your directory.

Example

```
* DELETE "TEST.SR"
*
```

BASIC removes file TEST.SR from your directory and frees the disk blocks which it occupied.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

DIM

Defines the size of one or more numeric variable arrays, or sets the maximum number of characters in a string.

Format

DIM { svar (n) / array (m) / array (row,col) } [{ ,svar (n) / ,array (m) / ,array (row,col) }]...

Arguments

svar A BASIC string variable name

array A BASIC numeric variable name

n The maximum string length

m An expression for the number of elements in a one-dimensional array

row An expression for the number of rows in the array

col An expression for the number of columns in the array

Remarks

1. Array Elements

   The concept of arrays is described in chapter 2. The DIM statement can declare the size of an array to be a number of elements other than the default upper bound (10) for each dimension. For example:

   ```
   * 10 DIM A(13),B(7,7),C(20,5)
   ```

   The initial value of all elements in an array is 0 until the program assigns other values.

   A subscript variable or expression must have a value between 1 and the upper bound given in the DIM statement. For example:

   ```
   * 01 DIM A (5,5)
   * 05 X=2
   * 10 PRINT A(1,X 2)
   ```

   If the subscript variable or expression does not evaluate to an integer, BASIC converts it to an integer using the INT function.

   If a subscript evaluates to an integer larger than the array's upper bound or smaller than 1, a subscript error condition occurs.

2. Redimensioning Arrays

   You can redimension a previously defined array during execution of a program by declaring the array in another DIM statement. The total number of elements of the newly dimensioned array must not exceed the original total number of elements.

```
* 100 DIM A(3,3)
  .
  .
  .
* 200 DIM A(2,3)
  .
  .
  .
* 300 DIM A(2,2)
```

   BASIC reassigns the values of elements in array A(3,3) to elements in array A(2,3) and then to elements in array A(2,2) as follows:

1 2 3 4 5 6 7 8 9	1 2 3 4 5 6	1 2 3 4
A(1,1) = 1 A(1,2) = 2 A(1,3) = 3 A(2,1) = 4 A(2,2) = 5 A(2,3) = 6 A(3,1) = 7 A(3,2) = 8 A(3,3) = 9	A(1,1) = 1 A(1,2) = 2 A(1,3) = 3 A(2,1) = 4 A(2,2) = 5 A(2,3) = 6	A(1,1) = 1 A(1,2) = 2 A(2,1) = 3 A(2,2) = 4

3. For a discussion on the dimensioning of strings, see chapter 2.

AOS, AOS/VS	✓
RDOS, DOS	

S	✓
C	✓
F	

DIR

Displays, sets or stores the pathname of the current working directory using DIR.

Format

DIR *[pathname]*

Argument

pathname An optional directory pathname expressed as a string variable or string literal

Remarks

1. DIR with no arguments returns the pathname of the current working directory.
2. If the pathname argument is a string literal or a non-null string variable, AOS BASIC sets the current working directory to the pathname specified in the argument.
3. If the pathname argument is a string variable with a current length of 0, DIR returns the pathname of the current working directory and stores this string in that variable.

Examples

Display the current working directory:

```
* DIR
:UDD:BASIC
```

Set the working directory:

```
* DIR ":UTIL"
* DIR
:UTIL
```

Set the working directory through a variable:

```
* A$=":BASIC"
* DIR A$
* DIR
:BASIC
```

Store the current working directory in a variable:

```
* DIR ":UDD:BASIC"
* A$=""
* DIR A$
* PRINT A$
:UDD:BASIC
```

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

DIR

Changes the current directory to another directory in the same or different partition and prints the name of your current directory.

Format

DIR [{ *primary part [[:secondary part[:subdirectory]* / *secondary part [:subdirectory]* / *subdirectory* / *null-length string variable* }]

Arguments

primary part	The target directory. Default: current directory
secondary part	The target subpartition under the primary partition
subdirectory	The target subdirectory under a primary or secondary partition
null-length string variable	A string variable name

Remarks

1. This command is an implementation of the RDOS CLI DIR command.
2. DIR must be executed from the master terminal. In a single-user system, the master terminal and user terminal are the same. In multiuser environments, programs using DIR as a statement do not work when run from terminals other than the master terminal.
3. DIR without any arguments is not privileged and returns the current directory.
4. If you do not specify the primary partition, the directory specified in the command is assumed to be in the current partition.
5. If necessary, DIR initializes the device or directory specified in the command if the current directory is a partition and the specified directory is a subdirectory below it. But if you are in a subdirectory, DIR does not initialize another subdirectory automatically. You can, however, always initialize another primary name to be placed in the string variable.
6. A null length string variable as the argument to DIR causes the current directory name to be placed in the string variable.
7. You can express the new directory name as either a string variable or as a string literal in quotes.

Examples

Change the current directory to subdirectory GHI in secondary DEF in primary partition DP1:

```
* DIR "DP1:DEF:GHI"
```

Change the current directory to subdirectory SAM in primary partition DP0:

```
* A$="DP0:SAM"
* DIR A$
```

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

DISABLE

Prevents the inadvertent use of the CTRL-A, CTRL-C, and CTRL-F (background) RDOS and DOS system console breaks.

Format

DISABLE

Remarks

1. DISABLE must be executed from the master terminal. In a single-user system, the master terminal and user terminal are the same. In multiuser environments, programs using DISABLE as a statement do not work when run from terminals other than the master terminal.
2. If you accidentally type one of the RDOS system terminal breaks, BASIC abnormally terminates. The DISABLE command issued at the BASIC master terminal prevents the recognition of the three control characters at the RDOS OR DOS system terminal.
3. You can cancel the DISABLE command with an ENABLE command.
4. A DISABLE command executed while BASIC is running in the foreground inhibits use of CTRL-F on the background terminal.

Example

```
* DISABLE
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

DISK

Determines the number of 256-word blocks still available in the partition in which your directory resides.

Format

DISK

Remarks

In AOS and AOS/VS, this command works only if the current directory is a control point directory (CPD).

Example

```
* DISK

USED: 332
LEFT: 193
```

Of 525 blocks, 193 are still available for use.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

ECHO

Enables character display on input.

Format

ECHO

Remarks

1. ECHO is enabled by default.
2. ECHO is equivalent to the command CHAR "ON","EB0" under AOS and AOS/VS, and CHAR "OFF","NOE" under RDOS/DOS.
3. ECHO is disabled by NOECHO.

Example

```
* 10 ECHO
* 20 INPUT "→",A
* 30 END
* RUN
→ 6
END AT 0030
*
```

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

ENABLE

Cancels a DISABLE command.

Format

ENABLE

Remarks

1. ENABLE must be executed from the master terminal. In a single-user system, the master terminal and user terminal are the same. In multiuser environments, programs using ENABLE as a statement do not work when run from terminals other than the master terminal.
2. ENABLE restores recognition of the CTRL-A, CTRL-C, and CTRL-F (background) RDOS or DOS terminal breaks.

Example

```
* ENABLE
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

END

Terminates execution of the program and returns to interactive mode.

Format

END

Remarks

1. Data General's Extended BASIC does not require an END statement to declare the physical end of a program. If control passes through the last executable statement of the program and if that statement does not change the flow of control (as with a GOTO or GOSUB statement), the program transfers control to interactive mode. END is included for compatibility with BASIC programs written for other systems.
2. Multiple END statements may appear in the same program. When BASIC encounters an END statement, it terminates execution of the program and displays a prompt.
3. If BASIC executes an END statement before reaching a FOR/NEXT terminating condition, it prints an error message.

Example

```
* 20 PRINT "PROGRAM DONE"
* 30 GOTO 60
   .
   .
   .
* 60 END
* RUN
PROGRAM DONE

END AT 0060
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

ENTER

Transfers and merges BASIC source statement lines from a device or disk file into your current program storage area.

Format

ENTER { "filename" / svar }

Arguments

filename — A device or disk file, expressed as a string literal

svar — A string variable representing a filename

Remarks

1. If the filename is not found in the current directory, BASIC searches the BASIC.DR library directory (RDOS, DOS) or the searchlist (AOS, AOS/VS) for it.
2. When a statement line from an entered filename has the same statement number as a line in the current program, the entered statement replaces the current program statement.
3. ENTER does not restore data statements even though some may exist in the entered program. If you desire a RESTORE, you must code it explicitly.
4. Although ENTER can be used as a statement, it is a questionable practice and should be done cautiously.

Example

```
* NEW
* ENTER "TEST1.SR"
* ENTER "TEST2.SR"
* LIST "FINAL.SR"
```

The system clears your storage area and enters and merges source programs TEST1.SR and TEST2.SR. It then lists the resultant program to your directory as FINAL.SR.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	
F	✓

EOF(X)

Detects the end of file when transferring data from a file.

Format

EOF (file)

Argument

file A numeric expression that evaluates to the number of a file opened for reading in mode 0 or 3.

Remarks

1. The EOF function returns an integer indicating whether or not the last READ FILE, LREAD, or INPUT FILE from the file included an end-of-file delimiter. If an end of file is detected, the function returns a value of 1; otherwise it returns 0.
2. When you use the EOF function in conjunction with the IF-THEN statement, you can make a conditional transfer if an end of file is detected.
3. Testing for an end of file should occur immediately after the INPUT FILE, LREAD, or READ FILE statement. If the EOF value is 1, ignore the values assigned to the variables in the INPUT FILE, READ FILE, or LREAD file. This concept also applies to matrix file I/0 statements.
4. BASIC sets the EOF function if you try to read a record with a higher number than any already written to the file. If the next READ FILE is successful, BASIC resets EOF. However, if you attempt to read consecutively a record with a number higher than any in the file, an error occurs. You may continue processing after an EOF with the RESET FILE statement.
5. All physical records written to magnetic tape and cassettes are of a fixed length and padded with nulls. Therefore, the EOF function is not necessarily set at the logical end of file; the function is set at the logical end of file only when it coincides with the physical end of file.

Example

```
* LIST
0100 OPEN FILE(1,3),"$PTR"
0110 READ FILE(1),A,B,C,D,E
0120 IF EOF(1) THEN GOTO 0200
0130 PRINT A,B,C,D,E
0140 GOTO 0110
0200 CLOSE FILE(1)
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

ERASE

Removes statements from a program.

Format

ERASE n1, n2

Arguments

n1, n2 Line numbers in a program

Remarks

1. ERASE simplifies editing by enabling you to delete more than one line at a time. Typically, use the ERASE statement in a program to clear a range of statements for replacement with the ENTER command, or to remove initialization code which is not needed during execution.
2. Both line number arguments are necessary.
3. ERASE removes lines n1 through n2, inclusively, from your program, regardless of which of the two line numbers is the larger. Thus, ERASE 110,250 and ERASE 250,110 both erase lines 110 through 250.
4. If no lines exist in your program within the range n1 to n2, BASIC outputs an error message to your terminal. If n1 and/or n2 does not exist but there are lines between n1 and n2, those lines are erased.
5. The RENUMBER command does not renumber the arguments to the ERASE statement, since they most often refer to fixed statement numbers in an external program file. If you renumber a program, make certain you change any ERASE statements in the program to agree with the new line numbers.

Example

Delete lines 1500 through 1900:

```
* 10 ERASE 1500, 1900
```

Delete line 200:

```
* 20 ERASE 200, 200
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

ESC

Enables the ESC key to produce an interrupt.

Format

ESC

Remarks

1. ESC is enabled by default.
2. ESC is equivalent to the command CHAR "ON","ESC".
3. ESC is disabled by NOESC.

Example

```
* 0005 NOESC
* 0010 FOR I = 1 TO 100
* 0020       PRINT I
* 0030       IF I = 50 THEN ESC
* 0040 NEXT I
* 0050 END
* RUN
 1
 2
 3
   .
   .
ESC
   .
   .
 50
   .
   .
ESC
STOP AT 0020
*
```

In this example, if I < 50 and the user presses ESC, an interrupt does not occur. However, when I > 50, pressing the ESC key causes an interrupt.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	
F	✓

EXP(X)

Calculates the value of *e* (2.71828) to the power of a numeric expression.

Format

EXP(expr)

Argument

expr A numeric expression from -178 through 175

Example

```
* LIST
0010 REM- CALCULATE VALUE OF E ↑ 1.5
0020 PRINT EXP(1.5)
* RUN
4.4816891

END AT 0020
*
```

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

FALL

Forces the transmission of a message to all active users.

Format

FALL message

Argument

message The text of the message

Remarks

1. FALL must be executed from the master terminal. In a single-user system, the master terminal and user terminal are the same. In multiuser environments, programs using FALL as a statement do not work when run from terminals other than the master terminal.
2. FALL overrides use of the CHAR command to disable message reception.
3. FALL prints the the message at each user's terminal in the following format, where OPER is the ID associated with the master terminal:

   FROM OPER: message
4. Message length is limited to one line per FALL command.
5. The message need not be enclosed within quotation marks.

Example

```
*FALL ***SYSTEM GOING DOWN IN 5 MINUTES***
*FALL ***PLEASE LOG OFF WITHIN 5 MINUTES***
*
```

AOS, AOS/VS	✓
RDOS, DOS	

$	✓
C	✓
F	

FILE

Prints all the filenames in your directory that match the template.

Format

{FILE / #} *["template"]*

Arguments

A synonym for the keyword FILE

template Any combination of up to 15 valid characters, including asterisk (*), dash (-), and plus sign (+), in accordance with AOS and AOS/VS template rules. You must express the template as a string literal or string variable.

Remarks

1. If you omit the template argument, BASIC prints a list of all files in the directory.
2. The following information follows each filename printed:
   - Type of file
   - Date last modified
   - Time last modified
   - Size of the file, in bytes

Examples

```
* FILE "-.-"
PAGE2.2 UDF 11-MAR-83 11:17:48 78
* FILE "-.SR"
* FILE "-.2"
PAGE2.2 UDF 11-MAR-83 11:17:48 78
* A$="-.2"
* FILE A$
PAGE2.2 UDF 11-MAR-83 11:17:48 78
*
```

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

FILE

Prints all filenames in your directory.

Format

{FILE / #}

Remarks

1. The number sign (#) is a synonym for the keyword FILE.
2. BASIC prints one filename per print zone.

Example

```
* FILE
157.        134.        GOSUB1.SR
STOP.SR     121.        NEW.
ON.ES       READ:SR     116.
TIME.       FOR1.SR     FOR2.SR
MORSE.      110.SR      FOR4.SR
113.        TAB.SR      132A.
COM.CM      CON.        110.
TAB.        SUBSTRINGS. CONCAT.
115.        GOTO.       111A.
PAGE.       HELLO.SV    PRINT1.SR
PRINT2.SR   109B.       PRINT3.SR
PRINT4.SR   92.         INPUT2.SR
107.        117.        IF3.
*
```

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

FMSG

Forces the transmission of a message to a specific user.

Format

FMSG userID message

Arguments

userID A four-character user account identification

message The text of the message

Remarks

1. FMSG must be executed from the master terminal. In a single-user system, the master terminal and user terminal are the same. In multiuser environments, programs using FMSG as a statement do not work when run from terminals other than the master terminal.
2. FMSG overrides the NOMSG command and the CHAR command's NRM characteristic, which disables message reception.
3. When you use FMSG, the message is printed at the user's terminal in the following format, where OPER is the ID associated with the master terminal:

   FROM OPER: message
4. Message length is limited to one line per FMSG command.
5. The message need not be enclosed within quotation marks.
6. To enable message reception, use CHAR "OFF","NRM".

Example

```
*FMSG JACK PLEASE LOG OFF.
*
```

On Jack's terminal:

FROM OPER: PLEASE LOG OFF.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	
F	

FOR and NEXT

Execute a block of statements a specified number of times.

Format

FOR control var=expr1 TO expr2 *[STEP expr3]*

.
.
.

(Block of statements)

.
.
.

NEXT control var

Arguments

control var The control variable. A nonsubscripted numeric variable

expr1 A numeric expression that defines the initial (first) value of the control variable

expr2 A numeric expression that defines the limiting value of the control variable

expr3 An optional numeric expression that defines the increment added to the control variable each time the loop is executed. The default increment is 1.

Block of statements
Any statements, which may also contain FOR-NEXT loops

Remarks

General

1. A program loop begins with a FOR statement that provides the specifications for repetition, a block of statements that BASIC executes during each repetition of the program loop, and a NEXT statement that denotes the end of the loop.
2. The initial, limiting, and incremental values for the control variable determine the number of times the statements contained in a FOR-NEXT loop are to be executed. The system repeats the loop until the value of the control variable meets the termination condition.

Rules

1. Every FOR statement must have a matching NEXT statement; otherwise an error condition occurs, and BASIC prints an error message.
2. The control variable must not be subscripted.
3. Expressions expr1, expr2, and expr3 may have positive or negative values; expr3 must not be 0.
4. If you omit the STEP argument, expr3 is assumed to be 1.
5. The termination condition for a FOR-NEXT loop depends on the values of expr1 and expr3. The loop terminates if (a) expr3 is positive and the next value of the control variable is greater than expr2, or (b) expr3 is negative and the next value of the control variable is less than expr2.
6. If the value of expr1 (the initial value) meets the termination condition, the loop is not performed (see third example below).
7. When the termination condition is met, the loop is exited; the control variable equals the first value not used in the loop.
8. You may branch in and out of a FOR-NEXT loop; but if you enter a loop at a point other than a FOR statement and the program then encounters a NEXT statement, an error occurs.
9. If the program begins a FOR-NEXT loop and before the loop is completed (a NEXT is encountered) the program encounters an END or the last statement in the program, BASIC displays an error message:

   ERROR 21 at xxxx -- FOR-noNEXT

   where xxxx is the program statement number. You must put any END statement after the completion of the loop.

Steps in Loop Operation

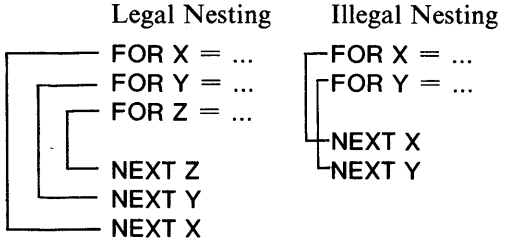
1. The system evaluates the expressions expr1, expr2, and expr3. If you omit expr3, it is assumed to be 1.
2. The control variable is set to the value of expr1.
3. If expr3 is positive and the control variable is greater than expr2, then the termination condition is satisfied; control passes to the statement following the corresponding NEXT statement. The value of the control variable then equals the first value not used in the loop; i.e., control variable + expr3.

   If expr3 is negative and the control variable is less than expr2, then the termination condition is satisfied; control passes to the statement following the corresponding NEXT statement. The value of the control variable then equals the first value not used in the loop; i.e., control variable + expr3.

   Otherwise, the system performs the following steps:
4. BASIC executes the statements in the FOR-NEXT block.
5. When the corresponding NEXT statement is executed, the control variable is set to the value of control variable + expr3.
6. Control passes to FOR statement. Repeat step 3.

Nesting Loops

1. In AOS and AOS/VS environments, the maximum number of FOR-NEXT loops is 20. In RDOS/DOS, you can nest FOR-NEXT loops to the depth specified by the system manager.
2. The FOR statement and its terminating NEXT statement must be completely contained within the loop in which they are nested.



FOR and NEXT (continued)

Examples

```
* LIST
0010 FOR I=1 TO 9
0020 NEXT I
0030 PRINT I
* RUN
10
```

(Control variable I equals first value not used in the loop.)

```
END AT 0030
*

* LIST
0040 FOR J=1 TO 9 STEP 3
0050 NEXT J
0060 PRINT J
* RUN
10
```

(Final value of J when terminating value, expr2, was exceeded.)

```
END AT 0060
*

* LIST
0010 FOR I=1 TO 3 STEP -1
0020 PRINT "SHOULD NOT ENTER HERE"
0030 NEXT I
0040 PRINT I
* RUN
1

END AT 0040
*
```

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

FREE

Interrupts execution of the program that is being processed for the specified user.

Format

```
FREE userID
```

Argument

userID A four-character user account identification

Remarks

1. FREE must be executed from the master terminal. In a single-user system, the master terminal and user terminal are the same. In multiuser environments, programs using FREE as a statement do not work when run from terminals other than the master terminal.
2. Normally the user can interrupt program execution by pressing the ESC key. However, if a program is locked in an inescapable loop, the user may need assistance from the system operator.
3. When the system operator executes the FREE command, the message UNLOCKED BY OPERATOR and a prompt are output to the affected user's terminal.

Example

This program, run by user JOHN, creates an inescapable loop:

```
* 10 ON ESC GOTO 20
* 20 ON ERR GOTO 30
* 30 X=1/0
* 40 END
* RUN
```

The command at the system terminal is:

```
* FREE JOHN
*
```

The following message appears at JOHN's terminal:

```
UNLOCKED BY OPERATOR
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

GDIR

Prints the name of your directory.

Format

GDIR

Remarks

1. This command is useful at the RDOS system terminal, particularly when you use the DIR command frequently to change directories.
2. RDOS users at terminals other than the system terminal may also use this command, but its usefulness is limited since the users are restricted to their current directories.
3. The knowledge of your directory name is useful to other programmers who wish to create a link to your files that are sharable (see CHATR, attribute O).
4. You can write files to your own directory, read files from the library directory BASIC.DR, and read files from other directories that have the sharable attribute (O).

Example

```
* GDIR
JOE
```

All of your file references are to directory JOE.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	
F	

GOSUB and RETURN

GOSUB directs program control to the first statement of a subroutine. RETURN exits the subroutine and returns program control to the next statement following the GOSUB statement.

Format

GOSUB line no.

RETURN

Argument

line no. A program line number

Remarks

1. A subroutine is a group of program statements that the system enters via the GOSUB statement and exits via the RETURN statement. Instead of repeating the statements each time they are required, you write the statements into the program only once and access them by GOSUB statements. The RETURN statement returns control to the statement following the last executed GOSUB statement. In this manner, the program continues at the appropriate place after the system has executed the subroutine.
2. You must always enter a subroutine by using a GOSUB statement. Otherwise, the system prints the RETURN-NO GOSUB error message when it executes the RETURN statement.
3. You may use more than one RETURN statement in a subroutine if program logic requires the subroutine to terminate at one of a number of different places.
4. Although a subroutine may appear anywhere in a program, it is good practice to place the subroutine distinctly separate from the main program. To prevent inadvertent entry to the subroutine by other than a GOSUB statement, the subroutine should be preceded by a STOP statement or GOTO statement that directs control to a line number following the subroutine.
5. Nesting occurs when a subroutine is called during the execution of a subroutine. Upon execution of the first RETURN statement, control passes to the statement immediately following the last executed GOSUB statement. The next RETURN statement causes control to pass to the next to last executed GOSUB statement, and so on.

GOSUB (continued)

6. In AOS and AOS/VS environments, subroutines may be nested to a depth of 20. In RDOS and DOS, subroutines may be nested to a depth specified by the system manager.

Examples

```
* LIST
0010 LET A=6
0020 GOSUB 0100
0030 LET A=10
0040 GOSUB 0100
0050 STOP
0100 FOR I=1 TO A STEP 2
0110      PRINT I;
0120 NEXT I
0130 PRINT
0140 RETURN
* RUN
1     3     5
1     3     5     7     9

STOP AT 0050
*

* LIST
0010 GOSUB 0040
0020 PRINT "EXAMPLE";
0030 STOP
0040 PRINT "NEST";
0050 GOSUB 0080
0060 PRINT "INE ";
0070 RETURN
0080 PRINT "ED ";
0090 GOSUB 0120
0100 PRINT "ROUT";
0110 RETURN
0120 PRINT " SUB";
0130 RETURN
* RUN
NESTED SUBROUTINE EXAMPLE

STOP AT 0030
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

GOTO

Unconditionally transfers control to the statement with the specified line number.

Format

GOTO line no.

Arguments

line no. A program statement line number

Remarks

1. If control passes to an executable statement, BASIC executes that statement and those following.
2. If control passes to a nonexecutable statement (e.g., DATA), program execution continues at the first executable statement that follows the nonexecutable statement.
3. If the specified line number is not in the program, an error occurs.

Examples

```
* LIST
0010 READ X
0020 PRINT X
0030 GOTO 0010
0040 DATA 1,2,3,4,5
0050 DATA 20,21,23
0060 END
* RUN
1
2
3
4
5
20
21
23
ERROR 15 AT 0010 - END OF DATA
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

GPOS FILE

Determines the current file pointer position in an open file.

Format

GPOS {FILE / #} (file), var

Arguments

A synonym for the keyword FILE

file A numeric expression that evaluates to a file number previously associated with an OPEN FILE statement.

var The name of the variable to which BASIC is to assign the current byte position value.

Remarks

Note that this statement returns the value, in var, of the current byte position of the file pointer; this does not necessarily coincide with the beginning of a record position.

Example

```
* 10 GPOS FILE (2), B1
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

HELP

Displays information about each BASIC statement and command.

Format

```
HELP "verb"
```

Argument

verb The name of a statement, command, or function expressed as a string literal or string variable

Remarks

To display a list of all statements, commands, and functions that can be the verb argument, use HELP either without an argument or with the argument "HELP".

Example

```
* HELP "DATA"
DATA  TO PROVIDE VALUES FOR VARIABLES
      SPECIFIED IN A [MAT] READ STATE-
      MENT...
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

IF-THEN

Executes a statement on the basis of an expression or a relational expression's truth or falsity.

Format

IF { rel-expr / expr } { [*THEN*] statement / THEN line no. }

Arguments

rel-expr	A relational expression as defined in chapter 2
expr	A numeric expression
statement	Any BASIC statement except DATA, DEF, END, FOR, NEXT, and REM.
line no.	A program line number

Remarks

1. If BASIC finds the relational expression (rel-expr) to be true, it executes the specified statement or transfers control to the specified line number. If the expression is false, program execution continues at the next statement after the IF-THEN statement.
2. You can use a numeric expression (expr) instead of a relational expression (rel-expr). The numeric expression is considered false if it has a value of 0, and true if it has a nonzero value.
3. Since the internal representation of noninteger numbers (see chapter 2) may not be exact (e.g., 0.2 cannot be exactly represented), it is advisable to test for a range of values when testing for a noninteger. For example, if the result of computation A was to be 1.0, then a reliable test for 1 is:

   ```
   IF ABS (A-1.0)<1.0E-6 THEN...
   ```

   If this test succeeds, then A is equal to 1 within 1 part in 10↑6. This is approximately the accuracy of single-precision floating-point calculations.
4. Lines 5, 10, and 20 in the following example are equivalent variations of the IF-THEN statement:

   ```
   * 05 IF A=B THEN 100
   * 10 IF A=B THEN GOTO 100
   * 20 IF A=B GOTO 100
   * 30 IF A-B < = 5 THEN C=0
   * 40 IF A*B < 50 THEN GOSUB 300
   * 50 IF A B > 100 GOSUB 400
   ```

Examples

```
* LIST
0005 REM - START
0010 LET N=10
0020 INPUT "X=",X
0030 IF X THEN GOTO 0050
0040 GOTO 0100
0050 IF X > =N THEN GOTO 0080
0060 PRINT X,"X IS LESS THAN 10"
0070 GOTO 0020
0080 PRINT X,"X IS GREATER OR EQUAL TO 10"
0090 GOTO 0020
0100 PRINT X,"X=0"
0110 END
* RUN
X = 5
5      X IS LESS THAN 10
X = 7
7      X IS LESS THAN 10
X = 12
12     X IS GREATER OR EQUAL TO 10
X = 10
10     X IS GREATER OR EQUAL TO 10
X = 0
0      X=0

END AT 0110
*
```

Note the nested IF statement in the following example:

```
* LIST
0010 LET X=5
0030 IF X=5 THEN IF A$="C" THEN PRINT "C"
0040 END
* RUN
C

END AT 0040
*
```

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

INIT

Initializes a directory or a device and thereby permits access to its files.

Format

INIT name

Argument

name The name of a directory or a device to be initialized

Remarks

1. INIT must be executed from the master terminal. In a single-user system, the master terminal and user terminal are the same. In multiuser environments, programs using INIT as a statement do not work when run from terminals other than the master terminal.
2. In BASIC, you will typically use INIT to initialize reserved filename devices, such as magnetic tape or cassette units. You must initialize these devices before use and use RELEASE on them afterwards.

Example

```
* INIT "MT0"
* INIT "CT3"
* INIT B$
* INIT "DC3"
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

INPUT

Requests data from your terminal and assigns the values you supply to a list of variables.

Format

INPUT *["str lit",]* {svar / var} [{*,svar* / *,var*}] ... *[;]*

Arguments

var, svar Numeric and string variables separated by commas

str lit A string literal that serves as a message or prompt

Remarks

1. You can use the INPUT statement to enter numeric data, string data, or both to a program.
2. When BASIC executes an INPUT statement, a question mark is output as a prompt unless INPUT contains the string literal option; in this case the string literal is the prompt.
3. Respond to the prompt by typing a list of data, separating each datum from the next by a comma or a carriage return. Terminate the list with a carriage return.
4. If you terminate the data list without supplying a value for each variable in the INPUT statement, BASIC outputs a question-mark prompt, indicating that you must supply more data.
5. Variables in the INPUT statement list may be subscripted array elements, scalars, or strings.
6. The data input at the prompt must be the same type (numeric or string) as the variable in the INPUT statement list for which the data is being supplied. If the data you input from the terminal does not match the variable's type, the system outputs \? to the terminal for the data in error.
7. If you end an INPUT statement's variable list with a semicolon, the cursor remains at the position following the last input item (AOS and RDOS). Otherwise, the system outputs a carriage return-line feed.

INPUT (continued)

8. Numeric variables may include digits, plus and minus signs, decimal points, and the letter E (exponential notation).
9. If you use commas to delimit the data list and you supply more items than there are variables in the INPUT list, an error condition occurs. The system assigns the values you supplied to the variables in the list and ignores the excess.

Examples

```
* LIST
0005 INPUT A,B,C,D,E
0010 PRINT A+B,C+D,D+E
* RUN
? 1,2,3,4,5
3   7   9

END AT 0010
*

* LIST
0010 INPUT "A,B,C,D,E= ",A,B,C,D,E
0020 PRINT A+B,C+D,D+E
* RUN
A,B,C,D,E= 1,2 ? 3,4,5
3   7   9

END AT 0020
*

* LIST
0010 INPUT A,B,C;
0020 PRINT " NO RETURN"
* RUN
? A \ ? 1,2,3 NO RETURN

END AT 0020
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

INPUT FILE

Reads data in ASCII format from a disk file or device.

Format

```
INPUT {FILE} {(file)          } , {var } , [ {var } ]...
      {#   } {(file,record)  }   {svar}   [ {svar} ]
```

Arguments

#	A synonym for the keyword FILE
file	A numeric expression that evaluates to the number of a file opened for sequential access or for random access
record	A numeric expression that evaluates to the number of a record in a file opened for random access
var, svar	Numeric variables and string variables whose values are read from a file

Remarks

1. The variable type in the INPUT FILE variable list must correspond to the data type of the corresponding data item being read from the file.
2. Format the data as in the INPUT statement, with commas separating data items and a carriage return at the end of a variable list.
3. You can use the EOF function to detect an end of file in the file that is being read.
4. The first record number in a random-access file is 0.

Example

```
* 40 OPEN FILE (1,3), "$PTR"
* 70 INPUT FILE (1), Z,Y,X,A$,B$
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	
F	✓

INT(X)

Returns the value of the largest integer not greater than the specified expression.

Format

INT(expr)

Argument

expr A numeric expression

Remarks

The INT function truncates numbers to return integers, but print formatting rounds numbers for output. There may appear to be discrepancies, but the internal number representation does not change.

Examples

Line 30 of the following example demonstrates a technique for rounding real numbers to the nearest integer:

```
* LIST
0010 PRINT "INT(15.8)= ";INT(15.8)
0020 PRINT "INT(-15.8)= ";INT(-15.8)
0030 PRINT "INT(15.8+.5)= ";INT(15.8+.5)
* RUN
INT(15.8)= 15
INT(-15.8)= -16
INT(15.8+.5)= 16

END AT 0030
*
```

The following example illustrates the rounding of PRINT and the truncation of INT that occurs for numbers that do not have an exact internal binary representation:

```
* LIST
0010 INPUT "→ ",X
0020 Y=X*10000
0030 Z=INT(Y)
0040 W=Z/10000
0050 PRINT X,Y,Z,W
0060 PRINT USING "###.###",W
0070 STOP
* RUN
→.4
.4       4000       3999       .3999
 0.400

STOP AT 0070
*
```

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

KILL

Forces a specific user off the system.

Format

KILL userID

Argument

userID A four-character user account identification

Remarks

1. KILL must be executed from the master terminal. In a single-user system, the master terminal and user terminal are the same. In multiuser environments, programs using KILL as a statement do not work when run from terminals other than the master terminal.
2. When KILL forces a user off the system, accounting information is displayed on the user's terminal in the following form:

   LOGGED OFF BY OPERATOR
   NAME mm/dd/yy hh:mm SIGN OFF,zz
   NAME mm/dd/yy hh:mm CPU USED,qq
   NAME mm/dd/yy hh:mm I/O USED,rr,ss

   Note: zz, qq, rr, and *ss* are defined in chapter 1.

Example

```
* KILL JACK
```

This information is displayed on JACK's terminal:

```
LOGGED OFF BY OPERATOR
JACK 11/15/83 14:20 SIGN OFF 03
JACK 11/15/83 14:20 CPU USED 9
JACK 11/15/83 14:20 I/O USED 3,12
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	
F	✓

LEN(X$)

Returns a value equal to the number of characters currently assigned to the specified string variable.

Format

LEN(svar)

Argument

svar A string variable

Example

```
* LIST
0005 DIM A$(80),B1$(80)
0010 INPUT A$,B1$
0020 LET B=LEN(A$)
0040 IF B > LEN (B1$) THEN GOTO 0060
0050 GOTO 0100
0060 PRINT "LENGTH OF A$=";LEN(A$)
0070 PRINT "LENGTH OF B1$=";LEN(B1$)
0080 PRINT "A$ > B1$"
0090 GOTO 0110
0100 PRINT "B1$ > A$"
0110 END
* RUN
? CHEESE ? CAKE
LENGTH OF A$= 6
LENGTH OF B1$= 4
A$ > B1$>>)
END AT 0110
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

LET

Evaluates an expression and assigns the resultant value to the specified numeric or string variable.

Format

[LET] { var / svar } = expr

Arguments

var, svar Numeric and string variables

expr An arithmetic or string expression

Remarks

1. The keyword LET is optional.
2. The variable argument may be subscripted.
3. You can assign string expressions to string variables.
4. You can assign more than one variable the same value by using a multiple LET assignment, valid as either a statement or a command. See line 40 in the examples.

Examples

```
* 10 LET A=A+1
```
Variable A is assigned a value one greater than it was before.

```
* 20 A(2,1) = B↑2+10
```
The element in row 2, column 1 of array A is assigned the value of expression B↑2+10.

```
* 30 A$=B$,C$
```
A$ is assigned the concatenated value of B$ and C$.

```
* 40 LET A, B, C=2
```
A, B, and C each equal 2.

```
* 50 LET M(2),A=3
```
M(2) and A both equal 3.

```
* 60 IF A=3 THEN LET D,F=1
```
This is a conditional multiple LET: if A equals 3, then D and F are set to 1.

```
* 70 LET A$=B$+C$
```
String arithmetic

```
* 80 LET A$,B$="ABC"
```
Both strings A$ and B$ equal ABC.

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

LEVEL

Monitors the priority constant for any user.

Format

LEVEL *[userID] [n]*

Arguments

userID A four-character user account identification

n A priority level constant in the range 0 =< n <= 50

Remarks

1. The default value of *n* is 25.
2. LEVEL can either raise or lower the priority level of any user's tasks. The lower the value, the higher the priority. All users may use LEVEL, but only the system operator can set priority levels.
3. For equal priority levels, BASIC handles tasks on a first-in, first-out basis.

Examples

```
* LEVEL
```

The system prints the current value of the priority level constant for the user. Any user can issue this form of the command to see that user's value of *n*.

```
* LEVEL JEFF
```

This prints the value for the priority level constant for user JEFF on the master terminal. Any user can issue this form of the command to determine other users' priority levels.

```
* LEVEL JEFF 10
```

This sets the value for the priority level constant for user JEFF. Only the system operator can issue this form of the command.

```
* LEVEL OPER 30
```

This sets the system console to a priority lower than the default.

AOS, AOS/VS	✓
RDOS, DOS	

S	✓
C	✓
F	

LIBRARY

Prints in the specified directory the filenames that match the template.

Format

LIBRARY [{ "directory", "template" / "directory" / "template" }]

Arguments

directory Any legal directory pathname starting from the root (:), expressed as a string literal or string variable

template Any combination of up to 15 valid characters, including asterisk (*), dash (-), and plus sign (+), in accordance with AOS and AOS/VS template rules. The template must be expressed as a string literal or string variable.

Remarks

1. If you omit both the directory and template options, BASIC prints a list of all files in the BASIC library directory (:BASIC).
2. If you omit the directory option and specify a template, BASIC prints a list of all files in the library directory that matches the template.
3. If you omit the template option and specify a directory, BASIC prints a list of all files in that directory.
4. BASIC provides the following information with each filename printed:
   - Type of file
   - Date last modified
   - Time last modified
   - Size of the file, in bytes
5. When completed, LIBRARY returns the user to the initial working directory.

LIBRARY (continued)

Examples

```
* DIM A$(30)
* A$=":UDD:XBASIC"
* LIBRARY A$,"-TAPE-.CLI"
ROOMTAPE.CLI      TXT 11-APR-83 07:29:06 263
RELEASETAPE0100.CLI
                  TXT 11-APR-83 07:28:36 368
ROOMTAPE0100.CLI TXT 11-APR-83 07:29:44 407
*
* A$=":UDD:XBASIC:DUDLEY"
* LIBRARY A$
HOBURG            UDF 05-APR-83 08:51:42 106543
MARATHON.BASIC  UDF 19-FEB-83 15:14:36   1943
```

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

LIBRARY

Prints all filenames in the library directory.

Format

LIBRARY

Remarks

One filename is printed per print zone.

Examples

```
* LIBRARY
A1.             SHOT.SR         SQRT.SR
BACKGAMMON      SUPERGUESS      STOCKS.SR
CASINO.SR       SWAP.SV         SNOOP.
COMPILER.SR     FCOM.CM         BLACK-
BANK.SR         FOOTBALL.SR     JACK.SR
KILLER.MS       K2.             BILL-
SNOOPY.SR       GUESS.SR        BOARD.SR
TEST1.          HORSERACE.SR    MAT.SR
BATNUM.SR       HEMAN.SR        QUEEN.SR
FISCAL.SR       HELLO.SR        LUNAR.SR
FISCAL.BT                       SHOT1.SR
*                               HELLO.SV
```

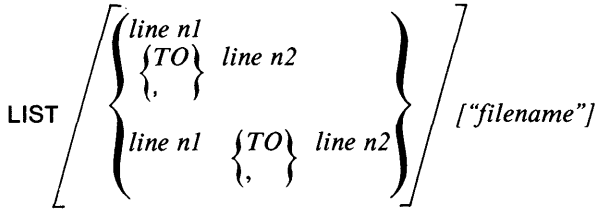
AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	✓
F	

LIST

Outputs part or all of your current program in ASCII to a disk file, device, by filename, or your terminal.

Format



Arguments

line n1 The line number of the first statement to be listed

line n2 The line number of the last statement to be listed

filename A disk file or device expressed as a string literal

Remarks

1. You can use LIST in the following four ways:

LIST	List the entire program from the lowest numbered statement.
LIST n1	List only the single statement at line number n1.
LIST {TO, } n2	List from the lowest numbered line through line number n2.
LIST n1 {TO, } n2	List from line number n1 through line number n2.

2. When you include the filename argument, LIST writes the specified lines to the disk file or device in ASCII format.
3. If the filename is a disk file that already exists in your directory, BASIC prints the message:

   TYPE CR TO DELETE OLD: (RDOS/DOS)

   TYPE NL TO DELETE OLD: (AOS,AOS/VS)

   This message lets you confirm whether or not the existing file is to be deleted and replaced by the file, with the specified lines, named in the LIST command. If you press CR/NEW LINE, BASIC accepts the replacement. If you type anything preceding CR/NEW LINE, BASIC cancels your LIST command.
4. You can read the file created by the LIST command back into the program storage area with the ENTER or NEW commands.

Examples

List your current program on your terminal:

```
* LIST
```

Output your current program to the line printer:

```
* LIST "$LPT"
```

List line number 20 on your terminal:

```
* LIST 20
```

List line numbers 700 through 9999 at the terminal:

```
* LIST 700,9999
```

Output your current program, in ASCII, to your directory with the filename TEST.SR:

```
* LIST "TEST.SR"
```

TEST.SR replaces any previous file with that name, provided you press the appropriate line terminator at the confirmation prompt.

List line numbers 100 through 200 to disk file TEMP:

```
* LIST 100, 200 "TEMP"
```

TEMP replaces any previous file of that name, provided you press the appropriate line terminator at the confirmation prompt.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	✓
F	

LOAD

Loads a program into your program storage area that was previously saved in core image format.

Format

LOAD "filename"

Argument

filename The name of a core image file created by a previous SAVE command

Remarks

1. The LOAD command executes an implicit NEW command (clearing the storage area) and then reads the specified file into memory.
2. The file may be on disk or it may also be on a binary input device such as the paper tape reader.
3. If the file is not found, BASIC searches the library directory (RDOS and DOS) or searchlist (AOS and AOS/VS) for the filename.
4. After you have loaded a file, you can list, modify, or run it.

Examples

```
* LOAD "$PTR"
* LOAD "MATH3"
* LOAD "MT0:1"
```

AOS, AOS/VS	✓
RDOS, DOS	

S	✓
C	✓
F	

LOCK

Gives exclusive access to a record in a file.

Format

LOCK iden,"filename",start, record-size *[,time]*

Arguments

iden A numeric expression used to identify a specific lock or group of locks in a program

filename A disk file in your directory, expressed as a string literal or string variable

start A numeric expression that specifies the starting byte number of the record to be locked

record-size A numeric expression that specifies the number of bytes to be placed in the lock area

time An optional numeric variable that specifies a wait time, in seconds, if the record to be locked is potentially locked by somebody else

Remarks

1. The lock identifier does not have to be unique for each lock area of the program.
2. Only the first ten characters of the filename, including extensions, are uniquely associated with each particular lock.
3. The record-size should be the actual number of bytes needed to be locked in the file. The physical record size with which a file is opened has no effect on the lock mechanism. The user of LOCK should be careful with the size and time of locking an area, as no one else can access this area until it is unlocked.
4. When you omit the optional time variable, an infinite wait time is defaulted. BASIC suspends the program until it can satisfy the requested lock.
5. BASIC places the return status from LOCK in the time variable. The possible returns are:
   a. The specified file area has been successfully locked. BASIC returns a value of -1.
   b. The lock request cannot be granted even after BASIC suspends the program for the specified time limit. BASIC returns the time variable with a value of 7.

c. The program has already locked the specified file area. The system returns the time variable with a value of 6. If the program specifies no time variable, "ERROR 64 - Attempt to lock same record twice" is displayed.

You should test the time variable after each return from LOCK to determine whether the lock was successful.

6. A keyboard interrupt stops a suspended program waiting for a lock request to be satisfied. The time variable then indicates if the lock was successful before the interrupt (-1) or if the system did not grant it (7).
7. Whenever BASIC terminates, it unlocks all areas locked by the user. The NEW command also unlocks all current locks.
8. LOCK does not physically prevent access to locked areas. All programs must follow the convention of requesting a lock and testing the time variable before accessing a record; this ensures reliable sharing of file areas under the locking facility.

Examples

```
* 10 A=10
```
(Set a wait time of 10 seconds.)

```
* 20 LOCK 1,"INVENTORY",0,128,A
```
(Lock a 128-byte record.)

```
* 30 IF A=-1 THEN GOTO 60
```
(If BASIC successfully locks the record, then goto to 0060.)

```
* 40 IF A=7 THEN GOTO 1000
```
(If the record is already in use, go to 1000 and do another routine.)

```
* 50 IF A=6 THEN STOP
```
(If this user has already locked the record, then stop. See 5 above.)

```
* 60 UNLOCK 1
```
(Release the lock, and allow other users access to the file area.)

```
* LOCK 2,A$,A,B,C
```

```
* LOCK 3,"FILE",I*A,A
```
(I is the current record number and A is the record size.)

```
* UNLOCK
```
(Unlock all earlier locks.)

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	
F	✓

LOG(X)

Calculates the natural logarithm of an expression.

Format

LOG (expr)

Argument

expr A numeric expression with a value greater than 0

Example

```
* LIST
0010 REM- CALCULATE THE LOG OF 959
0020 PRINT LOG(959)
* RUN
 6.8658911

END AT 0020
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

LREAD

Reads a string, terminated by either a null, form feed, or carriage return (NEW LINE) from the terminal.

Format

LREAD *["str lit",]* svar *[,svar1] [;]*

Arguments

str lit A string literal supplying a message or prompt

svar A string variable that is assigned the value of the string read from your terminal

svar1 A string variable that is assigned the value of the delimiter for the string read. Valid delimiters are null, form feed, carriage return (NEW LINE) and function keys under AOS and AOS/VS if the characteristic FKT is on.

Remarks

1. LREAD is specifically intended to process text where commas and quotes are input characters, not delimiters.
2. If LREAD contains the string literal option, the string literal appears an initial prompt; otherwise a question mark appears as the prompt.
3. The maximum string length allowed for svar is 133, which includes the delimiter. If a response to the LREAD prompt is longer than 133 characters, the length of svar1 is set to 0 and the value of svar is terminated at 133 characters. If svar is dimensioned smaller than the record being read, svar is set to its dimensioned size, and svar1 is set to zero.
4. If a response to the LREAD prompt is shorter than 133 characters, the length of svar1 is 1 and it contains the delimiter. Under AOS and AOS/VS, if a function key is the delimiter, the length of svar1 is 2 and it contains the 2-byte function key sequence.
5. If you end an LREAD statement variable list with a semicolon, the cursor remains at the position following the last item read. Otherwise, the system outputs a carriage return-line feed (NEW LINE).

Examples

```
* LREAD A$,B$
? ABCEDF
* PRINT A$
ABCEDF
* PRINT B$
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

LREAD FILE

Reads a string from a record, in either a random- or sequential-access file, that has a null, form feed, or carriage return (NEW LINE) terminator.

Format

LREAD { FILE / # } { (file) / (file,record) } ,svar *[,svar1]*

Arguments

\# A synonym for the keyword FILE

file A numeric expression that evaluates to a file number previously associated with an OPEN FILE statement

record A numeric expression that evaluates to the number of a record in a file opened for random access

svar A string variable to which BASIC assigns the value of the string read from the file

svar1 A string variable to which BASIC assigns the value of the delimiter for the string read. Valid delimiters are null, form feed, and carriage return (NEW LINE).

Remarks

1. LREAD is specifically intended to process text where commas and quotes are input characters, not delimiters.
2. The maximum string length allowed for svar is 133 characters, which includes the delimiter. If the record read is longer than 133 characters, BASIC sets the length of svar1 to 0, and truncates the value of svar at 133 characters.
3. If the record read is shorter than 133 characters, the length of svar1 is 1, and svar1 contains the delimiter.
4. If svar is dimensioned smaller than the record being read, svar is set to its dimensioned size, and svar1 is set to zero.
5. The number of the first record in a random-access file is 0.
6. You can use the EOF function to detect the end of the file being read.

Example

```
* LIST
0010 DIM A$(60)
0020 REM THIS ROUTINE USES TESTFILE1
0030 REM CREATED IN LWRITE FILE EXAMPLE.
0040 OPEN FILE(0,3),"TESTFILE1"
0050 FOR I = 1 TO 3
0060      LREAD FILE(0),A$,B$
0065      PRINT B$
0070      PRINT A$
0080 NEXT I
0090 CLOSE
*RUN
MONDAY
WEDNESDAY
FRIDAY
END AT 0090
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

LWRITE

Writes a string to your terminal.

Format

LWRITE svar *[,svar1]*

Arguments

svar — A string variable whose value is written to the terminal

svar1 — A string variable that is the value of the delimiter for the string written

Remarks

1. If the argument list includes svar1, and its length is 1, BASIC assumes svar1 is one of the valid delimiters and outputs it as the string terminator.
2. If the length of svar1 is 0, no delimiter is output.
3. If the argument list does not include svar1, a null is output as the string terminator.
4. LWRITE allows direct output of control characters to the terminal. For example, LWRITE does not insert a line feed after a carriage return; and it can instruct that no extraneous string terminators are to be output.
5. You must use LWRITE when you write data (e.g., terminal cursor position addresses) that would otherwise be interpreted as delimiters. If you do not use LWRITE, the system tries to use certain characters as delimiters, which will probably produce undesirable results.

Examples

```
* A$="ABCDEFGH"
* A1$=""
* LWRITE A$,A1$
ABCDEFGH*
* LWRITE A$
ABCDEFGH*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

LWRITE FILE

Writes a string to a record into either a random- or sequential-access file.

Format

LWRITE { FILE / # } { (file) / (file,record) } ,svar *[,svar1]*

Arguments

#	A synonym for the keyword FILE
file	A numeric expression that evaluates to a file number previously associated with an OPEN FILE statement
record	A numeric expression that evaluates to the number of a record opened for random access
svar	A string variable whose value BASIC writes to a file
svar1	A string variable that contains the value of the delimiter for the string written

Remarks

1. LWRITE is especially useful for creating records with nonstandard delimiters or for creating records in small "pieces" to be read later in larger "chunks."
2. If you include svar1 in the argument list, and set its length to 1, BASIC then assumes svar1 is one of the valid delimiters, and outputs it as the string terminator.
3. If you set the length of svar1 to 0, BASIC outputs no delimiter.
4. If you do not include svar1 in the argument list, BASIC outputs a null as the string terminator.
5. You must use LWRITE when you write data that the system will otherwise interpret as delimiters. If you do not use LWRITE FILE, the system tries to use the data as a delimiter, which produces undesirable results.

Example

```
* LIST
0010 DIM A$(60)
0020 LET Z$="<13>"
0030 LET A$="MONDAY WEDNESDAY FRIDAY"
0040 OPEN FILE(1,1),"TESTFILE1"
0050 LET A=1
0060 LET J=10
0070 LET K=10
0080 FOR I=0 TO 2
0090      LWRITE FILE(1),A$(A,J),Z$
0100      PRINT A$(A,J)
0110      LET A=A+K
0120      LET J=J+K
0130 NEXT I
0140 CLOSE
*RUN
MONDAY
WEDNESDAY
FRIDAY

END AT 0140
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

Matrix, Addition and Subtraction

Perform the scalar addition or subtraction of two matrixes.

Format

MAT mvar1 = mvar2 {+ / -} mvar3

Argument

mvar A matrix variable name

Remarks

1. Arithmetic is performed on an element-by-element basis of mvar2 and mvar3 with the result assigned to the element of mvar1.
2. Matrixes mvar2 and mvar3 must have the same dimensions.
3. Matrix mvar1 may appear on both sides of the equal sign.

Example

```
* LIST
0010 DIM A(3,2),B(3,2),C(3,2)
0040 MAT READ B,C
0050 MAT A=B+C
0060 DATA -2,-5,3,4,.5,.1,6,4,-2,15,1.5,
0070 MAT PRINT B
0075 PRINT
0080 MAT PRINT C
0085 PRINT
0090 MAT PRINT A
* RUN
-2     -5
3      4
.5     .1

6      4
-2     15
1.5    4

4      -1
1      19
2      4.1

END AT 0090
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

Matrix, Assignment

Copies the elements of matrix mvar2 into matrix mvar1.

Format

MAT mvar1 = mvar2

Argument

mvar A matrix variable name

Remarks

In the matrix assignment statement matrix mvar1 assumes the identical dimensions and values of matrix mvar2, provided the storage space allocated to mvar1 is sufficient to accommodate the current dimension of mvar2. If mvar1 has not been dimensioned, it is dimensioned by an assignment.

Example

```
* LIST
0010 DIM A(2,2)
0020 LET A(1,1)=5
0030 LET A(1,2)=10
0035 MAT PRINT A
0040 MAT B=A
0045 PRINT
0050 MAT PRINT B
* RUN
5   10   (Matrix A)
0   0

5   10   (Matrix B)
0   0

END AT 0050
*
```

Line 40 assigns matrix B the same dimensions as matrix A and assigns each value in matrix A to the corresponding element in matrix B. Therefore, B(1,1) = 5 and B(1,2) = 10.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

Matrix, Determinant (DET)

Obtains the determinant of the last matrix inverted by an INV statement.

Format

var = DET(X)

Arguments

var A numeric variable

X A dummy argument, which is necessary but not used

Remarks

The value of the determinant calculated for the matrix is assigned to the numeric variable.

Example

```
* LIST
0020 DIM A(2,2)
0030 MAT READ A
0040 DATA 1,2,3,4
0050 MAT PRINT A
0080 MAT A=INV(A)
0085 PRINT
0090 MAT PRINT A
0100 LET B=DET(X)
0120 PRINT
0130 PRINT "DETERMINANT=";B
* RUN
 1       2
 3       4

-2       1
 1.5    -.5

DETERMINANT=-2

END AT 0130
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

Matrix, Identity (IDN)

Sets each element of the major diagonal of the matrix to 1 and each remaining element to 0.

Format

MAT mvar = IDN *[([row,]col)]*

Arguments

mvar A matrix variable name

row The number of rows in the matrix

col The number of columns in the matrix, or elements in a vector

Remarks

1. The major diagonal starts at the last element of the array and runs diagonally upward until it encounters the first row or first column.
2. For previously dimensioned matrixes, use the form

   MAT mvar = IDN
3. If the matrix was not previously dimensioned or if the matrix is to be redimensioned, use the form

   MAT mvar = IDN(*[row,]*col)

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

Examples

```
* LIST
0050 DIM A(4,4)
0100 MAT A=IDN
0150 MAT PRINT A
* RUN
 1   0   0   0
 0   1   0   0
 0   0   1   0
 0   0   0   1

END AT 0150
*

* LIST
0010 DIM B(4,3)
0015 MAT PRINT B
0020 PRINT
0025 MAT B=IDN(2,3)
0030 MAT PRINT B
* RUN
 0   0   0
 0   0   0
 0   0   0
 0   0   0

 0   1   0   (Matrix B after line 25.)
 0   0   1

END AT 0030
*
```

Matrix, Inverse (INV)

Provides a matrix inversion of mvar2 and assigns the resultant matrix element value to mvar1.

Format

MAT mvar1 = INV(mvar2)

Argument

mvar A matrix variable name

Remarks

1. An inverse matrix is defined such that the product of a matrix and the inverse of the matrix is the identity matrix (see "Matrix, Identity").
2. Matrix mvar2 must be a square matrix.
3. Matrixes may be inverted into themselves (i.e., mvar1 equals mvar2 before the assignment in the matrix INV statement).
4. If the message ERROR 08 - SINGULAR MATRIX appears, mvar2 is singular or nearly singular, and DET(X) will be 0. In the same light, if DET(X) is very small in absolute value relative to the elements of mvar2, you should consider mvar2 to be singular or nearly singular and use the result, mvar1, with discretion.
5. The arithmetic of matrix inversion requires a knowledge of matrix determinants and of matrix cofactors. For further information on these subjects, consult a mathematics text.

Matrix, Inverse (INV) (continued)

Examples

```
* LIST
0010 DIM A(2,2)
0015 MAT READ A
0020 DATA 1,2,3,4
0030 MAT A=INV(A)
0040 MAT PRINT A
* RUN
-2      1
1.5    -.5

END AT 0040
*
```

This example may be analyzed as follows:

$\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix}$ = matrix A

Then:

$\begin{vmatrix} 4 & -2 \\ -3 & 1 \end{vmatrix}$ = cofactor of matrix A

$\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix}$ = (1*4) - (2*3) = -2 = determinant of matrix A

$$\text{INV(A)} = (1/-2) \begin{vmatrix} 4 & -2 \\ -3 & 1 \end{vmatrix} = \begin{vmatrix} -2 & 1 \\ 1.5 & -.5 \end{vmatrix}$$

When the original matrix is multiplied by its inverse,

$\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} * \begin{vmatrix} -2 & 1 \\ .5 & -.5 \end{vmatrix}$

the result is the identity matrix:

$\begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix}$

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

Matrix, Multiplication

Multiplies a matrix by a numeric expression or another matrix.

Format

MAT mvar1 = { mvar2 / (expr) } * mvar3

Arguments

mvar A matrix variable name

expr Any numeric expression enclosed in parentheses

Remarks

1. Matrix mvar1 may not be the same as either matrix mvar2 or mvar3 if you are multiplying the two matrixes. Otherwise, mvar1 and mvar3 can be the same if you are multiplying by a scalar, as in the statement:

   MAT mvar1 = (expr) * mvar3

2. If matrixes mvar2 and mvar3 are multiplied, the number of columns in mvar2 must equal the number of rows in mvar3. The resultant matrix (mvar1) has the same number of columns as mvar3, and the same number of rows as mvar2.

3. If a matrix is multiplied by a numeric expression, a scalar multiplication is performed on each element of the matrix.

4. To obtain the product of mvar2 * mvar3, the elements of each row in mvar2 are multiplied by the elements of each column in mvar3. Each row/column set is added together to provide the resultant value of the matrix element in mvar1.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

Examples

```
* LIST
0001 REM-MATRIX MULTIPLICATION
0010 DIM A(2,2),B(2,2)
0020 MAT READ B
0030 MAT A=(5)*B
0040 DATA -.5,.8,1.5,-1
0050 MAT PRINT B
0055 PRINT
0060 MAT PRINT A
* RUN
-.5   .8
1.5   -1

-2.5   4
7.5   -5

END AT 0060
*
```

```
* LIST
0001 REM-PRODUCT OF TWO MATRIXES
0010 DIM A(3,2),B(3,2),C(2,2)
0020 MAT READ B,C
0030 MAT PRINT B
0035 PRINT
0040 MAT PRINT C
0050 MAT A=B*C
0055 PRINT
0060 MAT PRINT A
0070 DATA 2,3,1,5,0,4,-1,-2,7,8
* RUN
2   3
1   5
0   4

-1   -2
7   8

19   20
34   38
28   32

END AT 0070
*
```

Matrix A is calculated as shown below:

[B(1,1)*C(1,1)+B(1,2)*C(2,1)] [B(1,1)*C(1,2)+B(1,2)*C(2,2)]

[B(2,1)*C(1,1)+B(2,2)*C(2,1)] [B(2,1)*C(1,2)+B(2,2)*C(2,2)]

[B(3,1)*C(1,1)+B(3,2)*C(2,1)] [B(3,1)*C(1,2)+B(3,2)*C(2,2)]

$$= \begin{matrix} [2*(-1)+3*7] & [2*(-2)+3*8] \\ [1*(-1)+5*7] & [1*(-2)+5*8] \\ [0*(-1)+4*7] & [0*(-2)+4*8] \end{matrix} = \left| \begin{matrix} 19 & 20 \\ 34 & 38 \\ 28 & 32 \end{matrix} \right|$$

Matrix, Transposition (TRN)

Transposes matrix mvar2 and assigns the resultant element values to mvar1.

Format

MAT mvar1 = TRN (mvar2)

Argument

mvar A matrix variable name

Remarks

1. Transposing a matrix reverses the row and column assignments of the matrix elements.
2. Variable names mvar1 and mvar2 cannot be the same in a TRN statement.
3. BASIC redimensions the resultant matrix, mvar1, to the reversed row and column dimensions of mvar2.

Example

```
* LIST
0020 DIM B(3,4)
0030 MAT READ B
0040 DATA 4,5,7,9,0,0,0,0,1,3,5,7
0050 MAT PRINT B
0060 PRINT
0080 MAT A=TRN(B)
0090 MAT PRINT A
* RUN
4   5   7   9
0   0   0   0
1   3   5   7

4   0   1
5   0   3
7   0   5
9   0   7

END AT 0090
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

Matrix, Unit (CON)

Sets the value of each element in a matrix to 1.

Format

MAT mvar = CON *[([row,]col)]*

Arguments

mvar A matrix variable name

row The number of rows in the matrix

col The number of columns in the matrix, or elements in a vector

Remarks

1. For previously dimensioned matrixes, use the form

   MAT mvar = CON

2. If the matrix was not previously dimensioned or if the matrix is to be redimensioned, use the form

   MAT mvar = CON (*[row,]*col)

3. All matrix elements are set to 1 regardless of any previously assigned values.

Example

```
* LIST
0010 DIM A(2,5)
0020 READ A(1,1),A(1,2),A(1,5)
0030 DATA 8,9,10,11,12
0040 MAT PRINT A
0045 PRINT
0050 MAT A=CON(2,4)
0060 MAT PRINT A
* RUN
 8  9  0  0  10
 0  0  0  0  0

 1  1  1  1  (Matrix A after line 50.)
 1  1  1  1

END AT 0060
*
```

In line 50 matrix A is redimensioned, and all elements of the matrix are assigned a value of 1.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

Matrix, Zero (ZER)

Sets the value of each element in a matrix to 0.

Format

MAT mvar = ZER *[([row,]col)]*

Arguments

mvar A matrix variable name

row The number of rows in matrix

col The number of columns in matrix, or number of elements in a vector

Remarks

1. For previously dimensioned matrixes, use the form

   MAT mvar = ZER

2. If the matrix was not previously dimensioned or if the matrix is to be redimensioned, use the form

   MAT mvar = ZER (*[row,]*col)

3. All matrix elements are set to 0 regardless of any previously assigned values.

Example

```
* LIST
0010 DIM A(3,4)
0020 LET A(1,2)=6
0030 LET A(3,4)=10
0040 MAT PRINT A
0045 PRINT
0050 MAT A=ZER(3,3)
0060 MAT PRINT A
* RUN
 0  6  0  0
 0  0  0  0
 0  0  0  10

 0  0  0
 0  0  0  (Matrix A after line 50.)
 0  0  0

END AT 0060
*
```

In line 50 matrix A is redimensioned, and all elements are assigned a value of 0.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

MAT INPUT

Reads values from your terminal and assigns them to the elements of a matrix or list of matrixes.

Format

MAT INPUT *["str lit",]* mvar *[([row,]col)] [,mvar[([row,]col)]]...*

Arguments

mvar A matrix variable name

row The number of rows in the matrix

col The number of columns in the matrix, or elements in a vector

str lit A string literal supplying a message or prompt

Remarks

1. You can dimension or redimension a matrix with a MAT INPUT statement.
2. Enter data values, separated by either a comma or a carriage return, for each element of the matrix. Terminate the list with a carriage return.
3. If you do not supply enough data to fill the matrix before typing the carriage return, the program continues to prompt for data until each element of the matrix has been filled.

Example

```
* LIST
0010 MAT INPUT X(2,3)
0015 PRINT
0020 MAT PRINT X
* RUN
? 2,4,6
? 77,7,9

 2   4   6
 77  7   9

END AT 0020
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

MAT INPUT FILE

Reads a record of matrix data in ASCII format from a file.

Format

MAT INPUT { FILE / # } { (file) / (file,record) } ,mvar *[,mvar]...*

Arguments

\# A synonym for the keyword FILE

file A numeric expression that evaluates to the number of a file opened for sequential or random access

record A numeric expression that evaluates to the number of a record in a file opened for random access

mvar A matrix array whose values are read from a record in a sequential- or random-access file

Remarks

1. You may list previously dimensioned matrix arrays in the statement by name only. Matrix arrays that have not been dimensioned must be dimensioned in a MAT INPUT FILE statement.
2. BASIC reads data items from the file sequentially and assigns them to the array elements by row.
3. You must separate data items in the file with a comma or carriage return.
4. You may use the EOF function to detect an end of file in the file that is being read.
5. The first record in a random-access file is 0.

Example

```
* 05 DIM Y(7,6),Z(13,2)
* 10 OPEN FILE (2,3), "XX.AA"
* 50 MAT INPUT FILE (2),X(5,5),Y,Z
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

MAT PRINT

Prints the values of the elements of a matrix or list of matrixes to your terminal.

Format

MAT PRINT mvar [{,;} *mvar*] ... [{,;}]

Arguments

mvar A matrix variable name

Remarks

1. Use a DIM statement or other matrix statement to dimension a matrix before using it in a MAT PRINT statement.
2. A semicolon after a variable name in the MAT PRINT statement indicates the matrix is to be printed in compact format. A comma or carriage return after the variable name indicates the matrix is to be printed in zone format.
3. The system prints column vectors (arrays) one value per line.

Example

```
* LIST
0010 DIM A(10,10)
0020 READ N
0030 MAT A=CON(N,N)
0050 FOR I=1 TO N
0060     FOR J=1 TO N
0070     LET A(I,J)=1/(I+J-1)
0080     NEXT J
0090 NEXT I
0130 MAT PRINT A
0190 DATA 3
* RUN
1            .5           .33333333
.5           .33333333    .25
.33333333    .25          .2
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

MAT PRINT FILE

Writes a record of matrix data in ASCII format into a sequential- or random-access file.

Format

MAT PRINT {FILE / #} {(file) / (file,record)}, mvar [{,;} *mvar*...] [{,;}]

Arguments

#	A synonym for the keyword FILE
file	A numeric expression that evaluates to the number of a file opened for sequential or random access
record	A numeric expression that evaluates to the number of a record in a file opened for random access
mvar	A matrix whose values are written into a record of a random- or sequential-access file

Remarks

1. Use this statement for outputting to an ASCII device, such as a line printer, or to a disk file for off-line printing.
2. A semicolon after a matrix variable, rather than a comma or carriage return, indicates that the matrix immediately preceding the semicolon is printed in compact format rather than zone format.
3. The first record number in a random-access file is 0.
4. Do not use the MAT INPUT FILE statement to input data that the MAT PRINT FILE has output; MAT PRINT FILE does not output delimiters between matrix elements. Use the MAT WRITE FILE statement to output data that will be reinput later using the MAT READ statement.

Example

```
* 05 DIM B(20,20)
* 10 OPEN FILE (0,1),"NUHROK"
* 20 MAT PRINT FILE (0), B
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

MAT READ

Reads values from the data list and assigns them to the elements of the matrix or matrixes listed in the MAT READ statement.

Format

MAT READ mvar *[([row,]col)] [,mvar[([row,]col)]]...*

Arguments

mvar A matrix variable name

row The number of rows in the matrix

col The number of columns in the matrix, or elements in a vector.

Remarks

If a matrix has not been previously dimensioned, a MAT READ statement dimensions it.

Example

```
* LIST
0010 MAT READ M(5,6)
0020 DATA 0,2,4,6,8,10,-9,-8,-7,-6,-5
0030 DATA -4,-3,-2,-1,0,1,3,5,7,9,11
0040 DATA .1,0,.5,7,-8,15,-15,35,41,13,18
0050 MAT PRINT M
* RUN
 0   2   4   6   8   10
-9  -8  -7  -6  -5  -4
-3  -2  -1   0   1   3
 5   7   9   11  .1  0
 .5  7  -8   15 -15  35

END AT 0050
*
```

This example reads values from the data list into the 30-element matrix dimensioned as 5 by 6 in the MAT READ statement.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

MAT READ FILE

Reads a data record from a sequentially or randomly accessed file, in binary format, for the elements of matrix arrays.

Format

MAT READ { FILE / # } { (file) / (file, record) }, mvar *[,mvar]...*

Arguments

\# A synonym for the keyword FILE

file A numeric expression that evaluates to the number of a file opened for random or sequential access

record A numeric expression that evaluates to the number of a record in a file opened for random access

mvar A matrix that is assigned values read sequentially from a randomly or sequentially accessed record

Remarks

1. The number of the first record in a random-access file is 0.
2. You may list previously dimensioned matrix arrays in the statement by name only. You must dimension matrix arrays that have not been dimensioned in the MAT READ FILE statement.
3. In randomly accessed files, records that have not been written into contain all zeros when read.
4. The system reads data items from the record sequentially and assigns them to the array elements by row.
5. You can use the EOF function to detect an end of file in the file that is being read.
6. The amount of data to be read must not exceed the record size specification for files opened for random access.

Example

```
* 10 DIM A(7,3), B(12,7)
* 30 OPEN FILE (1,3), "MATRIXA"
* 40 MAT READ FILE (1), A, B, C(3,4), D(5)
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

MAT TINPUT

Reads values from the terminal and assigns them to the elements of a matrix or list of matrixes, within a prescribed time.

Format

MAT TINPUT *[(line [,time]),][“str lit”,]*
mvar *[([row,]col)] [,mvar[([row,]col)]]...*

Arguments

line A valid statement line number

time A numeric expression that evaluates to an integer and represents time, in seconds

str lit A string literal supplying a message or prompt

mvar A matrix variable name

row The number of rows in the matrix

col The number of columns in the matrix, or elements in a vector

Remarks

The remarks for the MAT INPUT and TINPUT statements apply to the MAT TINPUT statement.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

MAT WRITE FILE

Writes a record of matrix data in binary form into a sequentially or randomly accessed file.

Format

MAT WRITE { FILE / # } { (file) / (file,record) } ,mvar *[,mvar]...*

Arguments

\# A synonym for the keyword FILE

file A numeric expression that evaluates to the number of a file opened for random or sequential access

record A numeric expression that evaluates to the number of a record in a file opened for random access

mvar A matrix whose values are written into a record of a randomly or sequentially accessed file

Remarks

1. The number of the first record in a random-access file is 0.
2. Matrix arrays listed in the MAT WRITE FILE statement must be previously dimensioned.

Example

```
* 50 OPEN FILE (0,1), "AAA"
* 80 MAT WRITE FILE (0),B,C,X
```

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

MAX

Establishes a limit for the number of active users.

Format

MAX *[val]*

Argument

val Any number from 0 to 33

Remarks

1. MAX must be executed from the master terminal. In a single-user system, the master terminal and user terminal are the same. In multiuser environments, programs using MAX as a statement do not work when run from terminals other than the master terminal.
2. The initial value for MAX is 33.
3. You may set MAX to a value smaller than the current number of active users.
4. If the number of active users exceeds the current MAX value, a user attempting to log on receives the following message and is denied access to the system:

   MAXIMUM USERS
5. A user can log on successfully if the number of active users is less than the current value of MAX.
6. If you use MAX without an argument, the system returns the current value of MAX.

Examples

```
* MAX = 7
* MAX
7
*
```

AOS, AOS/VS	✓
RDOS, DOS	

S	✓
C	✓
F	

MSG

Transmits a message from your terminal to another user or to the system operator.

Format

MSG { pid / "processname" / "console name" }, "message"

Arguments

pid	The process identification of the receiving user
processname	A process name expressed as a string literal or string variable
console name	A console identification (e.g., @CON1)
message	The text of the message, expressed as a string variable or string literal

Remarks

1. Message length is limited to one line per MSG command.
2. Quotation marks are necessary if you express the message as a string literal.
3. MSG without arguments turns the message receive flag on, in case it has been turned off by NOMSG.
4. If the transmission succeeds, BASIC prints your process identification number and message at the receiving terminal, in the following format:

   FROM PID XXX: message
5. If a receiving user is not on line, the transmission fails, and BASIC prints an error message at your terminal.
6. If a receiving user has disabled message reception by using CHAR with the NRM argument, or NOMSG, the transmission fails, and BASIC prints an error message.

Example

```
* MSG 11,"RSVP"
*
```

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

MSG

Transmits a message from your terminal to another user or to the operator.

Format

MSG *[userID message]*

Arguments

userID The identification of the receiving user

message The text of the message

Remarks

1. The system operator's userID is OPER.
2. Message length is limited to one line per MSG command.
3. Quotation marks are not necessary in the message.
4. MSG without arguments turns the message receive flag on, in case it has been turned off by NOMSG.
5. If the transmission succeeds, BASIC prints your identification and message at the receiving terminal, in the following format:

   FROM sendersID: message

6. If a receiving user is not on line, the transmission fails and BASIC prints an error message at your terminal.
7. If a receiving user has disabled message reception by using CHAR with the NRM argument, or NOMSG, the transmission fails, and BASIC prints an error message.

Example

UserID JACK types:

```
* MSG OPER MOUNT MY CASSETTE-THANKS
```

The master terminal receives:

FROM JACK: MOUNT MY CASSETTE-THANKS

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

NEW

Clears the program and variables currently stored in the program storage area and closes any open files.

Format

NEW *["filename"]*

Argument

filename A string literal or string variable for an ASCII listing file of a BASIC program

Remarks

1. Clearing your storage area with NEW before entering a new program, prevents the intermixing of lines from previous programs with the new program.
2. If you make NEW an executable statement in a program, the program clears itself from memory when BASIC executes NEW, and does not issue a STOP or END message.
3. You can combine the ON ESC or ON ERR statements with NEW to prevent unauthorized access to a program.
4. NEW closes any files left open by previously executed programs.
5. Using the filename argument is equivalent to the following pair of commands (or statements):

   ```
   NEW
   ENTER "FILENAME"
   ```

   The NEW command clears the storage area even if the filename does not exist.

Example

```
* LIST
0100 READ A,B,C,D
0110 LET E=A*23
0115 LET F=C*A
0120 PRINT E;F
0130 NEW
0135 DATA 1,2,3,4
* RUN
23 3
* LIST
ERROR 05 - LINE NUMBER
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	
F	

NEXT

Changes the value of the control variable in a FOR-NEXT statement.

Format

NEXT control var

Argument

control var A nonsubscripted numeric variable

Remarks

See the description for FOR and NEXT.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

NOECHO

Disables character display on input.

Format

NOECHO

Remarks

1. NOECHO is equivalent to the command CHAR "OFF","EB0" under AOS and AOS/VS, and CHAR "ON","NOE" under RDOS and DOS.
2. NOECHO is disabled by ECHO.
3. Input delimiters are not echoed when NOECHO is on, or both device characteristics EB0 and EB1 are off, and program input is required.

Example

```
* LIST
0010 NOECHO
0020 INPUT "→",A$
0030 ECHO
0040 INPUT "→",B$
0050 ; A$;" ";B$
0060 END
* RUN
→ (Character display is suppressed.)
→ TURNBULL
DREW TURNBULL

END AT 0060
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

NOESC

Disables the ESC key from producing an interrupt.

Format

NOESC

Remarks

1. NOESC is equivalent to the command CHAR "OFF","ESC".
2. ESC cancels the effect of NOESC.

Example

```
0005 NOESC
0010 FOR I = 1 TO 100
0020      PRINT I
0030      IF I = 50 THEN ESC
0040 NEXT I
0050 END
* RUN
1
2
3
   .
   .
ESC
   .
   .
50
   .
   .
ESC
STOP AT 0020
*
```

In this example, if I < 50 and the user presses ESC, an interrupt does not occur. However, once I > 50, pressing the ESC key causes an interrupt.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

NOMSG

Disables receipt of messages at the terminal from which the command or statement is issued.

Format

NOMSG

Remarks

1. NOMSG is equivalent to the command CHAR "ON","NRM".
2. On RDOS, NOMSG is overridden by the FMSG command.

Example

```
* WHO
PID: 20 DREW:020
* NOMSG
* MSG 20,"HELLO DREW"
I/O ERROR 100 - MESSAGE RECEIVE DISABLED
*
```

In this example, the user attempts to send a message to himself. Because the NOMSG command has been issued, the message is disabled.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	
F	

ON ERR THEN

Directs program control to an error-handling routine in your program instead of the BASIC system error handler.

Format

ON ERR { THEN line no. / *[THEN]* statement }

Arguments

statement — Any BASIC statement except those listed under "Remarks"

line no. — A program statement line number

Remarks

1. Do not use the following BASIC statements as the statement argument: FOR, NEXT, DEF, END, DATA, or REM.
2. Normally, when a BASIC error occurs, BASIC interrupts any operation in progress, prints an error message at the terminal, and places the terminal in interactive mode. If the system encounters an ON ERR THEN statement during execution of your program, any subsequent error causes the statement argument to be executed, or control to be transferred to the specified line number.
3. Place the ON ERR statement at the beginning of your program if you want all system errors handled by your error routine. If you place the ON ERR statement anywhere else in your program, BASIC executes your error routine only for errors that occur after it encounters the ON ERR statement.
4. If the statement argument is a GOSUB, after the subroutine is finished (RETURN), control passes to the statement following the one that caused the error. Do not use a RETRY statement in the body of the subroutine.
5. If the statement argument is other than STOP, GOTO, or GOSUB, BASIC executes the argument and passes program control to the statement following the one in which the error occurs.
6. You can restore the normal handling of errors by including the following statement in an appropriate place in your program:

   ```
   ON ERR THEN STOP
   ```

7. The keyword THEN is optional if a statement is specified; it is required if you specify line no.
8. Beware of getting into an infinite loop with an ON ERR-GOTO statement. If an error occurs in the error handling routine specified by the GOTO statement, looping could occur.

Example

```
0010 ON ERR THEN GOTO 1000
0020 OPEN FILE (0,0), "X"
0030 ON ERR THEN STOP
  .
  .
  .
1000 OPEN FILE (0,0), "Y"
1010 GOTO 30
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	
F	

ON ESC THEN

Directs program control to an escape-handling routine in your program instead of the BASIC system escape handler.

Format

ON ESC { THEN line no. / *[THEN]* statement }

Arguments

statement Any BASIC statement except those listed under "Remarks"

line no. A program statement line number

Remarks

1. Do not use the following BASIC statements as the statement argument: FOR, NEXT, DEF, END, DATA, REM
2. Normally, when you press the ESC key, you interrupt any operation in progress and place the terminal in interactive mode to wait for your next command. If BASIC encounters an ON ESC THEN statement while executing a program, pressing the ESC key causes the statement represented by the argument in the ON ESC THEN to be executed.
3. Place the ON ESC statement at the beginning of your program if you want your routine to handle all escapes. Place the ON ESC statement anywhere else in your program and BASIC executes your routine only for escapes that occur after it encounters the ON ESC statement.
4. If the statement argument is a GOSUB, after the subroutine is finished (RETURN), control passes back to the point of interruption.
5. Since ESC is not an error condition, do not use the RETRY statement in association with an ON ESC statement.
6. If the statement argument is other than STOP, GOTO, or GOSUB, then BASIC executes the statement argument and passes program control back to the point of interruption.
7. You can restore the normal handling of ESC by including the following statement in an appropriate place in your program:

   ON ESC THEN STOP

8. The keyword THEN is optional if you specify a statement; it is required if you specify a line number.

Examples

```
0100 ON ESC THEN PRINT X,Y,Z
      .
      .
      .
0140 PRINT X
0141 Y=Z
```

In this example, when you press the ESC key during program execution, control passes to the statement on line 100 and the values of X, Y, and Z are printed. After BASIC executes line 100, the program continues from the point of interruption. Therefore, if BASIC had completed line 140 when you pressed ESC, it would execute line 100 followed by line 141.

```
0010 ON ESC THEN GOSUB 0500
0020 DIM X(2500)
0021 LET A=0
0022 LET B=0
0023 LET C=0
0030 FOR I=1 TO 2500
0040      LET X(I)=A*I^2+B*I+C
0050 NEXT I
0060 STOP
0500 PRINT I,X(I)
0510 INPUT "CONTINUE (0), NEW INPUT (1)",D
0520 IF D=0 THEN RETURN
0530 INPUT "NEW VALUES FOR A,B,C=",A,B,C
0540 RETURN
```

In this example, a RETURN from line number 520 or 540 positions you to the line after the last executed line when the ESC key is pressed, not to line 20.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	
F	

ON-GOTO and ON-GOSUB

Transfers control to one of several lines in a program, depending on the value of an expression at the time BASIC executes the statement.

Format

ON expr { GOTO / GOSUB } line no. *[,line no.]...*

Arguments

expr — A numeric expression evaluated to an integer

line no. — A list of line numbers in the current program whose positions in the argument list are numbered from 1 through *n*

Remarks

1. The system evaluates the expression, ignoring any fractional portion.
2. The program transfers control to the line number whose position in the argument list corresponds to the computed value of the expression.
3. If the expression evaluates to an integer that is greater than the number of entries given in the argument list or that is less than or equal to zero, BASIC ignores the ON statement and passes control to the next statement.
4. The ON-GOSUB statement must contain an argument list whose entries are the first line of subroutines within the current program.

Examples

```
* 10 ON M-5 GOTO 500, 75, 1000
```

If M-5 evaluates to 1, 2 or 3, control passes to statement 500, 75, or 1000, respectively. If M-5 evaluates to any other value, control passes to the next statement in the program.

```
* 10 ON (SGN(M-5)+2) GOTO 100, 200, 300
```

The above statement is equivalent to the following three statements.

```
* 10 IF M-5<0 THEN GOTO 100
* 20 IF M-5=0 THEN GOTO 200
* 30 IF M-5>0 THEN GOTO 300
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

OPEN FILE

Assigns a file number and access mode to a filename for future referencing in file I/O statements in your program.

Format

OPEN { FILE / # } (file,mode), "filename" *[,record size [,file size]]*

Arguments

\# — A synonym for the keyword FILE

file — A numeric expression that evaluates to a number from 0 through 7 for RDOS and DOS, or 0 through 15 for AOS and AOS/VS. BASIC uses this number to simplify the reference to "filename" in other file I/O statements.

mode — A numeric expression that evaluates to a number from 0 to 3, or 7. This number specifies the access mode of the file. See "Remarks" for a description of modes.

filename — A string literal or string variable that evaluates to a valid filename

record size — An optional numeric expression that evaluates to a fixed length (in bytes) for each record in a file. A record size may be any value from 1 to 32768. In mode 0, the system assigns a default value of 128 bytes per record.

If the output of a WRITE FILE statement has fewer than the specified number of bytes, the output is padded with nulls until it reaches the specified record length. If the output has more than the specified number of bytes, an error occurs.

file size — An optional numeric expression that evaluates to the maximum number of records in the file, and thereby limits its size. *This argument is used only on RDOS and DOS,* and only in conjunction with record size, to create a contiguously organized file. "Error 34 - Function Argument" results if the size of the last two numerical arguments in the OPEN statement is 2^{24} (16,777,216) bytes or more. You cannot have a contiguous file with variable length records.

OPEN FILE (continued)

Remarks

1. If the filename argument is a string literal value, the filename must be explicitly terminated with a NULL (<0>), carriage return (<13>), form feed (<12>), or a space (<32>).
2. Calculate record length as follows:

   - Numeric Data

     Single-precision: 4 bytes per data item
     Double-precision: 8 bytes per data item

   - String Data

     One byte per character in string, plus one byte for string delimiter

   - Arrays

     (No. of rows)*(No. of columns)*(precision)

     Note: precision is 4 for single-precision, 8 for double-precision.

2. The following paragraphs define modes 0-3 and 7:

   Mode 0—Open random-access file for input/output. The file is exclusively opened under AOS and AOS/VS, and sharable under RDOS and DOS. You may only open disk files in random mode for reading and writing. If a disk file is named and it does not exist in your directory, BASIC creates it. Record length is fixed by the record size argument or by the default value (128 bytes).

   Mode 1—Create a new output file. You can open either a disk file or an appropriate output device in mode 1. BASIC opens the file exclusively and permits only writes. The system creates a new file, initialized with 0 length. If the filename already exists in your directory, BASIC deletes the previous copy from the disk. In RDOS and DOS only, the system does not check to insure that only writes are used; use the CHATR command to prevent input use of an output file.

   Mode 2—Append output to an existing file. You can use this mode to open any file previously opened in mode 1 or mode 2. When you open an existing file in mode 2, the file pointer moves to the end of the file so that subsequent data written to the file extends it. If the file does not exist in your directory, it is created. The file is exclusively opened, and only writes are permitted. In RDOS and DOS only, the system does not check to insure that only writes are used; use the CHATR command to prevent input use of an output file.

   Mode 3—Open input file for reading only. You can open either a disk file or appropriate input device in mode 3. If you open a disk file in this mode, the file must already exist. Only reads are permitted from a file opened in mode 3, and the file is not exclusively opened. If BASIC does not find the file in your directory, BASIC searches for it in the library directory.

   Mode 7—Open sharable random-access file for input/output. For RDOS and DOS, mode 7 is identical to mode 0. As in mode 0, BASIC can perform input and/or output processes, and the default record length is 128 bytes. AOS and AOS/VS users who want to share files should use mode 7 and record locking. Such mode 7 users share all pages of data accessed from the file. As long as the same records are not updated, simultaneous use of the file does not lose updates. Without record locking, however, you have no protection from other users who might access the same record. You must take care, moreover, that the file already exists and that the file element size is a multiple of 4. If either of these is not true for the file, a program exception occurs.

   The following tables summarize the characteristics of the file open modes for the given operating systems.

File Open Modes on RDOS and DOS

Mode	Type	I/O	Access	Rule
0	Sharable	I/O	Random	Create file if it does not exist.
1	Exclusive write Sharable read	I/O	Random or sequential	Create file.
2	Exclusive write Sharable read	I/O	Random or sequential	Append to existing file or create file.
3	Sharable	I	Random or sequential	File must exist.
7	Sharable	I/O	Random	Create file if it does not exist.

File Open Modes on AOS and AOS/VS

Mode	Type	I/O	Access	Rule
0	Exclusive	I/O	Random	Create file if it does not exist.
1	Exclusive	O	Random or sequential	Create file.
2	Exclusive	O	Random or sequential	Append to existing file or create file.
3	Nonexclusive	I	Random or sequential	File must exist.
7	Shared	I/O	Random	File must exist.

3. Default value for the mode argument is 0. OPEN FILE (1) is the same as OPEN FILE (1,0).

Examples

```
* 100 OPEN FILE (1,1), "NETSAK.JR"
```

This statement opens file 1, named NETSAK.JR, as an output file.

```
* 100 OPEN FILE (2,0), "RESSEHC.TO",20
```

This statement opens the file named RESSEHC.TO as file number 2. Mode 0 specifies random access read or write. Records are 20 bytes long.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	
F	✓

ORD(X$)

Represents the ordinal position of a character in the ASCII collating sequence.

Format

ORD { (svar) / ("str lit") }

Argument

svar A string variable

str lit A string literal

Remarks

1. The argument of the ordinal function may be any string expression.
2. If the argument of the ordinal function is not a string of length 1, BASIC returns an error message.

Example

```
* LIST
0005 REM CONVERT UPPER- TO LOWERCASE
0010 LET A$="LOWER CASE"
0020 FOR I=1 TO LEN(A$)
0030 LET A=ORD(A$(I,I))
0040 IF A>=65 THEN IF A<=90 THEN LET
A$(I,I)=CHR$(A+32)
0050 NEXT I
0060 PRINT A$
0070 STOP
* RUN
lower case

STOP AT 0070
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

PAGE

Sets the right margin of your terminal.

Format

PAGE=expr

Argument

expr An arithmetic expression evaluating to a number between 15 and 132, inclusive, and not less than the current TAB setting. For AOS and AOS/VS systems, the expression evaluates to a number between 14 and *n*, where *n* can be as high as 255, depending on the system.

Remarks

1. BASIC uses a default value of 80 as the maximum line width.
2. Strings may be as large as 32767 bytes. However, if a string is longer than the PAGE value, it cannot be printed and BASIC outputs an error message. To examine the entire string, you must print substrings of lengths smaller than or equal to the PAGE value.

Example

```
* LIST
0010 PAGE =30
0020 FOR I=1 TO 25
0030     PRINT I;
0040 NEXT I
* RUN
1  2  3  4  5  6  7  8  9
10  11  12  13  14  15  16
17  18  19  20  21  22  23
24  25

END AT 0040
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	
F	✓

POS(X$,Y$,Z)

Determines the position of a substring in a string.

Format

POS { (svar1) / ("str lit1") }, { (svar2) / ("str lit2") }, (expr)

Arguments

svar A string variable

str lit A string literal

expr A numeric expression

Remarks

1. Starting at position expr in a string (svar1 or str lit1), BASIC searches for the specified substring (svar2 or str lit2).
2. The POS function returns the first position of the substring in the string. If the substring cannot be found in the string, the POS function returns a value of 0. If the value of the expression is less than 0, an error message occurs.
3. If the expression is greater than the length of the string, the POS function returns a value of 0.
4. If the expression is equal to 0, the search begins at the first position of the string.

Example

```
* LIST
0005 DIM A$(25)
0010 LET A$="AMNOPFGHIJKLMNOPQRS"
0020 LET A=POS(A$,"MNOP",6)
0030 PRINT A
* RUN
 13

END AT 0030
*
```

In this example, the program searches for "MNOP", starting from the sixth character (F) in string A$. It finds a match that begins at character 13 in string A$. POS, therefore, returns a value of 13, which is assigned to variable A. Had the expression equaled 1, POS would have found the first MNOP and assigned a value of 2 to A.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

PRINT

Performs print operations at your terminal.

Format

{ PRINT / ; } [{ svar / expr / "str lit" / TAB(n) } [{ , / ; } { svar / expr / "str lit" / TAB(n) }] ...] [{ , / ; }]

Arguments

; (first) A synonym for the keyword PRINT

svar A string variable

expr A numeric or string expression

str lit A message or prompt

TAB(n) Tabulates to column *n*

Remarks

The following PRINT operations are possible:

1. Print the result of a computation.
2. Print verbatim the characters in a string literal or string variable.
3. Print a combination of operations 1 and 2.
4. Print a blank line, i.e., skip a line.

Zone Spacing of Output

The print line on a terminal is divided into print zones. The width of a print zone is determined by the TAB statement. The default value for TAB is 14 and is used in the following examples. The first column on a line is column 1.

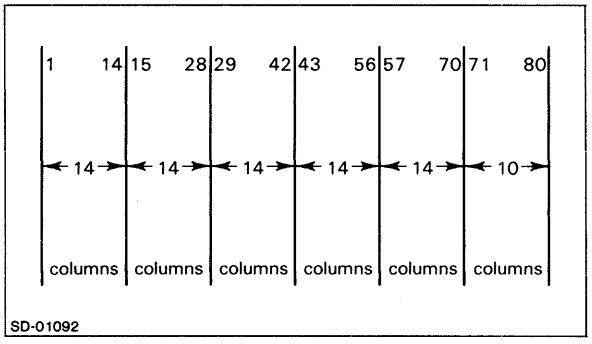


Figure 3-1.

PRINT (continued)

A comma (,) between items in the PRINT statement list prints the next item in the leftmost position of the next printing zone. If there are no more printing zones on the current line, printing continues in the first zone on the next line. If an item requires more than one print zone, BASIC prints the next item at the beginning of the next free print zone (see the first example).

Before each item is printed, its length is compared with the space remaining on the line. If insufficient space remains on the current line, the item is moved to the next line. If the length of the item is greater than the width of the page (see PAGE), BASIC issues an error message.

Compact Spacing of Output

A semicolon (;) between items in the PRINT statement list, prints the next item at the next character position. Note that the system reserves a space for the plus (+) sign, even though it is not printed (see second example), and there is always a trailing space.

Spacing to the Next Line

When BASIC prints the last item in a print list, it outputs a carriage return and line feed, unless the last item in the list is followed by a comma or semicolon. The carriage return and line feed are not output after commas or semicolons because of their punctuation function.

However, if the comma or semicolon would cause the next item to be printed beyond the allowable line width (see PAGE), a carriage return and line feed are output.

Printing Blank Lines

A PRINT statement with no list of print items or punctuation outputs a carriage return and line feed (see fourth example).

For more printing versatility, you can use the TAB(X) function and the TAB, PAGE, and PRINT USING statements.

The TAB(n) argument to the PRINT command allows you to print at the nth column position (see fifth example).

Examples

Column numbers are marked below examples.

```
* LIST
0010  LET X=25
0020  PRINT "SQUARE ROOT OF X IS; ",SQR(X)
* RUN
SQUARE ROOT OF X IS:         5


END AT 0020
*
↑             ↑             ↑
1            15            29
```

```
* LIST
0010  LET X=5
0020  PRINT X;(X*2) ↑6;X*2;(X*2)↑4;
0030  PRINT X-25;(X*2) ↑8;X-100>>)
* RUN
 5 1000000 10 10000-20 1E+08
-95

END AT 0030
*
↑ ↑       ↑ ↑     ↑ ↑
1 3       11 14    20 23
```

```
* LIST
0005  PAGE =70
0010  LET X=5
0020  PRINT X(X*2) ↑6,
0030  PRINT X ↑ 4
0040  PRINT "FIN"
* RUN
 5              1000000         625
FIN

END AT 0040
*
↑             ↑             ↑
1            14            29
```

Notice that the trailing comma in line 20 causes the value of X↑4 in line 30 to be printed in zone 3 instead of zone 1 of the next line.

```
* LIST
0010  LET X=5
0020  PRINT X;(X*2) ↑6,X*2
0030  PRINT X-25;(X*2) ↑8
0040  PRINT X-100
```

```
0050  PRINT
0060  PRINT "DONE"
* RUN
5 1000000       10
-20 1E+08
-95

DONE

END AT 0060
*
↑ ↑           ↑
1 4           15
```

In line 20, the comma and semicolon spacing characters are both used. Line 50 outputs a blank line before printing "DONE".

```
* 0010 PRINT "ABCDEFG";
* 0020 PRINT TAB(8);"HIJK"
* RUN
ABCDEFGHIJK

END AT 0020
*
```

```
* 0010 PRINT "ABCDEFG";
* 0020 PRINT TAB(3);"HIJK"
* RUN
ABCDEFG
  HIJK

END AT 0020
*
```

Note that if the current column is greater than TAB(n), printing continues on the next line.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

PRINT FILE

Writes data in ASCII format into a sequential- or random-access file.

Format

{PRINT / ;} {# / FILE} {(file) / (file,record)} , {expr / var / svar / "str lit"}

[{, / ;} {expr / var / svar / str lit}] ... [{, / ;}]

Arguments

; (first) — A synonym for the keyword PRINT

— A synonym for the keyword FILE

file — A numeric expression that evaluates to the number of a file opened for sequential or random access

record — A numeric expression that evaluates to the number of a record in a file opened for random access

expr, var, svar, and str lit... — A list of one or more numeric expressions, numeric variables, string variables, and string literals, whose values are written into a file

Remarks

1. This statement is intended for outputting to an ASCII device, such as a line printer, or to a disk file for later off-line printing.
2. Separate each item in the expression list from the next by a comma, semicolon, or carriage return. Output formatting is identical to that discussed in "Remarks" for the PRINT statement.
3. If an INPUT FILE statement will subsequently read the data written by PRINT FILE as numeric data, then it is necessary to print the comma (data separator) as a string literal between expressions, as shown in line 300 of the first example below.
4. The first record number in a random-access file is 0.

PRINT FILE (continued)

Examples

```
* 010 OPEN FILE(3,1), "$LPT"
* 100 PRINT FILE(3), "OUT6"
* 200 PRINT FILE(3),"X=";X,"X SQUARED=";X 2
* 300 PRINT FILE(3),A;",";B;",";C
* 400 CLOSE
```

```
* 010 OPEN FILE(3,1), "$LPT", 80
* 100 FOR I = 1 TO 10000
* 110 PRINT FILE(3),I;
* 120 NEXT I
* 130 CLOSE
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

PRINT FILE USING

Outputs the values of the expressions in the PRINT FILE USING statement to a previously opened file in the format specified.

Format

{PRINT / ;} {# / FILE} (file *[,record]*),USING format,expr *[,expr]*...

Arguments

;	A synonym for the keyword PRINT
#	A synonym for the keyword FILE
file	A numeric expression that evaluates to the number of a previously opened file (see PRINT FILE)
record	A numeric expression that evaluates to the number of a record in a file.
format	A string literal or string variable that specifies the output format (see "Remarks") for items in the expression list
expr...	A list of one or more numeric expressions, numeric variables, string variables, and string literals whose values are to be written into a file

Remarks

The remarks listed for the PRINT FILE statement and the argument descriptions in the PRINT USING statement apply to PRINT FILE USING.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

PRINT USING

Outputs the values of expressions in the PRINT USING statement list using the format specified.

Format

PRINT USING format, expr $\left[\left\{ \begin{matrix} ;expr... \\ ,expr \end{matrix} \right\} \right]$

Argument

format A string literal or string variable that specifies the format for printing the items in the expression list. See "Remarks."

expr A numeric or string expression

Remarks

1. BASIC ignores all normal PRINT formatting conventions for TAB, comma, and semicolon in a PRINT USING statement, except when you use a semicolon or comma at the end of a statement to inhibit a carriage return.

2. The format argument may have more than one format field. A format field may include string literals and the following special characters, which are used for formatting numeric output:

   # . + – $, ↑

   Remarks 3-10 describe the actions of these special formatting characters.

Note: In the following descriptions, a box (□) is used to clarify the presence of a blank space.

3. Digit Representation (#)

   For each number sign (#) in the format field, a digit (0 to 9) is substituted from the expression argument (see Table 3-2).

Table 3-2. Representing Digits by Number Signs

Format	Expr	BASIC Outputs	Remarks
####	25	□□25	Digits are right-justified in the field with leading blanks.
####	-30	□□30	Signs and other nondigits are ignored.
####	1.95	□□2	Only integers are represented; the number is rounded to an integer.
####	598745	****	If the number in expr has more digits than specified by format, all asterisks are output.

4. Decimal Point Representation (.)

   Use the decimal character (.) to place a decimal point in the fixed position in which it appears in the format. Digit (#) positions that follow the decimal point are filled; no blank spaces are left in these digit positions. When expr contains fewer fractional digits than the format specifies, 0s are output to fill the positions. When expr contains more fractional digits than the format allows, the fraction is rounded to the limits of format (see Table 3-3).

5. Fixed Sign Representation (+ and –)

   A fixed sign character appears as a single plus sign (+)or minus sign (–) in either the first or last character position in the format field.

   A fixed plus sign is used to print the sign (+ or –) of the expression in the position in which the fixed plus sign is placed in the format.

Table 3-3. Decimal Point Representation

Format	Expr	BASIC Outputs	Remarks
####.##	20	□□20.00	Fractional digit positions are filled with zeros.
####.##	29.347	□□29.35	Rounding off occurs on fractions.
####.##	789012.34	*******	When expr has too many significant digits to the left of a decimal point, a field of all all asterisks is output.

Table 3-4. Fixed Sign Representation

Format	Expr	BASIC Outputs	Remarks
+##.##	20.5	+20.50	Fractional digit positions are filled with zeros.
+##.##	1.01	+□1.01	Blanks precede the number.
+##.##	-1.236	-□1.24	Rounding off occurs on fractions.
+##.##	-234.0	*******	Too many digits occur to the left of the decimal point.
###.##-	20.5	□20.50□	Decimal digit positions are filled.
###.##-	000.01	□□0.01□	One leading zero is printed before the decimal point.
###.##+	1.236	□□1.24+	
###.##-	-234.0	234.00-	

Table 3-5. Floating Sign Representation

Format	Expr	BASIC Outputs	Remarks
---.##	-20	-20.00	The second and third minus signs are treated as # signs on output.
---.##	-200	*******	Too many digits occur in expr to the left of the decimal point.
+++.##	2	□+2.00	Blanks between the sign and digit are suppressed.
---.##	2	--2.00	

A fixed minus sign is used to print a minus sign for negative values of an expression or a blank space for positive values of an expression in the position in which the fixed minus sign is placed in the format.

When a fixed sign is used, any leading 0s appearing in the expression are replaced by blanks, except for a single leading 0 preceding a decimal point (see Table 3-4).

6. Floating Sign Representation (++ and --)

A floating sign appears as two or more plus or minus signs at the beginning of the format field.

The floating plus sign prints a plus or minus sign immediately before the value of the expression with no separating blank spaces as occur with fixed signs. A floating minus prints either a minus or blank (for plus) immediately preceding the value.

When you use floating signs, BASIC treats the second and subsequent signs in the format as number signs (#). BASIC replaces them with numbers from the expression as necessary (see Table 3-5).

Note: A format may include either a floating plus or minus sign or a floating $ sign (see remark 8), but not both.

7. Fixed Dollar Sign Representation ($)

When you use a dollar sign ($) as either the first or second character in the format field, BASIC prints a dollar sign ($) in that position. If the dollar sign ($) is in the second position, it must be preceded by a fixed sign (+ or –). A fixed dollar sign ($) causes leading 0s in the value of the expression to be replaced by blanks (see Table 3-6).

Table 3-6. Fixed Dollar Sign Representation

Format	Expr	BASIC Outputs
-$###.##	30.512	□$□30.51
$###.##+	-30.512	$□30.51-

Table 3-7. Floating Dollar Sign Representation

Format	Expr	BASIC Outputs	Remarks
+$$$#.##	13.20	+□□$13.20	Extra $ signs may be replaced by digits, as with floating + and – signs.
$$##.##	-1.00	□$01.00-	Leading zeros are not suppressed in the # part of the field.

Table 3-8. Separator Representation

Format	Expr	BASIC Outputs	Remarks
+$#,###.##	30.6	+$□□□30.60	Space printed for comma.
+$#,##.##	2000	+$2,000.00	
++##,###	00033	□+00,003	Comma is printed when leading zeros are not suppressed.

8. Floating Dollar Sign Representation ($$)

   A floating dollar sign appears as two or more dollar signs, beginning at either the first or second character in the format field. If the dollar signs start in the second position, they must be preceded by a fixed sign (+ or –).

   When you use a floating dollar sign BASIC prints a dollar sign immediately before the first digit of the expression value (see Table 3-7).

   Note: A format may include either a floating dollar sign ($$) or a floating plus or minus sign (see remark 6), but not both.

9. Separator Representation (,)

   When you use a comma separator (,) in a string of digits (#) in the format field, BASIC prints a comma in the fixed position in which it appears.

   If a comma is output in a field of suppressed leading 0s (blanks), then a blank space is output in the position for the comma (see Table 3-8).

10. Exponent Representation (↑↑↑↑)

    Use four consecutive up-arrows or circumflexes (↑↑↑↑) to indicate an exponent field in format. BASIC outputs the four up-arrows as E+*nn*, where each *n* is a digit.

    If the exponent field in the format does not have exactly four up-arrows, a runtime error results (see Table 3-9).

Table 3-9. Exponent Representation

Format	Expr	BASIC Outputs
+##.##↑↑↑↑	170.35	+17.04E+01
+##.##↑↑↑↑	-.2	-20.00E-02
++##.##↑↑↑↑	6002.35	+600.24E+01

11. A format expression may include more than one format field and may include string literals in addition to the special formatting characters. BASIC assigns values of the expression argument list sequentially to format fields.

    BASIC differentiates format fields from string literals by the characters that appear in format fields. For example:

"TWO FOR $1.25"	$1.25 is part of the string literal.
"TWO FOR $$$.##"	$$$.## is a format field in the format expression.
"ANSWER IS -85"	-85 are characters of the string literal.
"ANSWER IS -###"	-### is a format field in the format expression.

PRINT USING (continued)

12. You may specify a format expression by referring to a previously defined string variable, for example:

```
* 05 DIM S$(10)
* 10 LET S$="##.##"
* 20 PRINT USING S$, 1.5, 2
```

13. You must delimit the format fields in a format expression from each other by using any nonspecial formatting character after each format field. However, if the format expression in the PRINT USING statement is a string literal, you cannot use quotation marks (" ") as a field delimiter. BASIC treats delimiters as string literals and prints them on output.

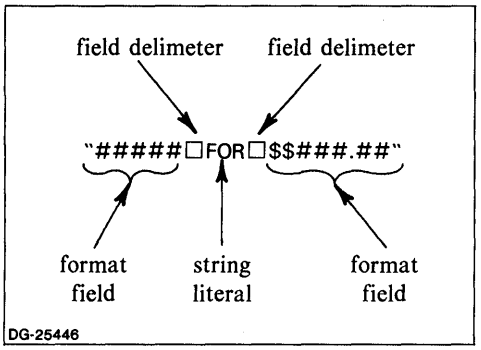


Figure 3-2.

14. String literals may appear in the expression argument list of the PRINT USING statement; they are superimposed on a format field in the following manner:

    a. Each character of the string literal replaces a single format field character, which may be any of the special format characters ($ # ^ + - . ,).

    b. Strings are left-justified in the format field, and filled with spaces if necessary.

    c. If the number of characters in the string is greater than the number of characters in the format field, BASIC truncates the string to fit the field. For example:

```
* 5 PRINT USING
  "#,###.##","TEST","CHARACTER"
* RUN
TEST□□□□CHARACTE
```

15. When there are more items in the expression argument list than format fields, BASIC uses the format fields repetitively. For example, in the following format argument, there are three format fields and two string literal fields. The string literal fields delimit the format fields.

"####□@$###.##□PER□###"

BASIC matches the expressions with the format fields in the following order:

Format Field	Order of Expressions
####	First, fourth, seventh, etc.
$###.##	Second, fifth, eighth, etc.
###	Third, sixth, ninth, etc.

The following statement yields output that may include two format fields and two string literals:

```
0100 PRINT USING "A(#)□=□##.#",I,A(I)
   .
   .
   .
RUN
A(1)□=□17.9
```

With the following statement the format expression is repeated as necessary for each item in argument list:

```
0100 PRINT USING "###.##□",I,A,B
   .
   .
   .
RUN
□□1.00□□17.90□□25.77□
```

16. When the number of characters on a line exceeds the page size, printing continues on the next line. When a literal is printed to a terminal, the literal must be shorter than the current PAGE width.

AOS, AOS/VS	
RDOS, DOS	✓

S	
C	✓
F	

PUNCH

Outputs part or all of the current program in ASCII to the terminal punch.

Format

PUNCH [{ *line n1* / *TO* / , } *line n2* | *line n1* { *TO* / , } *line n2*]

Arguments

line n1 The first statement to be punched

line n2 The last statement to be punched

Remarks

1. A leader of null characters precedes the punched listing, and a trailer of null characters follows the listing.
2. The number of null characters punched as leader and trailer equals the number defined as the page width (see PAGE.) This represents eight inches of leader for an 80-character line.
3. The PUNCH command does not turn on the terminal punch. The following procedure is required:
   a. Type the desired PUNCH command followed by a carriage return, and immediately press the ON button on the terminal punch.
   b. The machine punches a null leader, followed by a listing of the desired lines of the current program, followed by a null trailer.
   c. When you have completed punching, press the OFF button on the punch.
4. The variations of the PUNCH command are described as follows:

PUNCH	Punch the entire program starting at the lowest numbered statement.
PUNCH n1	Punch only the single statement at line number n1.
PUNCH { TO / , } n2	Punch from the lowest numbered line through line number n2.
PUNCH n1 { TO / , } n2	Punch from line number n1 through line number n2.

Example

```
* PUNCH 200 TO 500
```

Punch line numbers 200 through 500 of the current program.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

RANDOMIZE

Causes the random number generator to start at a different point in the sequence of random numbers generated by the RND function.

Format

RANDOMIZE

Remarks

1. RANDOMIZE resets the random number generator based on the time of day, thereby producing different random numbers each time you run a program using the RND function.
2. Without RANDOMIZE, the RND function generates the same sequence of random numbers each time you run a program. This feature is useful for debugging programs. When the program runs successfully, include the RANDOMIZE statement in the program before the first occurrence of an RND function if you desire different starting points in the sequence.

Examples

This program prints a different value each time it is run:

```
* 10 RANDOMIZE
* 20 FOR I=1 TO 3
* 30      PRINT RND(0);
* 35      PRINT
* 40 NEXT I
* RUN
.619604
.298047
.698036

END AT 0040
* RUN
.776468
7.84348E-02
.603916

END AT 0040
* RUN
.784302
.117651
.800002

END AT 0040
* RUN
.956853
.98038
9.41276E-02

END AT 0040
* RUN
.10981
.760783
6.31819E-06

END AT 0040
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

READ

Reads values from DATA statements and assigns them to variables.

Format

READ { var / svar } [{ ,var / ,svar }] ...

Arguments

var, svar Numeric and string variables separated by commas

Remarks

1. Always use READ statements in conjunction with DATA statements.
2. The variables listed in the READ statement may be subscripted or nonsubscripted, numeric or string.
3. The order in which variables appear in the READ statement is the order in which the system retrieves values for the variables from the DATA list.
4. The system moves a data element pointer to the next available value in the DATA list as values are retrieved for variables in READ statements. If the number of variables in the READ statement exceeds the number of values in the DATA list, BASIC prints an END OF DATA error message.
5. The type of variable (numeric or string) in the READ statement must match the type of the corresponding DATA value, or BASIC prints a READ/DATA TYPES error message.
6. You can use the RESTORE statement to reset the data element pointer to the first item of the lowest numbered DATA statement or to the first item of a specific DATA statement.

Examples

```
* LIST
0010 READ A,B,C
0020 READ D(1),D(2),D(3)
0030 PRINT C ^ 2,D(2) ^2
0040 READ E
0050 PRINT E
0060 READ F$
0070 PRINT F$
0080 DATA 1,2,3,4,5,6,7,"ABC"
0090 END
* RUN
9             25
7
ABC

END AT 0090
*
```

In this example, the values are assigned to the variables as follows:

Variable	Value
A	1
B	2
C	3
D(1)	4
D(2)	5
D(3)	6
E	7
F$	ABC

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

READ FILE

Reads data in binary format from a sequentially or randomly accessed file.

Format

READ {FILE / #} {(file) / (file, record)} , {var / svar} [, {var / svar}] ...

Arguments

#	An abbreviation for the keyword FILE
file	A numeric expression that evaluates to the number of a file opened for random or sequential access
record	A numeric expression that evaluates to the number of a record in a file opened for random access
var, svar	Numeric variables and string variables that are assigned values read sequentially from a randomly or sequentially accessed record

Remarks

1. The type of variable in the READ FILE variable list must correspond to the data type of the corresponding data item being read from the record.
2. The number of the first record in a random-access file is 0.
3. In random-access files, records that have not been written to contain all 0s when read. An attempt to read a record that is after the last record written causes an end-of-file condition. You can use RESET FILE to continue processing.
4. BASIC sets the EOF(X) function if you try to read a record with a higher number than any already written to the file. If the next READ FILE is successful, BASIC resets EOF(X) to 0. However, if you attempt to read consecutively from a record with a higher number than any in the file, an error occurs.
5. You may use the EOF function to detect an end-of-file condition in the file that is being read.

Examples

```
* LIST
0001 REM READ FILE
0005 TAB =10
0010 DIM B(3,4)
0020 OPEN FILE(1,0),"TESTFILE",20
0030 FOR I=1 TO 12
0040      LET I1=INT((I-1)/4)+1
0050      LET J1=I-(4*(I1-1))
0060      READ FILE(1,I),B(I1,J1)
0070 NEXT I
0080 MAT PRINT B
0090 CLOSE
* RUN
 36   33   30   27
 24   21   18   15
 12    9    6    3

END AT 0090
*
```

Note: This program uses the file TESTFILE, which was created in the program example provided with the WRITE FILE statement.

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

RELEASE

Prevents further I/O access to a previously initialized directory or device.

Format

RELEASE name

Argument

name The name of a directory or a device expressed as either a string variable or a string literal

Remarks

1. RELEASE must be executed from the master terminal. In a single-user system, the master terminal and user terminal are the same. In multiuser environments, programs using RELEASE as a statement do not work when run from terminals other than the master terminal.
2. Use this BASIC version of the CLI command primarily to release mag tape and cassette units. When you release these devices, the system automatically rewinds tapes mounted on them (see INIT).
3. You may also release directories and disks; you must release a disk before removing any disk pack.
4. Release partitions and subdirectories in the order they are nested.
5. You do not need to release line printers or card readers.

Examples

```
* DIR "DP1:JOE:MARY"
* RELEASE "MARY"
* RELEASE "JOE"
* RELEASE "DP1"
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	
F	

REM

Inserts explanatory remarks within a program.

Format

REM *[message]*

Argument

message Text comment

Remarks

1. REM statements do not affect program execution. BASIC stores them with a program and outputs them with each listing.
2. If control is transferred to a REM statement from a GOTO or GOSUB statement, execution continues with the next executable statement. If no executable statement follows the REM statement, the program ends, and control returns to interactive mode.
3. You can add trailing comments to any BASIC statement by using a comment sign (!) (see second example).

Examples

```
* LIST
0010 REM REMARKS IN A PROGRAM.
0020 REM HELPS EXPLAIN THE PURPOSE OF
0030 REM STATEMENTS. LINES 10, 20, 30
0040 REM AND 40 AREN'T EXECUTED.
0050 PRINT "END"
* RUN
END

END AT 0050

* LIST
0010 LET P=61.9      ! P IS THE PRICE
0020                 ! IN CENTS PER UNIT.
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

RENAME

Renames a file in your directory.

Format

RENAME "oldfilename", "newfilename"

Arguments

oldfilename A string literal or string variable that identifies a disk file in your directory

newfilename A string literal or string variable that identifies a new filename

Remarks

1. BASIC searches your directory for the "old" file; if the system finds it, BASIC changes its name to the new filename.
2. BASIC prints an error message at your terminal if:
   - The old filename does not exist.
   - The new filename already exists.
   - The old filename is attribute-protected.

Example

```
* RENAME "TEST.SR", "A.SR"
*
```

File TEST.SR is renamed A.SR. Any future references to TEST.SR will fail.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	✓
F	

RENUMBER

Renumbers the statements in the current program.

Format

RENUMBER [(*line n1* { STEP , } *n2*)]

Arguments

line n1 The line number in the current program where renumbering is to begin

n2 The new increment between line numbers

Remarks

1. The variations of RENUMBER act as follows:

RENUMBER	Renumber the current program starting with default line number 0010, with a default increment of 10 between line numbers.
RENUMBER n1	Renumber the current program starting with line number n1 and by incrementing line numbers by n1.
RENUMBER STEP n2	Renumber the current program starting with default line number 0010 and incrementing line numbers by n2. You may use a comma instead of STEP.
RENUMBER n1 STEP n2	Renumber the current program starting with line number n1 and incrementing line numbers by n2. You may use a comma instead of STEP.

2. Line numbers are limited to four digits. If a RENUMBER command causes a line number to exceed 9999, BASIC re-executes the command as:

   RENUMBER 1 STEP 1

3. The RENUMBER command modifies the line numbers specified in IF-THEN, GOTO, and GOSUB statements to agree with the new line numbers.
4. BASIC changes to 0000 line numbers that cannot be resolved and then prints an error message.
5. The RENUMBER command does not renumber the arguments of ERASE and CHAIN statements.

Examples

```
* LIST
0010 TAB =5
0015 DIM A(3,4)
0020 LET A(1,2)=6
0025 LET A(3,4)=10
0030 MAT PRINT A
0035 MAT A=ZER(3,3)
0037 PRINT
0040 MAT PRINT A
* RENUMBER 10 STEP 5
* LIST
0010 TAB =5
0015 DIM A(3,4)
0020 LET A(1,2)=6
0025 LET A(3,4)=10
0030 MAT PRINT A
0035 MAT A=ZER(3,3)
0040 PRINT
0045 MAT PRINT A
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

RESET FILE

Positions the file pointer to the beginning of a file.

Format

RESET [{ FILE / # } (file)]

Arguments

A synonym for the keyword FILE

file A numeric expression that evaluates to a file number previously associated with an OPEN FILE statement

Remarks

1. You can use the RESET FILE statement to reset the position of the file pointer to the beginning of a file without having to close and reopen the file.
2. Using RESET without an argument repositions the file pointers for all open files in your program to the beginning of their files.

RESET FILE (continued)

Examples

```
* ENTER "RESET"
* LIST
0010 DIM B(3,4)
0020 OPEN FILE (1,0),"TESTFILE",20
0030 FOR I=1 TO 12
0040      LET I1=INT((I-1)/4)+1
0050      LET J1=I-(4*(I1-1))
0060      READ FILE (1,I),B(I1,J1)
0065 NEXT I
0070 GPOS FILE (1),B1
0080 PRINT B1;
0086 RESET FILE (1)
0087 GPOS FILE (1),B1
0088 PRINT B1
0090 CLOSE
0095 PRINT
0100 MAT PRINT B
.
.
.
* LIST "RESET"
TYPE CR TO DELETE OLD:
* RUN
248 0

 36   33   30   27
 24   21   18   15
 12    9    6    3

END AT 0100
* AUDIT
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

RESTORE

Resets the position of the data element pointer.

Format

RESTORE *[line no.]*

Argument

line no. A DATA statement line number

Remarks

1. If you use RESTORE without a line number argument, the system resets the data element pointer to the beginning of the data list.
2. If you use RESTORE with a DATA statement line number argument, the system moves the data element pointer to the first value in the DATA statement line.
3. If the line number argument is not a DATA statement, the data element pointer will point to the first DATA statement following that line number. If the line number does not exist in the program, an error occurs.

Example

```
* 05 READ A,B,C
* 10 READ D,E,F
* 15 RESTORE 50
* 20 READ G,H,I
* 25 RESTORE
* 30 READ J,K,L
* 40 DATA 2,4,6
* 50 DATA 8,10,12
```

In the above example the value assigned to the variables are as follows:

Variable	Values	Variable	Value
A	2	G	8
B	4	H	10
C	6	I	12
D	8	J	2
E	10	K	4
F	12	L	6

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	
F	

RETRY

Repeats a statement that causes an error.

Format

RETRY

Remarks

1. You can use the RETRY statement in conjunction with the ON ERR statement to return control to the statement causing the error, and then attempt to re-execute that statement.
2. For the RETRY statement to work properly, an error condition must have occurred. If no error condition has occurred, a line-number error results.

Example

```
* 005 ON ERR THEN 100
* 010 OPEN FILE (0,2), "TEST"
```

(If statement 10 causes an error, RETRY directs the program to repeat the statement.)

```
.
.
.
* 100 RETRY
```

Note: If statement 10 causes an error, the program will loop indefinitely between statements 10 and 100. The program should, therefore, include some provision for exiting from the RETRY statement after a certain number of failures.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	
F	

RETURN

Returns program control to the statement following the last GOSUB statement executed.

Format

RETURN

Remarks

See "GOSUB and RETURN."

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	
F	✓

RND(X)

Produces a pseudorandom number n, such that 0 <= n < 1.

Format

RND(expr)

Argument

expr A numeric expression, required but not used

Remarks

1. The RND function requires a numeric argument (expr), although the argument does not affect operation of the function.
2. Each time the RND function is called, it provides a pseudorandom number *n*, such that $0 <= n < 1$. The sequence of these numbers is fixed. The sequence repeats starting at the 58384th value. (The 58383rd value is 0.) The sequence is the same for all systems, but the values in a double-precision system are output to more significant digits.
3. Each occurrence of the RND function in a program yields the value of the next random number in the list.
4. Each time you issue a NEW, CHAIN, or RUN, BASIC returns to its original starting place in the sequence of random numbers. Because the sequence is fixed and the starting place is the same for each RUN, the RND function provides the same numbers each time you execute your program. The capability of reproducing the sequence can be a useful debugging aid.
5. To alter the starting place in the sequence, use the RANDOMIZE statement. RANDOMIZE resets the starting place based on the time of day.

Examples

```
* LIST
0005 TAB =13
0010 FOR I=1 TO 4
0020      PRINT RND(I)
0030 NEXT I
* RUN
.21176298
.26666685
.5411776
.90979748

END AT 0030
*
```

Running the above program a second time will produce the same five random numbers.

```
* LIST
0001 RANDOMIZE
0005 TAB =13
0010 FOR J=1 TO 4
0020      PRINT INT(10*RND(I))
0030 NEXT J
* RUN
2
2
5
9

END AT 0030
*
```

This program produces four random integers in the range 0-9. Each time the program runs, it generates different random integers.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	✓
F	

RUN

Executes a program either from the first line or from a specified line.

Format

RUN [{ *line no.* / *"filename"* }]

Arguments

line no. The line in the current program from which execution is to begin

filename The name of a disk file or device

Remarks

The variations of the RUN command act as follows:

RUN	Clear all variables; undimension all arrays and strings; do a RESTORE; initialize the random number generator; and then run the current program from the first line number.
RUN n	RUN from line n. This form of the RUN command allows program execution to resume while retaining current values of all variables and parameters. You can use it after a STOP or after an error, and can incorporate any alterations you make to the program after the STOP or error occurred.
RUN "filename"	If the file is on disk, BASIC first searches your directory and then the library directory for the filename. When it finds the filename, the command executes a NEW, clearing the current program area, and then loads and executes program identified by the filename.

Examples

```
* RUN
* RUN "$PTR"
* RUN 250
* RUN "MATH3"
* RUN "MT1:0"
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

SAVE

Writes the current program and data in binary format to the specified device or disk file.

Format

SAVE "filename"

Argument

filename The name of a disk file or a device, expressed as a string literal or string variable

Remarks

1. If you specify a disk filename, the system checks to see if the file already exists in your directory. If it does, and you have entered SAVE as an immediate command, BASIC prints the following message:

   TYPE CR TO DELETE OLD: (RDOS/DOS)

   TYPE NL TO DELETE OLD: (AOS,AOS/VS)

   This message lets you confirm whether or not the existing file is to be deleted and replaced by the file named in the SAVE command. If you press CR/NEW LINE, BASIC accepts the replacement. If you type anything preceding CR/NEW LINE, BASIC cancels the SAVE command.

2. You can use LOAD, CHAIN, or RUN on a saved program.

3. When you save a program, BASIC stores the current values of all variables with the program, as well as the point where the program stopped last. Therefore, you can load the saved program and use CON or RUN *line no.* as if no interruption had occurred. Note that file status is not preserved when you save a program.

4. A saved program may not run under all configurations of BASIC. In particular, if the precision of the floating-point representation in the RUN environment differs from that of the SAVE environment, you can not load the program.

Examples

Commands:

```
* SAVE "FA.BC"
* SAVE "$PTP"
* SAVE S$(1,7)
```

Statements:

```
* 10 SAVE"OURSHIP"
* 20 SAVE B$
```

AOS, AOS/VS	✓
RDOS, DOS	

S	✓
C	✓
F	

SEARCHLIST

Displays, sets, or stores the searchlist setting. The current searchlist specifies which directories, in addition to the working directory, are searched for file references, other than DELETE.

Format

SEARCHLIST { *[pathname]* / pathname *[,pathname...]* }

Arguments

pathname Any valid directory pathname, or group of pathnames, expressed as a string variable or a string literal

Remarks

1. SEARCHLIST with no arguments displays the current searchlist.
2. Used with arguments, SEARCHLIST sets the searchlist to the directories specified by the pathname arguments.
3. A single pathname argument, whether a string variable or literal, can contain one or more directory pathnames, separated by commas.
4. BASIC on AOS and AOS/VS allows 0 to 7 pathnames in a searchlist setting.
5. If the pathname argument is a single string variable with a current length of 0, BASIC returns the current searchlist setting and stores it at the variable specified by the pathname argument.

Examples

Set the searchlist with a string literal:

```
* SEARCHLIST ":UDD,:UTIL",":BASIC"
```

Display the current searchlist:

```
* SEARCHLIST
:UDD,:UTIL,:BASIC
```

Use a variable to set the searchlist:

```
* A$=":UTIL,:"
* SEARCHLIST A$
* SEARCHLIST
:UTIL,:
```

Set the searchlist using both variables and literals:

```
* SEARCHLIST ":UDD:BASIC,:HELP",":BASIC",A$
* SEARCHLIST
:UDD:BASIC,:HELP,:BASIC,:UTIL,:
```

Store the searchlist in a variable:

```
* SEARCHLIST ":UTIL,:"
* A$=""
* SEARCHLIST A$
* PRINT A$
:UTIL,:
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	
F	✓

SGN(X)

Returns a value that represents the sign of an expression.

Format

SGN(expr)

Argument

expr A numeric expression

Remarks

The value returned is:

1 if positive
0 if 0
-1 if negative

Example

```
* LIST
0010 LET A=-3
0020 PRINT SGN(A)
* RUN
-1

END AT 0020
*
```

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

SHARE

Adds the resolution file attribute H (sharable) to a file that already exists in your directory.

Format

SHARE "filename"

Argument

filename A disk file in your directory, expressed as a string literal or string variable

Remarks

1. SHARE is equivalent to CHATR *filename*,+0.
2. Any attempt to share an open file generates an error message.

Example

```
* WHATS "COMMON"
COMMON. DP ...

* SHARE "COMMON"
COMMON. DPH ...
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	
F	✓

SIN(X)

Calculates the sine of an angle that is expressed in radians.

Format

SIN(expr)

Argument

expr A numeric expression specified in radians

Remarks

SYS(15) is assigned the value of pi (3.1416). See the SYS(X) function for more information.

Example

```
* LIST
0010 REM - PRINT SINE OF 30 DEGREES
0020 PRINT SIN(30*SYS(15)/180)
* RUN
 .5

END AT 0020
*
```

AOS, AOS/VS	✓
RDOS, DOS	

S	✓
C	✓
F	

SIZE

Prints the number of bytes and pages used by the program, and the total number of bytes and pages that are still available.

Format

SIZE

Remarks

1. BASIC breaks down the number of bytes used into two groups: the number of bytes used for the program segment (P), and the number of bytes used for the data segment (D).
2. The system reports the number of pages used and pages left. One page equals 2048 bytes.
3. BASIC reports the total number of bytes left.

Example

```
* SIZE
USED: 0(P), 0(D) BYTES    1(P), 1(D) PAGE(S)
LEFT: 40658 BYTES         18 PAGE(S)
```

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

SIZE

Prints the number of bytes and pages used by the program, and the total number of bytes and pages that are still available.

Format

SIZE

Remarks

1. BASIC breaks down the number of bytes used into two groups; the number of bytes used for the program segment (P), and the number of bytes used for the data segment (D).
2. If the RDOS/DOS Extended BASIC system uses swapping or extended memory, the system reports the number of pages used and left. One page equals 512 bytes.
3. BASIC reports the total number of bytes left.

Examples

On a swapping system:

```
* SIZE
USED: 14850 (P), 312 (D) BYTES 8 (P), 1(D) PAGE(S)
LEFT: 29594 BYTES       13 PAGE(S)
```

On a nonswapping system:

```
* SIZE
USED: 3793 (P), 2821 (D) BYTES
LEFT: 8077 BYTES
*
```

From a total of 14,691 bytes of memory available for program and data storage, the program occupies 3793 bytes, data occupies 2821 bytes, and 8077 bytes remain unused.

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

SPOS FILE

Moves the file pointer to the byte position specified by expr.

Format

SPOS { # / FILE } (file),expr

Arguments

A synonym for the keyword FILE

file A numeric expression that evaluates to a file number previously associated with an OPEN FILE statement

expr A numeric expression that evaluates to the number of a byte position in a file

Remarks

SPOS FILE moves the file pointer to a byte position, and not necessarily to a record position. Therefore, you must be certain that the value of the expression argument is indeed the calculated position at which you intend to place the file pointer.

Examples

```
* 100 SPOS FILE (1), I+132
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	
F	✓

SQR(X)

Computes the square root of an expression.

Format

SQR(expr)

Argument

expr A nonnegative numeric expression

Example

```
* LIST
0010 LET A=5
0020 PRINT SQR(A↑2+75)
* RUN
10

END AT 0020
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	
F	

STOP

Terminates execution of the current program and returns control to interactive mode.

Format

STOP

Remarks

1. You can place STOP statements anywhere in the program to terminate execution. When BASIC encounters STOP, it prints the following message on your terminal, where xxxx is the line number of the STOP statement:

   STOP AT xxxx

2. After resumption of interactive mode, you can modify the program if you wish. To restart the program from the beginning, use RUN; to continue from the STOP statement, use CON or RUN *line no.*

Example

```
* LIST
0010 REM--TERMINATE PROGRAM BY STOP
0020 INPUT A
0030 IF A < 0 THEN GOTO 0050
0040 GOTO 0020
0050 STOP
* RUN
? 1
? 3
? -5

STOP AT 0050
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	
F	✓

STR$(X)

Converts the numeric value of an expression to a string.

Format

STR$(expr)

Argument

expr A numeric expression

Remarks

1. Converting numerics to strings with no leading or trailing spaces permits string manipulation by other string-handling functions and statements.
2. This function is useful for combining numbers when you do not want spaces between them.

Example

```
* LIST
0010 READ A
0015 IF A=0 THEN STOP
0020 LET A$=STR$(A)
0030 IF A$(4,6)="222" THEN GOTO 0050
0040 GOTO 0070
0050 PRINT A;" -THIS IS MODEL 222"
0060 GOTO 0010
0070 PRINT A;" -THIS ISN'T OUR MODEL"
0080 DATA 111222,212222,123456,0
0090 GOTO 0010
* RUN
111222 -THIS IS MODEL 222
212222 -THIS IS MODEL 222
123456 -THIS ISN'T OUR MODEL

STOP AT 0015
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	
F	✓

SYS(X)

Returns system information based on the value of an expression that is evaluated to an integer from 0 to 19.

Format

SYS(expr)

Argument

expr A numeric value or expression, between 0 and 19

Remarks

The values returned by the SYS function are listed below:

SYS(0)	The time of day in seconds after midnight
SYS(1)	The day of the month (1 to 31)
SYS(2)	The month of the year (1 to 12)
SYS(3)	The year in four digits (e.g., 1983)
SYS(4)	The multiplexor line number (-1 if it is the CLI console in RDOS, the console number in AOS.)
SYS(5)	CPU time used, in seconds to the nearest tenth
SYS(6)	I/O usage (numbers of file I/O statements executed)
SYS(7)	The error code of the last runtime error
SYS(8)	The number of the file most recently referred to in a file I/O statement
SYS(9)	Page size
SYS(10)	Tab size
SYS(11)	Hours }
SYS(12)	Minutes } current time of day
SYS(13)	Seconds }
SYS(14)	Seconds remaining before expiration of timed input
SYS(15)	pi (3.14159)
SYS(16)	e (2.71828)
SYS(17)	1/10 second clock (not applicable to AOS)
SYS(18)	Total number of BASIC I/O calls (ENTER, LIST, etc.)
SYS(19)	Line number of the last error

Example

```
* PRINT SYS (0)
63736
* PRINT SYS(1);SYS(2);SYS(3)
31 10 1983
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

TAB

Sets the zone spacing between the data output by PRINT statements.

Format

TAB=expr

Argument

expr An arithmetic expression in the range: 1<= expr <= page width (see PAGE)

Remarks

1. The default zone spacing is 14 columns.
2. Since the maximum range of zone spacing depends upon the PAGE command setting, it is good practice to set the page width first and then the zone spacing.

Example

```
* LIST
0010 PAGE =50
0020 TAB =10
0030 FOR I=1 TO 25
0040      PRINT I,
0050 NEXT I
* RUN
 1    2    3    4    5
 6    7    8    9   10
11   12   13   14   15
16   17   18   19   20
21   22   23   24   25

END AT 0050
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	
F	✓

TAB(X)

Tabulates to the column number set by the expression.

Format

TAB(expr)

Argument

expr An expression that is evaluated to an integer

Remarks

1. Use the TAB function only in conjunction with PRINT statements. You cannot use it with any other BASIC statement. More than one TAB(X) function may appear in a PRINT statement. The system prints the next item in the print list at position X.
2. The first column on a line is column 1. The column number specified by the expression is always relative to column 1. The position at which BASIC prints an item in the print list depends on the value of the expression and on the PRINT statement punctuation (; or ,) following the TAB(X) function.
3. If the expression evaluates to a column number less than the present column number, printing proceeds at that position on the next line.
4. If the expression evaluates to a column number greater than the page length, the expression is reduced modulo the page length and positioning proceeds as in 2.
5. The TAB(X) function must assume that each character output on the same line prior to the TAB(X) occupies one space on the output line. If characters that violate this rule are used—such as the tab character (<9>), new line (<10>), or nonprinting control characters—you must compensate for the missing or extra character positions by adjusting the argument of the TAB(X) function.

TAB(X) (continued)

For example, in

```
* PRINT "AB<10>CD";
* PRINT TAB(8);"EF"
```

even though there is a new line imbedded in the first string literal, TAB(8) generates two blanks instead of five so that the output is

AB
CD□□EF

instead of

AB
CD□□□□□EF

Example

```
* LIST
0005 LET A=-6
0010 LET B=5
0015 PRINT TAB(B);A; TAB(2*B);2*A
0020 END
* RUN
□□□□□-6 □□□□□-12

END AT 0020
*
```

Note that in line 15 the semicolon after A prevents spacing to the next print zone and passing position 2*B (column 10).

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	
F	✓

TAN(X)

Calculates the tangent of an angle that is expressed in radians.

Format

TAN(expr)

Argument

expr A numeric expression specified in radians

Example

```
* LIST
0010 REM - PRINT TANGENT OF X DEGREES
0020 INPUT "X DEGREES ",X
0030 LET P=SYS(15)/180
0040 PRINT TAN(X*P)
* RUN
X DEGREES 45
 1

END AT 0040
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

TIME

Establishes the time limit for timed input (TINPUT) operation.

Format

TIME=expr

Argument

expr A numeric expression that represents time in seconds, to the nearest tenth of a second. In AOS, the maximum number is approximately 4,000,000. In RDOS/DOS, the maximum is approximately 65,000.

Remarks

1. Assigning a value to TIME sets the SYS(14) function to the value of expr.
2. The value of SYS(14) is decremented at the clock tick rate (1/10 of a second per tick) from the time a TINPUT statement is executed.
3. Decrementing of SYS(14) stops when you respond to the TINPUT prompt. Decrementing of SYS(14) is resumed when the next TINPUT is executed.
4. If you do not respond to the TINPUT prompt before the SYS(14) function has decremented to 0, the system prints an error message and the program stops (unless an ON ERR THEN statement is executed in your program).
5. You can reset TIME to another value; it appears as often as the program logic requires.

Examples

```
* LIST
0010 DIM A$(50)
0020 PRINT "LET'S TEST YOUR RECALL SPEED"
0030 PRINT
0040 TIME = 10
0050 TNPUT "WHAT COLOR ARE YOUR
     MOTHER'S EYES?", A$
0060 GOSUB 0140
0070 TINPUT "WHAT'S YOUR SOCIAL SECURITY
     NUMBER?", A$
0080 GOSUB 0140
0090 TINPUT "HOW OLD IS YOUR FATHER?", A$
0100 GOSUB 0140)
0130 GOTO 0190
0140 LET I = I + 1
0150 LET A(I) = (10-SYS(14))
0160 PRINT "TIME USED-",A(I); "SECONDS"
0170 TIME = 10
0180 RETURN
0190 FOR J = 1 TO I
0200     LET B = B + A(J)
0210 NEXT J
0220 LET C = B/I
0230 PRINT "AVERAGE RESPONSE TIME= ";
     C;"SECONDS"
0240 END
*RUN
LET'S TEST YOUR RECALL SPEED

WHAT COLOR ARE YOUR MOTHER'S EYES? BROWN
TIME USED - 6.8 SECONDS
WHAT'S YOUR SOCIAL SECUTIRY NUMBER? 11234567
TIME USED - 9.2 SECONDS
HOW OLD IS YOUR FATHER? 65
TIME USED - 5.5 SECONDS
AVERAGE RESPONSE TIME = 7.1666667 SECONDS

END AT 0240
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

TINPUT

Assigns to a list of variables, within a prescribed time, the values supplied by input from the terminal.

Format

TINPUT*[(line no.[,time]),]["str lit",]* {var / svar} [{,var / ,svar}]...*[;]*

Arguments

line no. A program line number

time A numeric expression that evaluates to an integer representing time, in seconds, to the nearest tenth of a second

var, svar Variables separated by commas, or carriage returns or new lines

str lit A message or prompt

Remarks

1. The remarks for INPUT apply to TINPUT.
2. Use TINPUT with the (line no. [,time],) arguments or in conjunction with TIME and the SYS(14) function. TIME sets SYS(14) to the value, in seconds, allowed for your response to the TINPUT prompt.
3. If TINPUT has a line number argument and you do not respond to its prompt before the SYS(14) function decrements to 0, the program branches to that statement number. If TINPUT has no line number argument, and SYS(14) decrements to 0, BASIC prints an error message, and the program stops (unless an ON ERR THEN statement in the program has previously executed).
4. A TINPUT time argument only applies to that specific TINPUT statement; it does not affect SYS(14). This argument only sets a time limit for the response; it does not determine the amount of time taken to respond.

Example

```
0005 ON ERR GOTO 300
0010 TINPUT (100, 25), "ENTER:", A$
```

The program branches to line 100 if there is no response to the ENTER prompt within 25 seconds. The line number argument overrides the ON ERR statement.

AOS, AOS/VS	✓
RDOS, DOS	

S	✓
C	✓
F	

UNLOCK

Frees locked areas for use.

Format

UNLOCK *[iden]*

Argument

iden A numeric expression that identifies a specific lock or group of locks in a program

Remarks

1. An UNLOCK on a particular identifier frees all file areas associated with that identifier. UNLOCK with no identifier unlocks all areas previously locked by the user (see LOCK).
2. UNLOCK is always successful. Specifying an unknown identifier causes UNLOCK to be ignored.
3. Whenever BASIC terminates, it unlocks all locks. The NEW command also unlocks all areas currently locked by the user.

Example

```
* 10 LOCK 1,"INVENTORY",0,128,A
```
This locks a 128-byte record.

```
* 20 UNLOCK 1
```
This releases the lock.

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

UNSHARE

Removes the resolution file attribute H (sharable) from a file that already exists in your directory.

Format

UNSHARE "filename"

Argument

filename A disk file in your directory, expressed as a string literal or a string variable

Remarks

1. UNSHARE is equivalent to CHATR *filename*, -0.
2. Any attempt to use UNSHARE on an open file generates an error message.

Example

```
* WHATS "COMMON"
COMMON DPH...
* UNSHARE "COMMON"
COMMON DP...
```

AOS, AOS/VS	
RDOS, DOS	✓

S	✓
C	✓
F	

USERS

Prints a status report of all active users.

Format

USERS *[userID]*

Argument

userID A user identification

Remarks

1. When you include the optional userID, BASIC prints the status of that user only.
2. The status report contains the following information:

USERS (continued)

Heading	Meaning
NAME	Four-character user identification; OPER is the system manager's identification name.
STATUS	Four hexadecimal digits. Four 0s mean idle terminal.

Hexadecimal	Meaning
1	RUN equals on, EDIT equals off.
2	CLI console doing [MAT] INPUT (does not echo carriage return).
4	Swap in progress or swapped out
8	Program storage has been changed.
10	Do not allow messages.
20	Execute-only file is in use.
80	BYE in progress
400	Timed input in progress
800	No subdirectory is associated with this user.
1000	Terminal input in progress
2000	Delay in progress
8000	Special mark-sense card translation

For example, 000D means:

0008	Program storage has been changed.
0004	Swap in progress or swapped out
0001	Run mode
000D	

Heading	Meaning
DIRECTORY	The name of the directory assigned to NAME, from the BASIC.ID file
LINE	QTY or ALM line number (0 to 31). Master terminal is line -1.
SWAPS	The number of accesses to the BASIC swapping file a user has made.
DATA	Two numbers. The first number is the core block address of the data; the second number is the number of data blocks in use for this user. An ellipsis (...) indicates BASIC has swapped the data out of main memory.
PROG	Two numbers. The first number is the core block address of the program; the second number is the number of program blocks in use for this user. An ellipsis (...) indicates BASIC has swapped the program out of main memory.
PRI	The current priority level is set by LEVEL.
OVLY	The number assigned to the current overlay in the queue or execute area. -1 means no overlay is in use.
FILES	Number of files opened at this time by this user
CPU	Number of seconds of CPU time used by this user

Example

NAME	STATUS	DIRECTORY	LINE	SWAPS	DATA	PROG	PRI	OVLY	FILES	CPU
0000	0041	USER0	0	9	32	4	25	-1	0	3.7
TREK	104C	TREK	1	1	4	4	25	-1	1	.1
JERR	0041	JERRY	2	7	32	4	25	-1	0	1.8
OPER	0044	BASIC	3	5	32	4	25	9	0	11.7
	0000		4							

AOS, AOS/VS	✓
RDOS, DOS	✓

S	
C	
F	✓

VAL(X$)

Returns the numeric representation of a string value.

Format

VAL { (svar) / ("str lit") }

Arguments

svar, str lit A string beginning with a number

Remarks

1. The string variable or string literal argument to the VAL function must begin with a number, or else the system outputs an error message. The number may include digits, plus and minus signs, decimal points, and the letter E (scientific notation). BASIC ignores any nonnumeric characters that appear after the number portion of the string, for example:

   "+35.5E-03ABCD7N"

   BASIC returns substring "+35.5E-03" as a numeric value and ignores substring "ABCD7N".

2. Misplaced signs terminate the input scan in a similar fashion:

   "123 + 47 - 17"

   BASIC returns substring "123" as a numeric value and ignores "+47-17".

Example

```
* LIST
0010 LET A$="12345ABCD"
0020 LET B=54321
0030 LET C=VAL(A$)
0040 LET D=B+C
0050 PRINT D
* RUN
66666

END AT 0050
*
```

AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

WHATS

Determines the status of a specified file.

Format

WHATS "filename"

Argument

filename The name of a file in your directory or in the library directory, expressed as a string literal

Remarks

1. BASIC searches your directory for the filename; if it is not found, BASIC searches the library directory.
2. The differences in output for the operating systems are shown in the examples.

Examples

RDOS and DOS:

```
* WHATS "ABC"

ABC   P      2039   06/14/82   09:15   (1/8/83)   00
```

filename — *attribute* — *length in bytes* — *date created* — *time created* — *date last used* — *in use count*

AOS and AOS/VS:

```
* WHATS "PHASE4"

PHASE4   UDF   01-MAY-78   16:20:34   690
```

filename — *type* — *date last modified* — *time last modified* — *length in bytes*

AOS, AOS/VS	✓
RDOS, DOS	

S	✓
C	✓
F	

WHO

Identifies other people on the system or determines your own identification.

Format

WHO *processID*
"processname"

Arguments

processID A process identification number

processname A process name, assigned by AOS, expressed as a string variable or string literal

Remarks

1. You can identify other people using the system by using the WHO command with either process identification numbers or process names. BASIC responds to the command by printing both the process identification number and the process name.
2. Without an argument, WHO prints your process identification number and process name on your terminal.

Example

```
* WHO 7
PID: 7 XBASIC:007
*
```

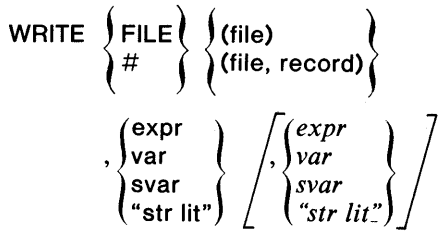
AOS, AOS/VS	✓
RDOS, DOS	✓

S	✓
C	✓
F	

WRITE FILE

Writes a record of data in binary format into a sequential- or random-access file.

Format



Arguments

#	A synonym for the keyword FILE
file	A numeric expression that evaluates to the number of a file opened for random or sequential access
record	A numeric expression that evaluates to the number of a record in a file opened for random access
expr, var, svar, and str lit...	A list of one or more numeric expressions, numeric variables, string variables and literals whose values are written as a record into a sequential or random-access file

Remarks

1. The first record number in a random-access file is 0.
2. READ FILE—but not INPUT FILE statements—can access data files created by WRITE FILE statements.
3. When a string is written, the number of bytes written is equal to the current length of the string plus one byte for a null byte terminator.
4. A single record is the result of a single READ FILE or WRITE FILE statement. When you specify a record size (or use the mode 0 default) in an OPEN FILE statement, the output of a WRITE FILE with fewer than the specified number of bytes is padded with nulls to fit the size of the record. Output having more than the specified number of bytes causes an error. If you specify no record size, WRITE FILE does not pad with nulls.

Examples

```
* LIST
0001  REM-FILE WRITE
0010  DIM A(3,4)
0020  FOR I=1 TO 3
0030    FOR J=1 TO 4
0040      LET A(I,J)=((I-1)*4+J)*3
0050    NEXT J
0060  NEXT I
0070  MAT PRINT A
0080  PRINT
0090  OPEN FILE(1,0),"TESTFILE",20
0100  FOR I1=1 TO 3
0110    LET I=4-I1
0120    FOR J1=1 TO 4
0130      LET J=5-J1
0140      LET R=(3-I)*4+(5-J)
0150      WRITE FILE(1,R),A(I,J)
0160      PRINT A(I,J),
0170    NEXT J1
0180    PRINT
0190  NEXT I1
0200  CLOSE
* RUN
3       6       9       12
15      18      21      24
27      30      33      36

36      33      30      27
24      21      18      15
12      9       6       3


END AT 0200
*
```

End of Chapter

Chapter 4
Calling an Assembly Language Subroutine from Extended BASIC

You can call a subroutine written in assembly language from an Extended BASIC program. The format of the BASIC call is:

CALL sub# [, A_1,..., A_n]

where:

sub#	is a numeric expression evaluating to a positive integer, from 0 to 32767, representing the subroutine number.
A_1,..., A_n	are optional arguments to be passed to the subroutine; *n* must be in the range 1 to 8. They may be arithmetic variables or expressions, or string variables or expressions. Dimensioned numeric variable names should include subscripts. Statement numbers are not permitted as arguments.

Character String Storage and Definitions

You must refer to the following information if you wish to handle character strings in a called subroutine. BASIC keeps a count of the number of characters currently defined in each string variable, referred to as the *current length* of the string variable. BASIC stores the current length as part of the header immediately preceding the contents of each string variable (see Figure 4-1). You must update the current length each time characters are added to or taken away from the string variable.

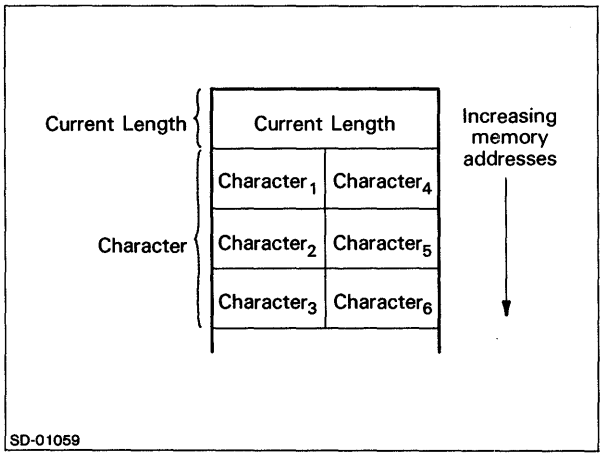


Figure 4-1. String Variable Storage

A *substring* is any contiguous part of a string variable. For example, A$(2,4) and A$ are substrings of A$ (that is, A$ is wholly contained in itself).

The *current length of a substring* is the number of defined characters within the substring. For example, if only A$(4,4) and A$(5,5) are defined, the current length of A$(4,7) is 2.

The *maximum length of a substring* is the number of character positions within the substring. For example, the maximum length of substring A$(4,7) is 4.

Linking the Assembly Language Subroutine

Improper use of assembly language subroutines, system calls, or task calls can crash the system.

You must submit assembly language subroutines to the system manager at system load time. BASIC inputs the subroutines in a file named SBRTB.RB to the relocatable loader when it creates the BASIC system save file. You must include a subroutine table with your subroutines. The table must have the entry point SBRTB.

The subroutine table is a list of all assembly language subroutines available to a BASIC program. For each assembly language subroutine, BASIC requires a four-word list in the table containing the following:

- Subroutine number
- Subroutine entry point
- Number of arguments
- Argument control word

Terminate the table by using a subroutine number of -1.

BASIC uses the argument control word to check runtime errors on the types of arguments. The control word is divided into eight two-bit fields for the eight possible arguments $A_1...A_n$. The value of the two-bit field determines the allowable argument:

00 Argument may be any string expression.
01 Argument must be a string variable.
10 Argument may be any numeric expression.
11 Argument must be a numeric variable.

You must write the argument control word in an assembly language program so that the arguments are connected by a plus sign (+), as described in Figure 4-2.

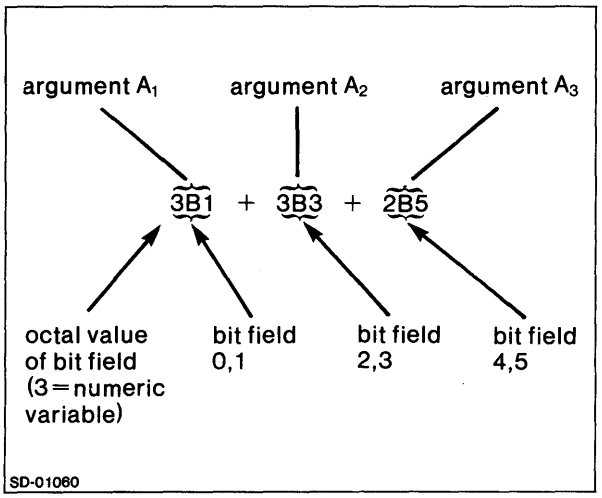


Figure 4-2. Argument Control Word

If you use an argument to return a value to the calling program, you must set the argument's flags to either a 01 (for string) or 11 (for numeric).

BASIC calls the assembly language subroutines by the sequence:

```
        LDA     2,.+2   ;AC2 POINTS TO TOP
                        ;OF ADDRESS LIST
        JMP     <SUB>   ;JMP TO ASSEMBLY
                        ;LANGUAGE SUBROUTINE
        ADLST

ADLST:  <arg A1>
        <arg A2>
          .
          .
          .
        <arg An>
        JMP     BASIC   ;RETURN TO BASIC
                        ;INTERPRETER
```

If A_n is a substring of a string variable or the whole string variable, the address list contains the address of the string descriptor words that contain the following information:

Word 1 Byte address of the first character of the substring

Word 2 Current length of the substring

Word 3 Maximum length of the substring

Word 4 Word address of the current length of the string variable

If A_n is a string expression, the address list contains the address of the string descriptor words that contain the following information:

Word 1 Byte address of the first character of the string

Word 2 Length of the string

If A_n is a numeric variable, the address list contains the storage address of the variable. (BASIC represents all numeric variables in standard floating-point format.)

If A_n is a numeric expression, the address list contains the storage address of the value of the expression.

Figure 4-3 shows calls to a subroutine, the subroutine table, and the subroutine itself. The argument list in a BASIC call to a subroutine must match the argument control word specified in the subroutine table.

An illegal CALL results from an attempt to pass a variable in the CALL that does not have a previously assigned value. You must have assigned values to all variables passed in the CALL even if their current value will not be used in the called subroutine.

Several subroutines are available in BASIC to help you manipulate numbers and character strings. The pointers to the routines are in page 0 and should be declared as displacement externals.

On AOS and AOS/VS, SBRTB.SR should be assembled to produce SBRTB.OB. You should edit the bind command line LOAD.CL (LOADSP.CL for single-precision), to include SBRTB and execute it to incorporate the assembly language routines. The routines .MPY, .MPYA, .DVD, and .DVDI (supported by RDOS Extended BASIC) do not exist in AOS and AOS/VS Extended BASIC, since single ECLIPSE instructions can be used in their place.

RDOS BASIC Multiuser

If you use multiuser RDOS BASIC assembly language subroutines, they must be reentrant or interlocked (see Figure 4-3). To make reentrant programs easier to write, the system saves and restores temporary registers (TR0 to TR15 in page zero) for each user or task switch. You can access these registers as displacement externals (see .MOST, Table 4-1).

If your subroutine performs I/O to or from the user program or data area, you must prevent extended memory remapping during I/O. You can do this if you enter single-task mode, with the S.ING subroutine, before the system call. After you finish your I/O, use the M.ULT subroutine to reenter multitask mode, for example:

```
        JSR         @?SING      ; ENTER SINGLE-TASK MODE
        .SYSTEM                 ; SYSTEM CALL WHICH READS DIRECTLY
        .RDS        77          ; INTO THE USER DATA SEGMENT
        JSR         ERR         ; ERROR RETURN
        JSR         @?MULT      ; GOOD RETURN, RE-ENTER MULTI-TASK MODE
        .
ERR:    JSR         @?MULT      ; RE-ENTER MULTI-TASK MODE IF ERROR
                                ; ERROR PROCESSOR
```

The pointers ?SING and ?MULT are in page zero and should be declared as displacement externals.

The routines in Table 4-1 are helpful when linking your assembly language subroutines. In systems having floating-point hardware, BASIC stores the floating-point number and returns it in the Floating-Point Accumulator (FPAC) rather than in AC0-AC1.

```
                    .TITLE  SBRTB   ; BASIC ASSEMBLY LANGUAGE SUBROUTINES
                    .ENT    SBRTB   ; ENTRY POINT : SBRTB
                    .NREL           ; NORMAL RELOCATABLE CODE


; SUBROUTINE TABLE


SBRTB:      1                       ; SUBROUTINE NUMBER
            SUB1                    ; SUBROUTINE ENTRY POINT
            2                       ; NUMBER OF ARGUMENTS
            3B1 + 3B3               ; ARGUMENT CONTROL WORD, BOTH ARGS ARE
                                    ; NUMERIC VARIABLES
            -1                      ; END OF TABLE


SUB1:
; CALLING SEQUENCE:  CALL 1,A,B
; THIS ROUTINE IS THE EQUIVALENT OF LET B = A
; THIS ROUTINE IS REENTRANT
            STA         2,TRO       ; SAVE ADDRESS LIST
            LDA         3,0,2       ; ADDRESS OF ARG 1
            LDA         3,0,3       ; WORD 1 OF ARG 1
            LDA         2,1,2       ; ADDRESS OF ARG 2
            STA         3,0,2       ; WORD 1 OF ARG 1 TO WORD 1 OF ARG 2
            LDA         2, TRO      ; ADDRESS LIST
            LDA         3,0,2       ; ADDRESS OF ARG 1
            LDA         3,1,3       ; WORD 2 OF ARG 1
            LDA         2,1,2       ; ADDRESS OF ARG 2
            STA         3,1,2       ; WORD 2 OF ARG 1 TO WORD 2 OF ARG 2
            LDA         3,TRO       ; ADDRESS LIST=RETURN ADDRESS -2
            JMP         2,3         ; RETURN TO BASIC (2=NO. OF ARGS)
            .E
```

Figure 4-3. Example of an Assembly Language Subroutine

Table 4-1. Subroutine Conversion Routines

Routines	Result
.FIX	Converts floating-point number in AC0-AC1 (or floating-point AC0 in the case of hardware floating-point support) to an integer in AC0-AC1. If there is overflow, the largest possible integer is returned in AC0-AC1. Bit 0 of AC0 is the sign of the number. Bit 0 of AC1 is a significant bit. There are two returns from .FIX: Return 1: overflow Return 2: OK
.FLOT	Converts an integer in AC0-AC1 to floating-point format in AC0-AC1.
.ADDF F0+F1 .SUBF F0-F1 .MPYF F0*F1 .DIVF F0/F1	Arithmetic routines to perform floating-point add, subtract, multiply, divide. In each routine, AC0-AC1 initially contains the floating-point value of F1 and AC2 contains the address of the value of F0. The result is returned in AC0-AC1. Underflow returns a zero result; overflow results in error number 16.
.MPY A1*A2 A0,A1 .MPYA A0+A1* A2 A0,A1	In the integer multiply routines, AC1 contains the unsigned integer multiplicand and AC2 contains the unsigned integer multiplier. The result is a double length product with high-order bits in AC0 and low-order bits in AC1. Contents of AC2 are unchanged. The difference between the routines is that .MPYA adds the result of the multiplication to the contents of AC0.
.DVD (A0,A1) /A2 A1,A0 .DVDI A1/A2 A1,A0	In the integer divide routines the dividend is an AC1 (single-length) or in AC0 and AC1 (double-length with high-order bits in AC0). The divisor is in AC2 and the result is left with the quotient in AC1 and the remainder in AC0. Contents of AC2 are unchanged.
.MOST	Moves the character string described by the string descriptor words in AC0, AC1 to the substring described by the string descriptor words in the page 0 memory locations labeled TR3, TR4, and TR5. Before a JSR to .MOST, these accumulators and memory locations should be loaded as follows: AC0 Byte address of the first character of the source string AC1 Length of the source string TR3 Byte address of the destination substring TR4 Maximum length of the destination substring TR5 Word address of the current length of the destination string. Bit 0 should be on if the destination string is a substring and you desire blank padding. You should declare TR3, TR4, and TR5 as displacement externals in the assembly language subroutine. .MOST automatically updates the current length of the destination string variable. .MOST returns the address of the next byte of the destination string in TR3 and the number of bytes left to be filled in the destination string in TR4 (not counting blanks padded onto substrings).

End of Chapter

Appendix A
Error Messages

Extended BASIC error messages are printed as two-digit codes, followed by a brief explanatory message. The following categories of errors may occur under Extended BASIC.

1. Errors recognized by BASIC during program input (Table A-1)
   a. If BASIC detects an error in a statement input from a terminal, the error message refers to the last statement typed.
   b. If the statement in error is input from a file or other input device, BASIC prints the incorrect statement followed by the error message.
   c. BASIC recognizes all syntax errors during program input.
   d. The form of the error message is:

      ERROR xx text

      where:

      xx is a two-digit decimal error code.

      text is a brief description of the error.
2. Runtime errors, except file I/O (Table A-1)

   BASIC system runtime errors cause displays of error messages in the following form:

   ERROR *[xx AT yyyy]* text

   where:

   xx is a two-digit decimal error code.

   yyyy is the line number at which the error occurred, if used in a statement.

   text is a brief description of the error.
3. RDOS/DOS Extended BASIC File I/O errors (Table A-2)

   The format for file I/O error messages is as follows:

   I/O ERROR xx *[AT yyyy]* text

   where:

   xx is a two-digit decimal error code.

   yyyy is the line number at which the file I/O error occurred, if used in a statement.
4. AOS and AOS/VS Extended BASIC File I/O errors. See the AOS or AOS/VS *Programmer's Manual* for I/O error messages.

Table A-1. BASIC Error Messages (continues)

Code	AOS Text	RDOS/DOS Text	Meaning
00	Invalid operator	Format	Unrecognizable statement format
01	Character not recognized	Character	Illegal ASCII character or unexpected character
02	Illegal statement syntax	Syntax	Invalid syntax or argument type
03	Data types don't match	READ/DATA types	READ specifies a different data type than DATA statement.
04	Hardware or software fault	System	Hardware or software malfunction
05	Missing or illegal line number	Line number	Statement number not in the range $1 <= n <= 9999$
06	348 variables have already been defined	Excessive variables	Attempt to declare too many variables
07	Keyword not valid as command	Command	Attempt to execute an illegal command
08	Singular matrix - cannot be inverted	Singular matrix	Attempt to invert a singular matrix
09	File cannot be loaded - wrong revision	(Not used)	Core image file incompatible with system
10	Illegal attribute	Attribute	Attempt to assign an illegal attribute to a file
11	Unmatched parenthesis	Parenthesis	Parentheses in an expression are not paired properly.
12	Mantissa overflow	(Not used)	Hardware or software malfunction
13	Arithmetic underflow	(Not used)	Result of arithmetic expression is too small.
14	Program overflow	Pgm ovfl	Not enough storage to ENTER source program
15	End of DATA	End of data	Not enough DATA arguments to satisy READ
16	Arithmetic	Arithmetic	Value too large or too small to evaluate, or a divide by 0
17	Arithmetic overflow	(Not used)	Result of arithmetic expression too large
18	Gosub nesting	GOSUB nesting	More nested GOSUBs than specified at BASIC system generation
19	RETURN - no GOSUB	RETURN - no GOSUB	RETURN statement encountered without a corresponding GOSUB
20	FOR nesting	FOR nesting	More nested FORs than specified at BASIC system generation
21	FOR - no NEXT	FOR - no NEXT	Unexecutable FOR-NEXT loop; FOR without a NEXT

Table A-1. BASIC Error Messages (continues)

Code	AOS Text	RDOS/DOS Text	Meaning
22	NEXT - no FOR	NEXT - no FOR	NEXT statement encountered without a corresponding FOR
23	Data overflow	Data ovfl	Not enough storage left to assign space for variables
24	Attempt to divide by zero		Attempt to divide by 0
24		Directory empty	No files in your directory
25	Feature not available	Option	Feature specified not available
26	(Not used)	(Not used)	
27	Illegal file number	File number	Invalid file designation in an I/O statement
28	Upward re-dimension	DIM ovfl	An array or string exceeds its original dimensions.
29	Expression is too complex for evaluation	Expression	An expression is too complex for evaluation.
30	Illegal file mode	MODE	Invalid mode designation in an I/O statement
31	Subscript out of bounds	Subscript	Subscript exceeds array's dimensions.
32	Undefined function	Undefined function	Attempt to use a function never defined by DEF
33	Function nesting	Function nesting	User function nesting exceeds BASIC system generation specification.
34	Function argument	Function argument	Argument range out of bounds
35	Illegal mask	Illegal mask	PRINT USING format is illegal.
36	Cannot execute commands now	No commands now	An ENTER file has a command instead of a statement.
37	User routine not found	User routine	CALL statement specifies a user routine not in storage.
38	(Not used)	(Not used)	
39	Duplicate matrix	Dup matrix	Same matrix appears on both sides of a MAT multiply or the transpose statement.
40	Matrixes are not the same size	Matrixes sizes	Matrixes have different sizes.
41	Undimensioned variable	Undimensioned variable	Attempt to use an undimensioned matrix.
42	(Not used)	(Not used)	
43	Matrix not square	Matrix not square	Attempt to invert a nonsquare matrix
44	(Not used)	(Not used)	
45	Data is greater than specified record size	Data > lrecl exceeded	Logical record length limit

Table A-1. BASIC Error Messages (continues)

Code	AOS Text	RDOS/DOS Text	Meaning
46	More data supplied than requested	Input	Too many responses to [MAT] INPUT
47	Checksum	Buffer empty	File did not load correctly. Attempt to use BASIC editing commands on an empty buffer
48	Not a core image file		A filename not created by SAVE was specified in a LOAD, RUN, or CHAIN command.
48		Specified string not found	String you attempted to find with the BASIC editing commands does not exist in the line in the buffer.
49	(Not used)	No room for directory	A FILE or LIBRARY command cannot find 256 words in your program storage area to read the disk directory.
50	Illegal edit function	(Not used)	Attempt to use illegal command with BASIC editing commands
51	Edit buffer empty		Attempt to use BASIC editing commands on an empty buffer
51		User not active	Attempt to send message to an inactive or nonexistent user
52	Specified string not found		String you attempted to find with the BASIC editing commands does not exist in the line in the buffer
52		User in nomsg state	Attempt to send message to user whose terminal is in NOMSG state
53	(Not used)	(Not used)	

Table A-1. BASIC Error Messages (concluded)

Code	AOS Text	RDOS/DOS Text	Meaning
54	Generated statement is greater than 132 bytes	Statement length	A statement exceeded 132 characters in either internal or ASCII format, when expanded
55	Execute-only	Execute-only	Attempt to examine a program originating from a file with the execute-only attribute
56	Range	Range	Attempt to refer to a random record beyond 262,144
58	Incompatible core image file	Incompatible core image	Attempt to load a core image file saved under a different version of BASIC
59	Zero step	Zero step	FOR-NEXT with step 0
60	Keyboard response not in time	Time-out	Timed input decremented to 0.
61	Invalid decimal string	Invalid decimal string	Attempt to perform string arithmetic with nonnumeric characters
62	String arithmetic overflow	STAR ovfl	The result of string arithmetic requires more than 18 digits for precision representation.
63	Attempts to issue lock/unlock with RLS not running.	(Not used)	You cannot use lock/unlock if RLS is not running.
64	Attempt to lock same record twice.		The same user cannot lock the same record twice.
64		System active	Attempt by system manager to execute a BYE command while people are still on system.
65	RLS out of memory		RLS is full and cannot lock any more files. See the system manager if this is a serious problem.
65		Device timeout	Attempt to reference an off-line device. If device not found, time-out occurs within ten seconds.

Table A-2. RDOS/DOS Extended BASIC File I/O Error Messages

Code	Text	Meaning
01	Illegal filename	A to Z, 0 to 9 and $ are only valid characters
02	Illegal system command	Command not defined in operating system
03	Illegal command for device	INIT "$PTR", WRITE to $CDR, etc.
04	Not a core image file	File not in SAVE format
06	End of file	Attempt to read beyond EOF marker
07	Read protected file	Attempt to read from a read-protected file
08	Write protected file	Attempt to write to a write-protected file
09	File already exists	Attempt to create an existent file
10	File not found	Attempt to refer to a nonexistent file
11	Permanent file	Attempt to alter a permanent file
12	Attribute protected	Attempt to change file attributes when file is protected with RDOS attribute A
13	File not opened	Attempt to refer to an unopened file
14	Swapping disk data check	Disk error on swapping file
15	Revision check	Object file of LOAD or CHAIN not created by this revision of BASIC
16	Checksum	Disk error
17	Channel not available	Open two files with the same file number, or attempt to open too many files. Operating system file pool overflowed.
18	Line limit	Line limit exceeded on read or write line
20	Parity	Parity error on read line
23	No file space	Out of disk space. Delete files to make more room.
24	Read error	File read error
25	Select status	Unit is not ready or is write-protected.
29	Different directories	Files specified on different directories
30	Illegal device code	Device not in system or illegal device code
31	Illegal overlay	This is an unexpected system software error. If it occurs, please notify your local Data General representative.
37	Device already initialized	INIT has been used on the device.
38	Insufficient contiguous blocks	Insufficient number of free contiguous disk blocks. Reorganize partition.
39	Simultaneous I/O to QTY	Attempt by more than one user to read and/or write I/O to the same QTY line
41	No more DCB's	Attempt to open more devices or directories than are configured in the operating system

Table A-2. RDOS/DOS Extended BASIC File I/O Error Messages

Code	Text	Meaning
42	Illegal directory specifier	Illegal directory specifier
43	Unknown directory specifier	Directory specifier unknown
44	Directory too small	Directory is too small (operator only). Minimum directory size is 48 blocks.
45	Directory depth	Directory depth exceeded (operator only)
46	Directory in use	Attempt to release a directory in use by another program
47	Link depth	Link depth exceeded
48	File in use	Contact system operator if file is in your directory.
52	File position	Attempt to read out of bounds
54	Directory not initialized	Directory/device not initialized
58	Directory shared	No file space left
69	Disk is full	No file space left
89	Character overrun	The system failed to remove one character from the buffer before it input another. This error reflects a hardware condition.
90	Character framing	System failed to recognize a START or STOP bit in the buffer. This error reflects a hardware condition.

End of Appendix

Appendix B
Programming on Mark-Sense Cards

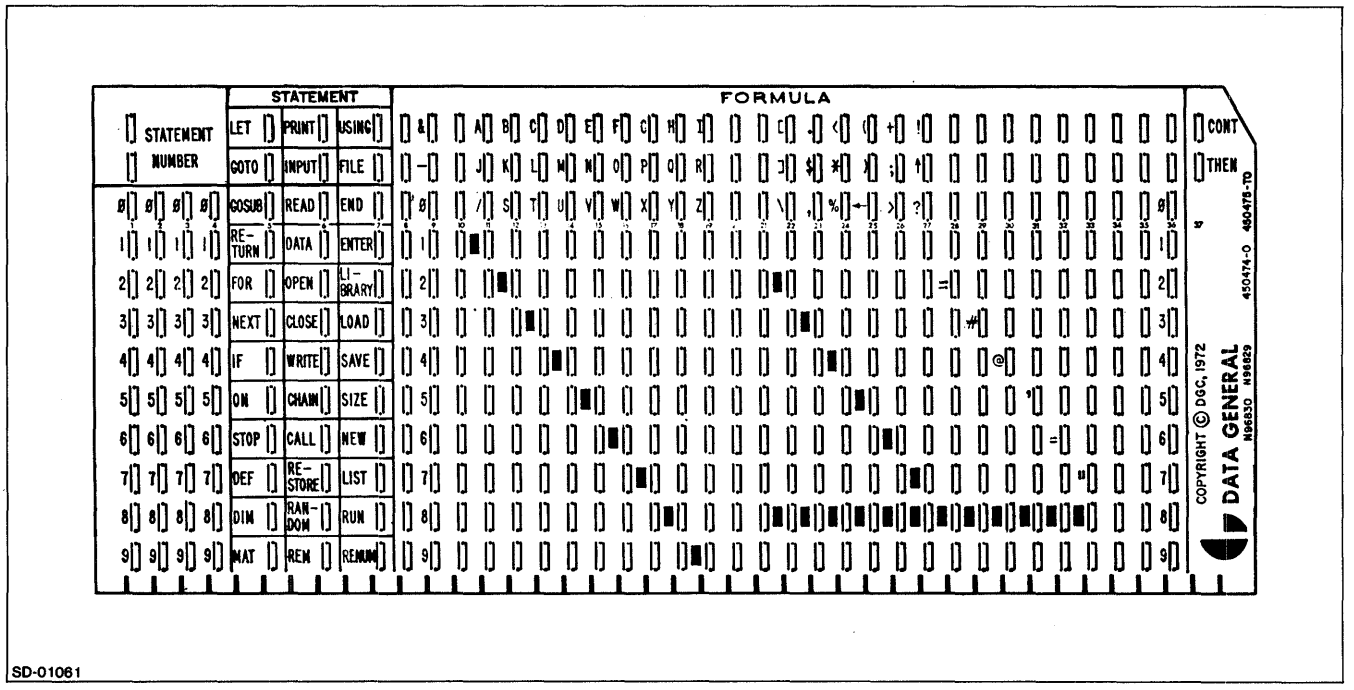


Figure B-1. Data General Extended BASIC Mark-Sense Programming Card

You may write BASIC programs on Data General's Extended BASIC mark-sense programming cards for input to the mark-sense card reader.

You may mark a stack of cards to include an entire BASIC program and input your stack to the card reader as a batch job. Your system manager will know about any special cards your system may require.

The mark-sense reader has an option that permits either markings or punches. If your card reader can read punches, it can also read marks, so you can mix marks and punches on a 37 column card. You cannot, however, intermix 37- and 80-column cards. If you want to use both types of cards on the same system, you must put them in separate stacks and process the 37-column cards with the CARDS command; use ENTER for the 80-column cards. Use a No. 2 pencil to mark cards.

A Data General Extended BASIC mark-sense card has 37 columns, as shown in Figure B-1. The first four columns assign statement numbers; the next three assign the BASIC statement keyword. You may write a single BASIC statement or part of a statement on each card.

The BASIC statement field of the mark-sense card is three columns, which allows all possible combinations of statement keywords. Mark the cards in the appropriate column(s); for example, Figure B-2 shows the statement

```
450 GOTO 200
```

The formula section of the card is 29 columns long and 12 rows deep. Proceed from the leftmost column to the right, for up to 29 characters. The CONT box on the far right allows you to continue your statement on the next card.

You must fill out the formula section of each card in Hollerith code. Each Data General mark-sense card contains a Hollerith code key (the black squares in the formula section), which indicates the lines to mark for each character. On all mark-sense cards, numbers are

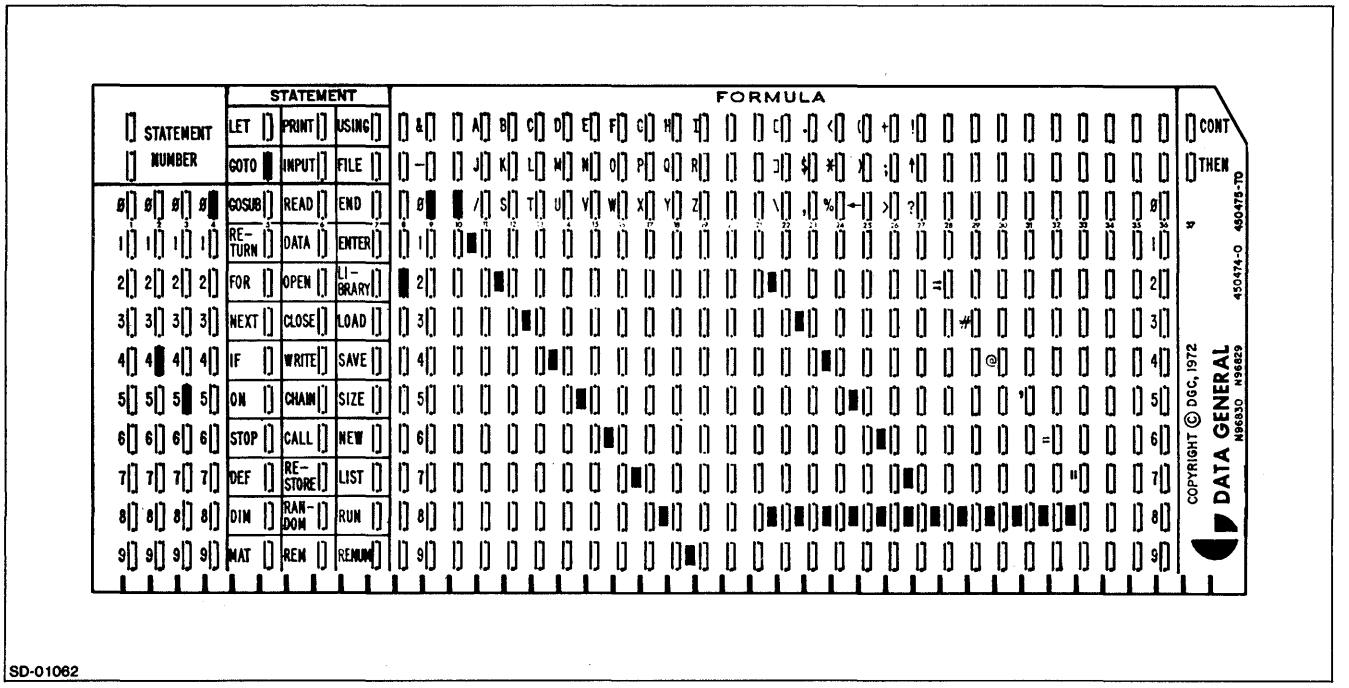


Figure B-2. 450 GOTO 200

marked directly in the appropriate rows, without reference to the key. Letters require two marks in a row; special characters require either two or three marks.

To mark a card, find the character column and the character you want to mark. Figure B-3a indicates the first column with an arrow and the letter V as the character to be marked. Mark the rectangle at the intersection of the two arrows, as shown.

Next, find any black boxes directly under the character; see the circled box under the V in Figure 3-Bb. (Some characters have more than one box below them.) Again, mark the rectangle at the intersection of the arrows, as shown in Figure B-3b. The first column now contains the completed markings for the character V, as shown in Figure B-3c.

Move over one column and repeat this process for your next character. *Note:* Do not draw the arrows on the mark-sense cards, as BASIC will try to interpret them.

If you use cards without a key, fill them out according to the Hollerith character set in Appendix C. The mark-sense card key and the Hollerith character set work exactly the same way; you may use whichever you find easier. If you use the Hollerith code set, the top horizontal line is number 12, the second from the top is number 11, and the other lines are numbered from 0 through 9. To indicate 4, put a mark on line 4; to indicate an asterisk (*), put marks on lines 11, 4, and 8; to indicate a number sign (#), put marks on lines 3 and 8.

On any card, you can continue a statement to the next card by marking the CONT box in the upper right-hand corner. Continue the statement on the following card in the FORMULA section.

To write an IF statement, mark IF in the statement section, mark the test expression in the formula section, and mark the THEN box in the upper right-hand corner of the card. Begin the next card in the statement section.

Figure B-4 further illustrates the use of mark-sense cards; the card shows the code for

```
10 IF V$ = "CAT" THEN
```

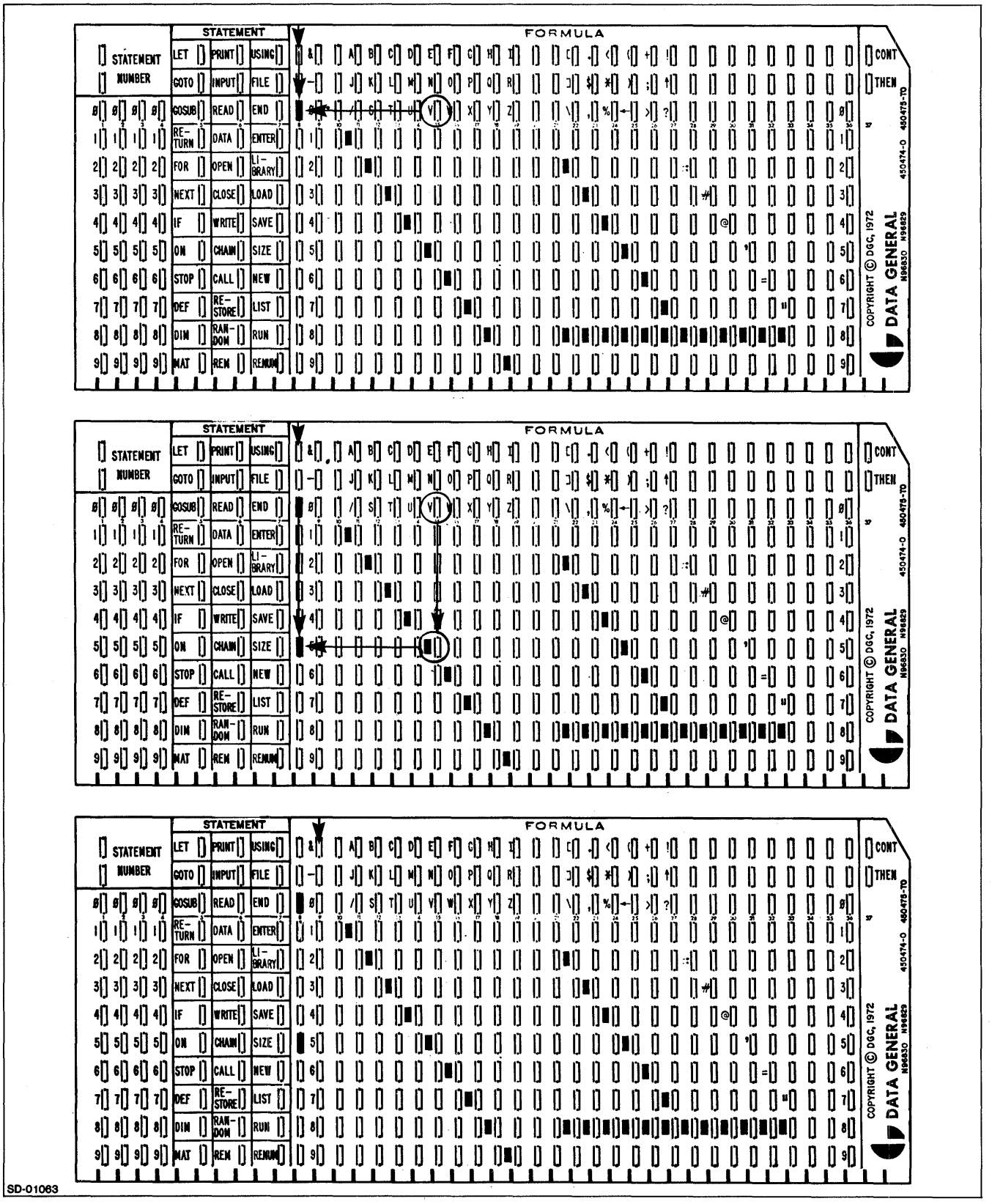


Figure B-3. Marking the Letter V

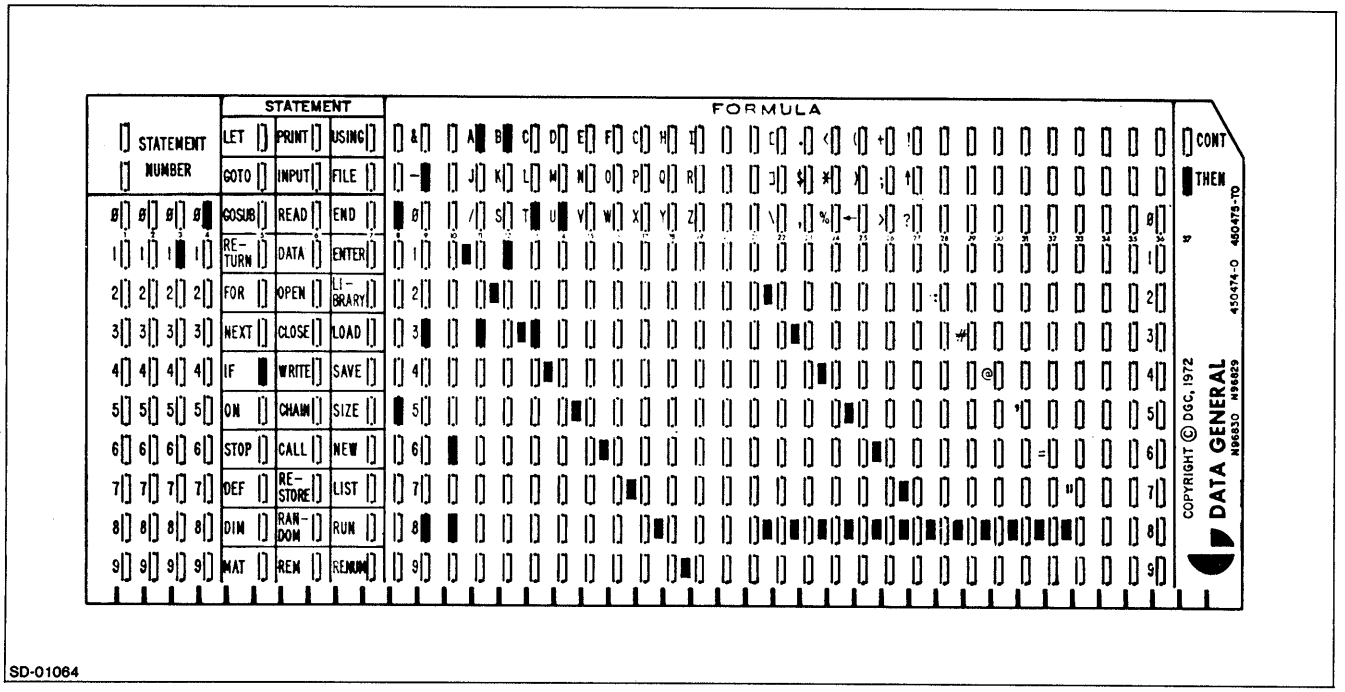


Figure B-4. 10 IF V$ = CAT THEN

End of Appendix

Appendix C
Hollerith Character Set

Character	Lines			Character	Lines		
0	0	-	-	.	12	3	8
1	1	-	-	<	12	4	8
2	2	-	-	(	12	5	8
3	3	-	-	+	12	6	8
4	4	-	-	!	12	7	8
5	5	-	-	]	11	2	8
6	6	-	-	$	11	3	8
7	7	-	-	*	11	4	8
8	8	-	-	)	11	5	8
9	9	-	-	;	11	6	8
A	12	1	-	↑	11	7	8
B	12	2	-	/	0	1	-
C	12	3	-	\	0	2	8
D	12	4	-	,	0	3	8
E	12	5	-	(comma)			
F	12	6	-	%	0	4	8
G	12	7	-	→	0	5	8
H	12	8	-	>	0	6	8
I	12	9	-	?	0	7	8
J	11	1	-	:	2	8	-
K	11	2	-	#	3	8	-
L	11	3	-	@	4	8	-
M	11	4	-	'	5	8	-
N	11	5	-	(apos.)			
O	11	6	-	=	6	8	-
P	11	7	-	"	7	8	-
Q	11	8	-	&	12	-	-
R	11	9	-	-	11	-	-
S	0	2	-	(minus)			
T	0	3	-				
U	0	4	-				
V	0	5	-				
W	0	6	-				
X	0	7	-				
Y	0	8	-				
Z	0	9	-				
[	12	2	8				

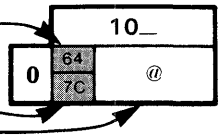
End of Appendix

Appendix D
ASCII Character Set

To find the *octal* value of a character, locate the character, and combine the first two digits at the top of the character's column with the third digit in the far left column.

Legend:

Character code in decimal
EBCDIC equivalent hexadecimal code
Character



Octal	00_		01_		02_		03_		04_		05_		06_		07_	
0	0 00	NUL	8 16	BS (BACK- SPACE)	16 10	DLE ↑P	24 18	CAN ↑X	32 40	SPACE	40 4D	(	48 F0	0	56 F8	8
1	1 01	SOH ↑A	9 05	HT (TAB)	17 11	DC1 ↑Q	25 19	EM ↑Y	33 5A	!	41 5D	)	49 F1	1	57 F9	9
2	2 02	STX ↑B	10 15	NL (NEW LINE)	18 12	DC2 ↑R	26 3F	SUB ↑Z	34 7F	'' (QUOTE)	42 5C	*	50 F2	2	58 7A	:
3	3 03	ETX ↑C	11 0B	VT (VERT TAB)	19 13	DC3 ↑S	27 27	ESC (ESCAPE)	35 7B	#	43 4E	+	51 F3	3	59 5E	;
4	4 37	EOT ↑D	12 0C	FF (FORM FEED)	20 3C	DC4 ↑T	28 1C	FS ↑\	36 5B	$	44 6B	, (COMMA)	52 F4	4	60 4C	<
5	5 2D	ENQ ↑E	13 0D	RT (RETURN)	21 3D	NAK ↑U	29 1D	GS ↑]	37 6C	%	45 60	-	53 F5	5	61 7E	=
6	6 2E	ACK ↑F	14 0E	SO ↑N	22 32	SYN ↑V	30 1E	RS ↑↑	38 50	&	46 4B	. (PERIOD)	54 F6	6	62 6E	>
7	7 2F	BEL ↑G	15 0F	SI ↑O	23 26	ETB ↑W	31 1F	US ↑←	39 7D	' (APOS)	47 61	/	55 F7	7	63 6F	?

Octal	10_		11_		12_		13_		14_		15_		16_		17_	
0	64 7C	@	72 C8	H	80 D7	P	88 E7	X	96 79	` (GRAVE)	104 88	h	112 97	p	120 A7	x
1	65 C1	A	73 C9	I	81 D8	Q	89 E8	Y	97 81	a	105 89	i	113 98	q	121 A8	y
2	66 C2	B	74 D1	J	82 D9	R	90 E9	Z	98 82	b	106 91	j	114 99	r	122 A9	z
3	67 C3	C	75 D2	K	83 E2	S	91 8D	[	99 83	c	107 92	k	115 A2	s	123 C0	{
4	68 C4	D	76 D3	L	84 E3	T	92 E0	\	100 84	d	108 93	l	116 A3	t	124 4F	|
5	69 C5	E	77 D4	M	85 E4	U	93 9D	]	101 85	e	109 94	m	117 A4	u	125 D0	}
6	70 C6	F	78 D5	N	86 E5	V	94 5F	↑ or ^	102 86	f	110 95	n	118 A5	v	126 A1	~ (TILDE)
7	71 C7	G	79 D6	O	87 E6	W	95 6D	← or _	103 87	g	111 96	o	119 A6	w	127 07	DEL (RUBOUT)

Character code in octal at top and left of charts.

↑ means CONTROL

End of Appendix

Appendix E
Statement, Command and Function Summary

The following table lists alphabetically, and summarizes briefly, the statements, commands, and functions of Extended BASIC. The RDOS and DOS privileged commands are indicated with a P in the column for the operating system. These commands can be executed only at the system terminal.

Formats and Descriptions	S	C	F	AOS AOS/VS	RDOS DOS
ABS(expr) The absolute value of an expression.			•	•	•
ACL "filename" *[,"UserID","attributes"]...* Prints a report of, or changes, the access control list for a file.	•	•		•	
ALL message Sends a message to all active users.	•	•			P •
ATN(expr) The arctangent of an angle. Result expressed in radians.			•	•	•
AUDIT *["filename"]* Audits your terminal's input and output.	•	•		•	•
BYE Sign-off command.	•	•		•	•
CALL subr *[,expr]...* Calls an assembly language subroutine.	•	•		•	•
CARDS "filename" Enters program in mark-sense card format.	•	•			•
CDIR name Creates a subdirectory.	•	•			P •
CHAIN "filename" *[THEN GOTO line no.]* Transfers control to the program named in the statement.	•	•		•	•
CHAR [{ "ON" / "OFF" / "characteristics" / "LPP", svar / "CPL", svar / "device" } [, { "ON" / "OFF" / "characteristics" / "LPP", svar / "CPL", svar / "device" }]...] Prints a report of, or changes, the current device characteristics.	•	•		•	

Formats and Descriptions	S	C	F	AOS AOS/VS	RDOS DOS
CHAR [[{"ON" / "OFF"}] "characteristics"[,]] ... Prints a report of, or changes, the current device characteristics.	•	•			•
CHATR "filename", attributes Changes file attributes.	•	•			•
CHR$(expr) Generates the character represented in the ASCII collating sequence by a number.			•	•	•
CLI *[command]* Provides access to the CLI without terminating the BASIC process.	•	•		•	
CLOSE [{*FILE* / #} *(file)*] Closes an open file or files.	•	•		•	•
CON Continues execution of a stopped program.		•		•	•
COS(expr) The cosine of an angle expressed in radians.			•	•	•
CPART name,size Creates a secondary partition.	•	•			P •
CPU(expr) Returns a value (0 or 1) equal to the status of a CPU console switch.			•		•
DATA {val / "str lit"} [, {*val* / *"str lit"*}] ... Defines data to be used by READ and MAT READ.	•			•	•
DEF Used with FNa(d) function to define a user function.	•			•	•
DELAY=expr Delays program execution for a specified amount of time.	•	•		•	•
DELETE "filename" Deletes a file from your directory.	•	•		•	•
DIM {svar(n) / array(m) / array(row,col)} [{*,svar(n)* / *,array(m)* / *,array(row,col)*}] ... Specifies the size of string variables and numeric arrays.	•	•		•	•
DIR *[pathname]* Displays, sets or stores the pathname of the current working directory.	•	•		•	
DIR [{*primary part [[:secondary part [:subdirectory]]* *secondary part [:subdirectory]* *subdirectory* *null-length string variable*}] Changes the directory and prints the name of the current directory.	•	•			P •
DISABLE Prevents the use of system console breaks.	•	•			P •

Formats and Descriptions	S	C	F	AOS AOS/VS	RDOS DOS	
DISK Prints the number of blocks used and the number available in the partition in which your directory resides.	•	•		•	•	E
ECHO Enables the display of characters on input.	•	•		•	•	E
ENABLE Cancels a DISABLE command.	•	•			P •	Y
END Stops program execution.	•			•	•	Y
ENTER {"filename" / svar} Merges the program named into the current program.	•	•		•	•	N
EOF (file) Returns a 1 if an end of file is detected, otherwise, 0.			•	•	•	E
ERASE n1, n2 Deletes statements from a program.	•	•		•	•	N
ESC Enables an ESC interrupt condition.	•	•		•	•	E
EXP(expr) The value of e to the power of an expression.			•	•	•	Y
FALL message Forces a message to all active users.	•	•			P •	E
{FILE / #} *["template"]* Prints the names of files in your directory that match the template.	•	•		•		Y
{FILE / #} Prints the filenames in your directory.	•	•			•	E
FMSG userID message Forces a message to a specific user.	•	•			P •	E
FNa(d) A user function which is defined in a DEF statement and returns a value.			•	•	•	E
FOR control var = expr1 TO expr2 *[STEP expr3]* Begins a FOR-NEXT loop and defines the number of times the loop is executed.	•			•	•	Y
FREE userID Interrupts execution of userID's program.	•	•			P •	E

Formats and Descriptions	S	C	F	AOS AOS/VS	RDOS DOS
GDIR Verifies the name of your directory.	•	•		•	•
GOSUB line no. Transfers program control to the first statement of a subroutine.	•			•	•
GOTO line no. Transfers program execution to a specified line.	•			•	•
GPOS { FILE / # } (file), var Determines the current file pointer position in an open file.	•	•		•	•
HELP "verb" Displays information about each BASIC statement and command.	•	•		•	•
IF { rel-expr / expr } { *[THEN]* statement / THEN line no. } Executes a statement based on whether an expression is true or false.	•	•		•	•
INIT name Initializes a directory or device for accessibility.	•	•			P •
INPUT ["str lit",] { var / svar } [{ *,var* / *,svar* }] ...*[;]* Accepts data for variables from the terminal.	•	•		•	•
INPUT { FILE / # } { (file) / (file,record) }, { var / svar }, [{ *var* / *svar* }]... Reads data in ASCII from a sequential-access file.	•	•		•	•
INT(expr) The largest integer not greater than the expression.	•	•		•	•
KILL userID Forces a specific user off the system.	•	•			P •
LEN(svar) Returns the number of characters currently assigned to a string variable.			•	•	•
[LET] { svar / var } = expr Assigns values or solutions to formulas to a variable.	•	•		•	•
LEVEL *[userID]* Monitors the priority constant for any user.	•	•			•
LIBRARY [{ *"directory", "template"* / *"directory"* / *"template"* }] Prints the names of files in the BASIC library directory that match the template.	•	•		•	

Formats and Descriptions	S	C	F	AOS AOS/VS	RDOS DOS
LIBRARY Prints the filenames in the library directory.	•	•			•
LIST [{ line n1 { TO , } line n2 ; line n1 { TO , } line n2 }] ["filename"] Outputs part or all of the current program to the terminal or other ASCII device.	•	•		•	•
LOAD "filename" Loads a previously saved program into the program storage area.		•		•	•
LOCK iden, "filename", start, record-size *[,time]* Gives exclusive access to a record in a file.	•	•		•	
LOG(expr) The natural logarithm of an expression.			•	•	•
LREAD *["str lit",]* svar *[,svar]* Reads a string that is terminated by either a null, form feed, or carriage return (NEW LINE) from the terminal.	•	•		•	•
LREAD { FILE # } { (file) (file, record) } , svar *[,svar1]* Reads a string from a record in either a random- or sequential-access file that is terminated by either a null, form feed, or carriage return (NEW LINE).	•	•		•	•
LWRITE svar *[,svar1]* Writes a string to your terminal that is delimited by either a null, form feed, or carriage return (NEW LINE).	•	•		•	•
LWRITE { FILE # } { (file) (file, record) } , svar *[,svar1]* Writes a string to a record into either a random- or sequential-access file which will be terminated by either a null, form feed, or carriage return (NEW LINE).	•	•		•	•
MAT mvar1 = mvar2 Assigns the dimensions and values of mvar2 to mvar1.	•	•		•	•
MAT mvar1 = mvar2 { + − } mvar Performs matrix addition or subtraction.	•	•		•	•
MAT mvar1 = { mvar2 (expr) } * mvar3 Multiplies a matrix by a numeric expression or another matrix.	•	•		•	•
MAT mvar = CON *[([row,]col)]* Sets the value of each matrix element to 1.	•	•		•	•
MAT mvar = IDN *[([row,]col)]* Sets the elements of the major diagonal of a matrix to 1s and all other elements to 0s.	•	•		•	•

Formats and Descriptions	S	C	F	AOS AOS/VS	RDOS DOS
MAT mvar1 = INV (mvar2) Performs matrix inversion.	•	•		•	•
MAT mvar1 = TRN (mvar2) Transposes matrix mvar2.	•	•		•	•
MAT mvar = ZER *([row,] col)* Sets the value of each matrix element to 0.	•	•		•	•
MAT INPUT *["str lit"]* mvar *[([row,]col)][,mvar[([row,]col)]]...* Specifies matrixes for which you enter data from the terminal when the statement is executed.	•	•		•	•
MAT INPUT {FILE / #} {(file) / (file,record)},mvar *[,mvar]...* Reads a record of matrix data in ASCII format from a file.	•	•		•	•
MAT PRINT mvar [{, / ;} mvar] ... [{; / ,}] Prints the contents of the specified matrixes.	•	•		•	•
MAT PRINT {FILE / #} {(file) / (file,record)}, mvar [{; / ,} *mvar...* {; / ,}] Outputs matrix file data to an ASCII device.	•	•		•	•
MAT READ mvar *[([row,]col)][,mvar[([row,]col)]]...* Reads data into the specified matrixes from the data list defined by DATA statement(s).	•	•		•	•
MAT READ {FILE / #} {(file) / (file,record)},mvar *[,mvar]...* Reads matrix data in binary format from a file.	•	•		•	•
MAT TINPUT *[(line no. [,time]),] ["str lit",]* mvar *[([row,] col)] [,mvar [([row,]col)]]...* Reads values from your terminal and assigns them to the elements of a matrix or list of matrixes, within a prescribed time.	•	•		•	•
MAT WRITE {FILE / #} {(file) / (file,record)},mvar *[,mvar]...* Writes matrix data in binary format to a file.	•	•		•	•
MAX *[val]* Sets the limit for a number of active users.	•	•			P •
MSG {pid / "processname" / "console name"}, "message" Transmits a message from your terminal to another programmer or to the system operator.	•	•		•	
MSG userID message Transmits messages to other users or the operator or cancels NOMSG.	•	•			•

Formats and Descriptions	S	C	F	AOS AOS/VS	RDOS DOS
NEW *["filename"]* Clears your storage area.	•	•		•	•
NEXT control var The last statement in a FOR-NEXT loop; changes the value of the control variable.	•	•		•	•
NOECHO Disables character echoing on input.	•	•		•	•
NOESC Disables ESC interrupt handling.	•	•		•	•
NOMSG Prevents receipt of messages.	•	•		•	•
ON ERR {THEN line no. / *[THEN]* statement} Directs the program to an error-handling routine when an error occurs.	•			•	•
ON ESC {THEN line no. / *[THEN]* statement} Directs the program to a user-handling routine when ESCape is pressed.	•			•	•
ON expr {GOTO / GOSUB} line no. *[,line no.]...* Transfers program control to a line number whose position in the argument list is computed from an expression.	•			•	•
OPEN {FILE / #} (file,mode),"filename" *[,recordsize [,filesize]]* Opens a file which can then be referred to by other file I/O statements.	•	•		•	•
ORD(expr) Represents the ordinal position of a character in the ASCII collating sequence.			•	•	•
PAGE=expr Sets the right margin of the terminal.	•	•		•	•
POS {(svar1) / ("str lit1")}, {(svar2) / ("str lit 2")}, (expr) Locates the position of a substring in a string.			•	•	•
{PRINT / ;} [{*expr* / *"str lit"* / *svar* / *TAB(n)*} [{,} {*expr* / *"str lit"* / *svar* / *TAB(n)*}]...] [{,} {;}] Prints specified data.	•	•		•	•
{PRINT / ;} {FILE / #} {(file) / (file,record)}, {expr / var / svar / "str lit"} [{,} {;} {expr / var / svar / "str lit"}]... [{,} {;}] Outputs data to an ASCII device.	•	•		•	•

Formats and Descriptions	S	C	F	AOS AOS/VS	RDOS DOS
{PRINT / ; } {FILE / # } (file,/*record*/), USING format, expr/,*expr*/... Formats output to files.	•	•		•	•
PRINT USING format, expr {;*expr* / ,*expr*} ... Formats printed output.	•	•		•	•
PUNCH [(*line n1* {TO / ,} *line n2*) (*line n1* {TO / ,} *line n2*)] Outputs part or all of the current program to the terminal punch.		•			•
RANDOMIZE Resets the random number generator.	•	•		•	•
READ {var / svar} [{,*var* / ,*svar*}]... Reads data from DATA statements.	•	•		•	•
READ {FILE / # } {(file) / (file, record)} , {var / svar} [, {*var* / *svar*}] ... Reads data in binary format from a file.	•	•		•	•
RELEASE name Prevents further I/O access to a previously initialized directory or device.	•	•			P •
REM */message/* Inserts explanatory comments into a program.	•			•	•
RENAME "oldfilename", "newfilename" Renames files.	•	•		•	•
RENUMBER [(*line n1* {STEP / ,} *n2*) (*line n1* {STEP / ,} *n2*)] Renumbers statements in the current program.		•		•	•
RESET [{*FILE* / # } */file/*] Positions the file pointer to the beginning of a file.	•	•		•	•
RESTORE */line no./* Moves the data element pointer to the beginning of a data list or DATA statement line.	•	•		•	•
RETRY Repeats the statement which caused an error.	•			•	•

Formats and Descriptions	S	C	F	AOS AOS/VS	RDOS DOS
RETURN Last statement of a subroutine; returns program control to statement following last GOSUB statement executed.	•			•	•
RND(expr) Random number n, such that 0 <= n < 1.			•	•	•
RUN [{ *"filename"* / *line no.* }] Executes a program either from the first line or from a specified line.		•		•	•
SAVE "filename" Writes the current program into your directory or to a device in binary format.	•	•		•	•
SEARCHLIST { *[pathname]* / pathname *[,pathname]* } Displays, sets or stores the searchlist setting. The current searchlist specifies which directories, in addition to the working directory, are searched for file references other than DELETE.	•	•		•	
SGN(expr) The algebraic sign of an expression.			•	•	•
SHARE "filename" Adds the resolution file attribute H (sharable) to a file that already exists in your directory.	•	•			•
SIN(expr) The sine of an angle expressed in radians.			•	•	•
SIZE Provides program and data storage usage information.	•	•		•	•
SPOS { FILE / # } (file), expr Moves the file pointer to the byte position specified by expr.	•	•		•	•
SQR(expr) The square root of an expression.			•	•	•
STOP Stops program execution.	•			•	•
STR$(expr) Converts a numeric expression to its string representation.			•	•	•
SYS(0) The time of day (seconds past midnight).			•	•	•
SYS(1) The day of the month.			•	•	•
SYS(2) The month of the year.			•	•	•

Formats and Descriptions	S	C	F	AOS AOS/VS	RDOS DOS
SYS(3) The year.			•	•	•
SYS(4) The multiplexor line number or terminal number (-1 for operator's terminal).			•	•	•
SYS(5) CPU time used, in seconds.			•	•	•
SYS(6) The number of file I/O statements executed.			•	•	•
SYS(7) The error code of the last runtime error.			•	•	•
SYS(8) The number of the file most recently opened.			•	•	•
SYS(9) Page size.			•	•	•
SYS(10) Tab size.			•	•	•
SYS(11) Hour of the day.			•	•	•
SYS(12) Minutes past last hour.			•	•	•
SYS(13) Seconds past last minute.			•	•	•
SYS(14) Seconds remaining on timed input.			•	•	•
SYS(15) The constant PI (3.14159).			•	•	•
SYS(16) The constant e (2.71828).			•	•	•
SYS(17) 1/10 second clock.			•	•	•
SYS(18) Total number of BASIC I/O calls.			•	•	•
SYS(19) Line number of the last error.			•	•	•
TAB=expr Sets the zone spacing for PRINT statements.	•	•		•	•

Formats and Descriptions	S	C	F	AOS AOS/VS	RDOS DOS
TAB(expr) Function used with PRINT for tabulating to a column.			•	•	•
TAN(expr) The tangent of an angle expressed in radians.			•	•	•
TIME=expr Establishes the time limit for timed input operation.	•	•		•	•
TINPUT *[(line no.[,time]),] ["str lit",]* {var / svar} [{,*var* / ,*svar*}] ...[;] Sets a time limit for programmer response.	•	•		•	•
UNLOCK *[iden]* Frees locked areas for use.	•	•		•	
UNSHARE "filename" Removes the resolution file attribute H (sharable) from a file that already exists in your directory.	•	•			•
USERS *[userID]* Prints a status report of all active users.	•	•			•
VAL {(svar) / ("str lit")} Returns decimal representation of a string.			•	•	•
var = DET(X) Obtains the determinant of the last matrix inverted by an INV statement.	•	•		•	•
WHATS "filename" Prints attributes and other information relating to a file.	•	•		•	•
WHO [{*processID* / *"processname"*}] Identifies others on the system or provides your own identification.	•	•		•	•
WRITE {FILE / #} {(file) / (file,record)} , {expr / var / svar / "str lit"} [, {*expr* / *var* / *svar* / *"str lit"*}] ... Writes data in binary format to a file.	•	•		•	•

Appendix F
Checklist of Operating System Incompatibilities

Generally differences between RDOS-DOS and AOS-AOS/VS are not syntactical but products of different operating environments. This appendix documents many of these differences, as well as some similarities.

1. For both RDOS-DOS and AOS-AOS/VS operating environments, the nesting limits are identical: 20 for FOR-NEXT and GOSUB statements, and 10 user-defined functions.
2. SAVE file formats are different. Thus, when converting programs from one operating system to another, you must transfer the programs in source format. If your programs are in SAVE format, they must first be loaded and then listed to disk files. This converts the SAVE format to source format. You can then enter the source files in the destination BASIC, and resave them.
3. AOS and AOS/VS diskettes are incompatible with both RDOS and DOS diskettes. RDOS and DOS diskettes are accessible under AOS by using the CLI utility RDOS.
4. Logon procedures are different. An RDOS user must press either ESC or the DEL key to begin logging on, and can logon to the CLI only from the system terminal. Other RDOS users logon directly to BASIC. AOS users press NEW LINE to begin logging on, and may have access to the CLI from any console before invoking BASIC.
5. Unlike RDOS, AOS-AOS/VS BASIC provides no system accounting data at logoff.
6. RDOS BASIC saves the user's workspace upon a forced logoff caused by line drop (but not operator kill), while AOS-AOS/VS BASIC has no such facility.
7. I/O involving strings can produce different results because standard delimiters for data-sensitive I/O under RDOS are CR, FF, and NULL, while under AOS they are CR, NL,FF, and NULL.
   - LWRITE FILE and WRITE FILE append a NULL to strings on output, and are compatible between RDOS and AOS BASIC.
   - READ FILE terminates string input upon detection of the NULL delimiter only, and is compatible between RDOS and AOS BASIC.
   - LREAD FILE and INPUT FILE terminate string input by the delimiters CR, FF, and NULL in RDOS BASIC, and by CR, FF, NL, and NULL in AOS BASIC. In this case,RDOS ignores the character NL on input, since these two statements use the .RDL system call, which ignores NL.
   - When a CR is imbedded in a string and is output to a terminal using LWRITE or PRINT, the CR produces a carriage return/line feed in RDOS, but only a carriage return in AOS.
   - Terminal input (INPUT or LREAD) is delimited by a CR or FF under RDOS, and by CR, NL, or FF in AOS. Terminals on a RDOS multiplexor line recognize the NL character as a delimiter, but convert the NL to a CR before passing the delimiter to BASIC.
8. The ACL statement is restricted to AOS.
9. Unlike RDOS, an execute-only BASIC program cannot be generated for AOS.
10. In RDOS, BYE deactivates the terminal from which it is issued. If issued from the system terminal, BYE brings the system down if no users are logged on.
11. AOS BASIC does not accept mark sense cards in BASIC's abbreviated format.
12. Characteristics for the statement CHAR differ somewhat between AOS and RDOS. For example, the RDOS characteristic NOE (no echo of input) becomes EB0 (echo all characters) and EB1 (echo all characters except control characters) under AOS. The RDOS characteristic XON is not found in AOS. And, the AOS characteristic FKT (enable function keys as delimiters) does not exist under RDOS.
13. The CHATR (change file attributes), SHARE, and UNSHARE statements are restricted to RDOS. Under AOS and AOS/VS, their functions are performed by a combination of the ACL (BASIC) and PERM (CLI) statements.

14. The command CLI is restricted to AOS and AOS/VS BASIC.
15. The maximum time interval for the DELAY statement is 65000 under RDOS, and approximately 4,000,000 in AOS.
16. RDOS Extended BASIC always reports the user's disk space. AOS and AOS/VS Extended BASIC sometimes generates the message "ERROR 517 - File is not a control point directory."
17. The command FILE in AOS accepts a template; under RDOS, it does not. This makes a difference to programs that combine FILE and AUDIT to build lists of filenames. The output in RDOS is a condensed display of filenames only, while the output in AOS is equivalent to the command CLI FILESTATUS/AS.
18. The syntax LET A=B=C is accepted by AOS only, while the syntax LET A,B=C is accepted by both AOS and RDOS.
19. The LEVEL statement exists in RDOS BASIC only.
20. In AOS, the LIBRARY statement accepts filename template and directory name arguments, and produces detailed output equivalent to the command CLI FILE/AS. The RDOS version accepts no arguments, and produces a condensed tabular display of filenames only.
21. The LOCK statement exists in AOS BASIC only.
22. Programs that construct messages and select message targets cannot be run under RDOS BASIC.
23. Under RDOS, the page width is 15 to 132; under AOS, it is 8 to 255. The page width cannot be set under AOS BATCH because it is considered a console characteristic.
24. The PUNCH statement exists in RDOS BASIC only.
25. When the SAVE command requests user confirmation to delete an existing file of the same name, the user must press CR under RDOS, NEW LINE under AOS.
26. The SEARCHLIST statement exists in AOS BASIC only.
27. The SHARE statement exists in RDOS BASIC only.
28. The maximum value of the argument to the TIME statement is approximately 65,000 under RDOS and 4,000,000 under AOS.
29. The UNLOCK statement exists in AOS BASIC only.
30. The UNSHARE statement exists in RDOS BASIC only.
31. The results of the WHATS statement differ between AOS and RDOS. AOS does not provide the date created or the date last used, and does not display the current use count.
32. The function CPU exists in RDOS BASIC only.
33. SYS(17) exists only in RDOS BASIC. AOS returns the message "ERROR 25 - Feature not available," which halts the program.
34. The lists of reserved filenames differ between RDOS and AOS.
35. The OPEN FILE statements differ:
    - The maximum file number is 15 for AOS, 7 for RDOS.
    - The file size argument is syntactically accepted in both RDOS and AOS, but is ignored in AOS BASIC.
    - The file I/O mode numbers in RDOS and AOS differ in interpretation and results:

Mode	RDOS	AOS
0	Sharable I/O	Exclusive I/O
1	Exclusive output Sharable input	Exclusive write only
2	Exclusive output Sharable input	Exclusive write only
3	Shared input only	Nonexclusive input only
7	Sharable I/O	Shared I/O

36. The assembler subroutine support routines .MPY. MPYA, .DVD, .DVDI do not exist in AOS and AOS/VS Extended BASIC because single ECLIPSE instructions can be used in their place.
37. In RDOS multiuser BASIC, the program must enter single-task mode to perform I/O to or from program variables.
38. File I/O error messages and their numbers differ between AOS and RDOS. For example, the RDOS message "I/O Error 10 - File not found" is "I/O Error 21 - File does not exist" in AOS. Such inconsistency prevents intelligent error handling by programs that are RDOS-AOS compatible.

    Note the following distinctions between error messages:

Number	AOS,AOS/VS	RDOS/DOS
09	File cannot be loaded - wrong revision	
12	Mantissa overflow	
13	Arithmetic underflow	
17	Arithmetic overflow	
24	Attempt to divide by zero	Directory empty
47	Checksum	Edit buffer empty
48	Not a core image file	String not found (edit)
49		No room for directory
50	Illegal edit function	
51	Edit buffer empty	User not active (MSG)
52	Specified string not found	User in NOMSG state
63	Attempt to issue lock/ unlock with RLS not running	
64	Attempt to lock same record twice	System active
65	RLS out of memory	Device timeout

End of Appendix

Index

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Z

reader comment form

Extended BASIC Reference Manual

093-000065-10

Your comments will help us improve the quality of this publication. They will be carefully reviewed by the writers. Please refer to page numbers if appropriate.

DID YOU FIND THE MATERIAL:

	YES	NO		YES	NO
• Useful?	☐	☐	• Well illustrated?	☐	☐
• Complete?	☐	☐	• Well written?	☐	☐
• Accurate?	☐	☐	• Easy to read?	☐	☐
• Well organized?	☐	☐	• Easy to understand?	☐	☐

COMMENTS:

_____

_____

_____

_____

_____

_____

_____

_____

HOW DID YOU USE THIS PUBLICATION?

- ☐ As an introduction to the subject
- ☐ For information about operating procedures
- ☐ To instruct in a class
- ☐ As a student in a class
- ☐ As a reference manual
- ☐ Other (*please explain*):

Name _____ Title _____

Firm _____ Date _____

Street _____ State _____

City _____ Zip _____

First fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 26 WESTBORO, MASS 01580

POSTAGE WILL BE PAID BY ADDRESSEE:

DataGeneral

ATTN: SDD Documentation
62 Alexander Drive
Research Triangle Park, NC 27709
USA

Second fold

CUT ALONG DOTTED LINE

Data General Corporation, Westboro, MA 01580

093-000065-10