# BASIC/1000D
Multi-User Real-Time BASIC

Reference Manual

BASIC/1000D
Multi-User Real-Time BASIC

Reference Manual

hp HEWLETT
PACKARD

PRINTING HISTORY

The Printing History below identifies the Edition of this Manual and any Updates that are included. Periodically, Update packages are distributed which contain replacement pages to be merged into the manual, including an updated copy of this Printing History page. Also, the update may contain write-in instructions.

Each reprinting of this manual will incorporate all past Updates, however, no new information will be added. Thus, the reprinted copy will be identical in content to prior printings of the same edition with its user-inserted update information. New editions of this manual will contain new information, as well as all Updates.

To determine what manual edition and update is compatible with your current software revision code, refer to the appropriate Software Numbering Catalog, Software Product Catalog, or Diagnostic Configurator Manual.

Multi-User Real-Time BASIC provides an augmented real-time version of the BASIC language with which as many as four users may code and execute programs simultaneously from different terminals. The Multi-User Real-Time BASIC subsystem provides functions, subroutines, and statements which allow you to schedule tasks, control instrument subsystems, the plotter and magnetic tape devices, and provides many additional capabilities. It may be run under control of the RTE Operating System.

This manual is a reference guide to the BASIC language, the BASIC system commands, and the subroutines available with the system. You should be familiar with the RTE Operating System. If a BASIC system has been generated and is available for your use, you will find all the information you need to create and run BASIC programs in this manual. These manuals are shown in the documentation maps which follow this preface.

Section I introduces Multi-User Real-Time BASIC and describes some of its general features. Sections II through VII describe the BASIC programming language. Expressions are defined in Section II and statements in Section III. Section IV describes statements in relation to strings and special characteristics of string variables and constants. Section V describes functions, lists the functions provided with BASIC, and tells you how to define your own functions. Both BASIC subroutines embedded in a BASIC program and external subroutines written in BASIC or other languages are described in Section VI. Section VII describes disc files and the statements and functions which manipulate files.

Section VIII tells you how to execute the Real-Time BASIC Interpreter. Section IX describes the commands used to communicate with the Interpreter once it is running. Debugging commands are described separately in Section X.

Sections XI through XVII deal with the subroutines and statements which schedule tasks and control specific hardware. Section XI describes real-time task scheduling and the subroutine calls BASIC provides for this purpose. Bit manipulation functions are described in Section XII. Both commands and subroutine calls used to read, write, and control magnetic tape devices are described in Section XIII. Section XIV provides instructions on generating the Branch and Mnemonic Tables which are required if external subroutines are used with BASIC. Section XV describes the HP 2313/91000 Subsystem subroutine calls and configuration. Section XVI describes the HP 6940 Subsystem configuration and routines. The HP 7210 Plotter subroutine calls are described in Section XVII.

Section XVIII provides instructions on generating the Instrument Table tape which is required if the instrumentation subroutines are to be used.

Appendix A contains alphabetical summaries of all statements, commands, and library subroutines. Appendix B describes error messages, Appendix C contains the ASCII character set, and Appendix D provides instructions for loading the Multi-User Real-Time BASIC software. Appendix E contains information about HP-IB/BASIC data conversion and subroutine table requirements for calling HP-IB utility subroutines.

CONTENTS

CONTENTS (continued)

CONTENTS (continued)

ILLUSTRATIONS

TABLES

INTRODUCTION

1-1. FEATURES

Multi-User Real-Time BASIC is a subsystem designed for use on RTE disc systems and provides a simple, easy-to-use augmented real-time version of the BASIC language. As many as four users may efficiently employ Real-Time BASIC concurrently, each with a uniquely named copy of the Real-Time BASIC software. Interaction with Multi-User BASIC can be via local or remote terminal devices, keypunched cards, paper tape, magnetic tape, or disc.

Real-Time BASIC provides you with these capabilities:

- Conversational programming.
- Multiple peripheral device I/O including graphics display.
- Real-time and event task scheduling.
- Dynamic program debugging aids.
- Fast access disc file storage for programs and data.
- Bit manipulation.
- Scheduling of BASIC, FORTRAN, ALGOL, and Assembly language programs.
- Instrumentation I/O and device subroutine simulation.
- User defined subroutines and functions.
- Character string manipulation.
- Program statement character editing and line resequencing.

1-2. CONVERSATIONAL PROGRAMMING

BASIC is an English-like programming language that is easy to learn and use. You enter programs directly into the Real-Time BASIC subsystem from a keyboard device. The BASIC Interpreter checks each statement as it is entered. If the statement contains an error, a message is printed which defines the error and you can correct it immediately. This type of interaction between you and the Interpreter is called conversational programming.

Conversational interaction allows you to test your programs step-by-step as they are being prepared. You are in constant touch with the system, its functioning, and its results. Programming and debugging are completed quickly, easily, and efficiently.

1-3. MULTIPLE PERIPHERAL DEVICE I/O

Multi-User Real-Time BASIC can provide a wide selection of input-output capabilities. It can be used with either hardcopy or display screen terminals, line printers, tape punches, and magnetic tape units. Data can be displayed on a hardcopy graphic plotter or TV monitor. The Interpreter also makes use of the fast-access disc storage capabilities of the RTE-II, RTE-III, RTE-IV, or RTE-IVB Operating System under which it operates.

1-4. REAL-TIME AND EVENT TASK SCHEDULING

Multi-User Real-Time BASIC is called *real-time* because the order of processing may be governed by time or by the occurrence of external events rather than by a strict sequence defined in the program itself. Because these events can occur in random order and require different amounts of processing, conflicts may arise between tasks. BASIC is capable of resolving these conflicts.

BASIC includes statements that assign execution priority to tasks, and statements to schedule execution of tasks as a function of time. The user can also connect task subroutines to event interrupts such as contact closures. Each task subroutine that is to be repeated during the course of system operations specifies the interval between successive executions of the task.

1-5.   PROGRAM DEBUGGING AIDS

Multi-User Real-Time BASIC provides commands that enable you to debug a program while it is running. The path of flow through a program can be displayed, the values of variables can be displayed and modified, and subroutine calls can be simulated.

1-6.   FILE CAPABILITIES

If you need or want a data base external to particular programs, Multi-User Real-Time BASIC provides a file capability allowing flexible yet straightforward manipulation of large volumes of data stored on disc files. Extensions to the READ, PRINT, and IF statements provide you with facilities for reading from or writing onto mass storage files and/or peripheral units.

Internally, files are organized as a collection of records each of 128 16-bit words. Thus, each record of a file may contain up to 64 numeric quantities. A string data item will occupy $1 + INT [(n + 1)/2]$ words, where n is its length in characters and INT truncates the quotient of the expression in brackets to an integer value.

When manipulated on a record-by-record basis, a file appears as a collection of subfiles which are the records. The ability to reference any record of the file directly allows you to partition your data and alter any group without disturbing the rest of the file.

BASIC, FORTRAN, ALGOL, and Assembly language programs can use the same files but BASIC requires a special format to which programs in the other languages must conform if BASIC programs are to use the files. The file must be type 1 with 128 word fixed length records. Each word in the record must be initialized with all bits equal to 1.

1-7.   ENVIRONMENT

1-8.   HARDWARE

The BASIC Interpreter operates within the RTE-II, RTE-III or RTE-IV hardware environment consisting of an HP 1000 Series Computer System with Floating Point Hardware. (Refer to the appropriate system Programming and Operating Manual for equipment configurations.)

For RTE-II, the BASIC Interpreter can operate within the minimum system on a 2100 or 21MX computer with 24K memory.

For RTE-III, the BASIC Interpreter can operate within the minimum system on a 21MX computer with 32K memory.

Peripheral devices required for BASIC are a system console and a disc drive. Optional devices include a line printer, card reader, photoreader, plotter, TV monitor, HP 2313 and HP 6940 Subsystems, and additional discs and terminals.

A typical system configuration is depicted in Figure 1-1.
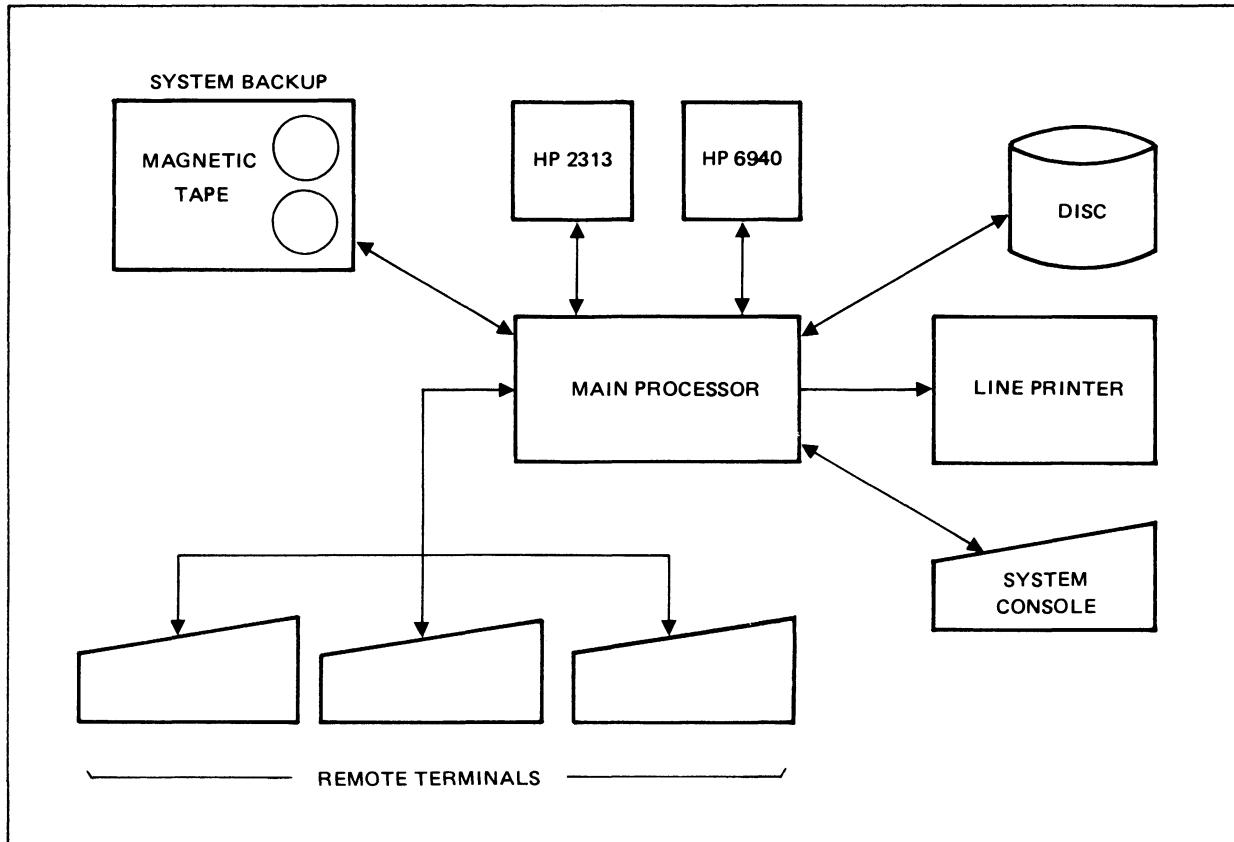
Figure 1-1. Typical System

1-9.   SOFTWARE

The BASIC Interpreter is an option which runs under control of either of the following operating systems:

- Real-Time Executive II (RTE-II)
- Real-Time Executive III (RTE-III)

The BASIC Interpreter requires an RTE-II system with a background memory area of at least 8K words and the File Management Package. It requires an RTE-III system with a main memory partition of at least 8K words. At least 450 words of base page must be available.

Multi-User Real-Time BASIC is a self-contained segmented program operating in RTE-II background or an RTE-III partition. The subsystem consists of the following modules and components:

- BASIC, the main program and all disc resident segments used for control and I/O.

- Branch and Mnemonic Tables, used to link BASIC to subroutines and functions. These tables are binary disc files, not relocatable modules and are created by a separate table generator program, RTETG.

- Disc Resident User-Written subroutines.

- Trap Table Module, used for keeping track of all real-time tasks and traps.

The BASIC program consists of a main program and 8 disc resident segments. The purpose of each segment is:

Segment 1 - Statement syntax checking.
Segment 2 - Program and error listing.
Segment 3 - Pre-execution processing, building symbol tables and intermediate code.
Segment 4 - Execution of Programs.
Segment 5 - Command execution.
Segment 6 - Command execution.
Segment 7 - Tracing, debugging and subroutine simulation.
Segment 8 - Execution of PAUSE, STOP, END, ASSIGN, and CHAIN statement.

Each segment is loaded from the disc as required by the BASIC main program.

Figure 1-2 illustrates the layout of BASIC components in the RTE-II system memory. The layout is the same for RTE-III except two partitions are used instead of background and foreground.
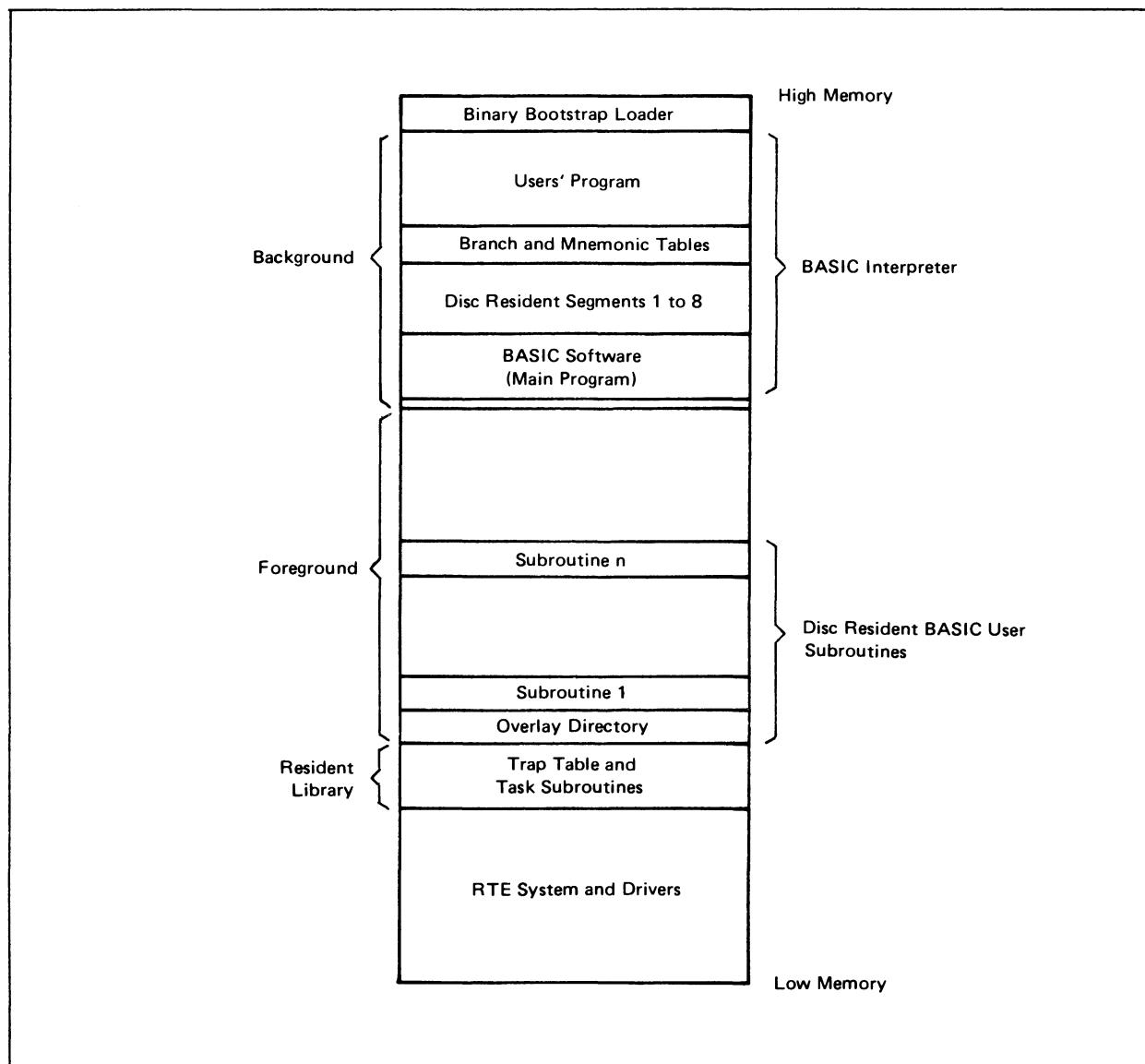
Figure 1-2. RTE Memory Layout with BASIC.

1-10. COMMANDS AND STATEMENTS

1-11. COMMANDS

BASIC commands instruct the BASIC Interpreter to perform certain control functions. Commands differ from the statements used to write a program in the BASIC language.

A command instructs the Interpreter to perform some action immediately, while a statement is an instruction to perform an action only when the program is run. A statement is always preceded by a statement number; a command never is.

Any BASIC command can be entered following the BASIC prompt character >. Each command is a single word that must be typed in its entirety with no embedded blanks. (DELETE and RESUME are exceptions, you may type DEL and RES.) If misspelled, the computer will return an error message. Some commands have parameters to further define command operation.

For instance, BYE is a command that you use to terminate the BASIC Interpreter and return to the operating system. It has no parameters. Another command, LIST, prints the program currently being entered. It may have parameters to specify that only part of the program is to be listed, or to indicate a particular list destination.

1-12. STATEMENTS

Statements are used to write a BASIC program that will subsequently be executed. Each statement performs a particular function. Every statement you enter becomes part of the current program and is kept until explicitly deleted or you exit from BASIC with BYE.

A statement is always preceded by a statement number. This number is an integer between 1 and 9999. The statement number indicates the order in which the statements will be executed. Statements are ordered by BASIC from the lowest to the highest statement number. Since this order is maintained by the Interpreter, it is not necessary for you to enter statements in execution order so long as the numbers are in that order.

Following each statement, you must press the RETURN key to inform the Interpreter that the statement is complete. The Interpreter generates a linefeed and prints the prompt character > on the next line to signal that the statement is accepted. If an error is made entering the statement, the computer prints an error message.

BASIC statements have a free format. This means that blanks are ignored. For instance, all these statements are equivalent.

```
>30     PRINT     S
>30 PRINTS
>30PRINTS
>    30 PRINTS
> 3 0 P R I N T S
>
```

1-13. BASIC PROGRAMS

Any statement or group of statements that can be executed constitutes a program.

A program may consist of only two statements.

This is an example of such a program.

```
>100 PRINT 35+5
>110 END
```

100 is the statement number. PRINT is the key word or instruction that tells the Interpreter the kind of action to perform. In this case, it prints the result of the expression that follows. 35+5 is an arithmetic expression. It is evaluated by the Interpreter, and when the program is run, the result is printed. The END statement indicates the program is complete.

Usually a program contains more than one statement.

These four statements are a program:

```
>10 INPUT A,B,C,D,E
>20 LET S = (A+B+C+D+E)/5
>30 PRINT S
>40 END
```

This program, which calculates the average of five numbers is shown in the order of its execution. It could be entered in any order if the statement numbers assigned to each statement were not changed.

This program runs exactly like the program above.

```
>20 LET S = (A+B+C+D+E)/5
>10 INPUT A,B,C,D,E
>30 PRINT S
>40 END
```

It is generally a good idea to number statements in increments of 10. This allows room to intersperse additional statements as needed. If you have too many statements to insert, you may renumber the statements with the RESEQ command.

1-14. CHARACTER EDITING

If you make an error while entering a statement, an error message is printed which indicates the nature of the error, and you can correct the error immediately. If you want to character edit the error, you must redisplay the incorrect statement by typing a P and then make the desired corrections using the control characters required.

Similarly if you find an inappropriate statement in an existing program, you can easily correct the statement. You can position to the line you want to edit by typing a slash (/) for each line, stepping line by line through the program until you find a line which requires editing. You can also use the LIST command to position to the line if you know the statement number. In this case, you do not have to redisplay the line with the P command.

If a syntax error is encountered when loading a program from a disc file, BASIC prints an error message and the offending line. The line is ready for character editing. After it is corrected, you can use the MERGE command to load the remainder of the program.

In each case, after the line you want to edit is displayed, you type a P and the appropriate control characters to change the line. Five special control characters are available for character editing and line correction. Four of the control characters are entered by first pressing the control key (CNTL) and holding it down while pressing the letter key I, R, C, or T. The fifth control character is a slash which is entered without using the control key.

Control Character	Use
I^c	Inserts new characters into a line prior to the current position.
R^c	Replaces characters in a line at the current position.
C^c	Deletes specific characters from a line beginning at the current position.
T^c	Truncates all remaining characters in a line including the current character.
/	Leaves a character unchanged or if typed instead of the P command, lists the next line in the program.
/n	Positions to program statement n and lists it.

The following is an example of character editing used to correct an entry error. (User entries are underlined for clarity.)

>10 A*10	*Incorrectly entered line.*
MISSING ASSIGNMENT OPERATOR IN LINE 10	*Error message.*
>P	*Request line be reprinted.*
10 A*10	
MISSING ASSIGNMENT OPERATOR IN LINE 10	*Line is reprinted.*
>P///X =	*Request edit. Insert X = after first three charac-*
10 X =A*10	*ters which are unchanged. (Space is counted as a character.) Line is reprinted showing corrections.*

You can change the / control character to some other character by typing X*l* in response to the BASIC prompt, >. *l* will be the new character replacing the slash. *l* can be any non-numeric character except: P, I^c, R^c, C^c, T^c.

Here is an example of editing an incorrect line loaded from a disc file:

```
BASIC READY
>LOAD PROGA
UNDECIPHERABLE OPERAND IN LINE 200
   200 LET A= -
>P///////////6
   200 LET A= -6
>MERGE PROGA
```

Command to load a program PROGA from a disc file.

Error message and line number printed.

Line displayed.

Type P and control characters to edit line.

BASIC redisplays corrected line.

Command to merge the remainder of the program with the new line and previous ones.

1-15.   CHARACTER EDIT IN MULTIPOINT ENVIRONMENT

The Q command is used for character edits when using BASIC in a multipoint environment. The command is used to edit the pending line. When the Q command is entered, the pending line is displayed, along with a delimiter (GS) to the left of the line. The delimiter is not part of the text string and must be preserved to assure proper operation. The delimiter is represented as a pound sign (#) throughout the rest of this section. The EDITR will position the cursor underneath the first character of the line. You may now edit the line using any of the following procedures.

To retain the pending line as it is, immediately hit the ENTER key. For example:

```
/Q
  #ABCDEFGHIJKL  ◄──────────────── Cursor displayed under first character in line.
  _                                Press the ENTER key and line is retained as is.
/P
   ABCDEFGHIJKL  ◄──────────────── Line displayed remains the same
/
```

To truncate characters from the end of a line, position the cursor immediately after the last character to be retained. Strike the ENTER key to enter all the characters between the left margin and the current cursor position. The intrinsic terminal key CLEAR DISPLAY can also be used to delete characters at the end of a line. After using the CLEAR DISPLAY key, the ENTER key is used to enter the edited line. For example:

```
/Q
  #ABCDEFGHIJKL  ◄──────────────── Position cursor under I, press ENTER key
          _
   ABCDEFGH  ◄──────────────────── Edited line is displayed
/Q
  #ABCDEFGH  ◄──────────────────── Position cursor under F, press CLEAR DISPLAY, then ENTER
      _
   ABCDE  ◄──────────────────────── Edited line is displayed
/
```

To add characters in the middle of a line, the INSERT CHAR key may be used. Press the INSERT CHAR key. The red light above the key should come on. Move the cursor to the position where the characters are to be added, and type in the new characters. Finally, position the cursor at the end of the line and hit

the ENTER key. The insert light will go off and the edited line will remain as the pending line.
For example:

```
/Q
   #ABCDEFGHIJKL  ◄─────────────────── Position cursor underneath F, depress the INSERT CHAR
              ▔                          key, and type in the new characters "123". Position
                                         cursor at end of line and press ENTER.
      ABCDEF123GHIJKL ◄──────────────── Edited line is displayed
/
```

To delete one or more characters, position the cursor under each character to be deleted and press the
DELETE CHAR key. The character will be deleted from the display and the rest of the line will be shifted
left to fill in the gap. After all of the desired deletions have been made, move the cursor to the end of the
line and press the ENTER key. Do not delete the delimiter at the beginning of the line. For example:

```
/Q
   #ABCDEFGHIJKL ◄──────────────────── Position cursor under the G, press the
              ▔                          DELETE CHAR key three times, move the cursor
                                         to the end of the line and press ENTER.
      ABCDEFJKL ◄──────────────────── Edited line is displayed
/
```

1-16. CORRECTION OF TYPING ERRORS

You may use the following keys to correct typing errors:

RUBOUT deletes the current line you are typing; on some terminals a DEL key is used instead.

BACKSPACE deletes a character.

If the terminal does not have a key labeled BACKSPACE, you may use Control H (H^c). Press the
control key (CTRL), hold it down and press the H key. The backspace is printed as an underline ▁ or a
back arrow ◄. More than one character may be deleted by repeating BACKSPACE or H^c for each
character you want to delete.

Terminals which have both upper and lower case characters should be locked into upper case mode if
possible.

1-17. LOGICAL UNIT NUMBERS

Logical unit numbers, abbreviated LU in this manual, are decimal integers between 0 and 63 used to
address I/O devices. Numbers 1 through 6 must always refer to the following devices:

      1 - system console
      2 - system disc
      3 - auxiliary disc (optional)
      4 - standard output unit
      5 - standard input unit
      6 - standard list unit

The standard devices may be:

> output - paper tape punch or magnetic tape
> input - paper tape reader, card reader, or terminal
> list - line printer or terminal.

The remaining logical unit numbers (7 - 63) may be assigned to any type device. Logical unit number 0 is not associated with a particular device but it used to essentially turn off an input or output statement.

1-18. SYNTAX CONVENTIONS

The following syntax conventions are used in this manual to specify command and statement formats.

UPPER-CASE BLOCK LETTERS	Literals that must be specified exactly as shown.
lower-case italics	Type of information to be supplied by you; most parameters are in this form.
[*,parameter*]	Optional parameters are enclosed in brackets.
parameter 1 *parameter 2* *parameter 3*	One and only one of the stacked parameters may be specified
⌈*parameter 1*⌉ |*parameter 2*| ⌊*parameter 3*⌋	All bracketed parameters are optional, only one may be specified.
[*,param1* [*,param2*]]	Series of optional parameters; the last parameter may be omitted with no indication; embedded parameters must be supplied.
. . .	Ellipsis indicates that the previous parameter or series of bracketed parameters can be repeated.

EXPRESSIONS

An expression combines constants, variables, or functions with operators in an ordered sequence. When evaluated, an expression must result in a value. An expression that, when evaluated, is converted to an integer, is called an integer expression. Constants, variables, and functions represent values; operators tell the computer the type of operation to perform on these values.

Some examples of expressions are:

(P + 5)/27

P is a variable that must have been previously assigned a value. 5 and 27 are constants. The slash is the divided operator. Parentheses group those portions of the expression evaluated first.

If P = 49, it is an integer expression with the value 2.

(N − (R + 5)) − T

N, R, and T must all have been assigned values. + and − are the add and subtract operators. The innermost parentheses enclose the part evaluated first.

If N =20, R =10, and T =5, the value of the integer expression is zero.

2-1. CONSTANTS

A constant is either numeric or it is a literal string.

2-2. NUMERIC CONSTANTS

A numeric constant is a positive or negative decimal number including zero. It may be written in any of the following three forms:

● As an integer - a series of digits with no decimal point.

● As a fixed point number - series of digits with one decimal point preceding, following, or embedded within the series.

● As a floating point number - an integer or fixed point number followed by the letter E and an optionally signed integer.

Examples of Integers:

    1234
    −70
    0

Examples of Fixed Point Numbers:

    1234.
    1234.56
    −.0123

2-3.   FLOATING-POINT NUMBERS

In the floating point notation, the number preceding E is a magnitude that is multiplied by some power of 10. The integer after E is the exponent, that is, it is the power of 10 by which the magnitude is multiplied.

The exponent of a floating point number is used to position the decimal point. Without this notation, describing a very large or very small number would be cumbersome:

    1E+35 = 100000000000000000000000000000000000
    1E−35 = .00000000000000000000000000000000000001

Examples of Floating-Point Numbers:

1E+23	$=1 \times 10^{23} = 100000000000000000000000$
1.0E23	(same as above)
.001E26ʼ	(same as above)
1.02E+4	$=1.02 \times 10^4 = 10200.$
1.02E−4	$=.000102$

Within the computer, all these constants are represented as floating-point real numbers whose precision is 6 or 7 digits and whose size is between 10^{-38} and 10^{38}.

2-4.   LITERAL STRINGS

A literal string consists of a sequence of characters in the ASCII character set enclosed within quotes. The quote is the only character excluded from the character string.

Examples of Literal Strings:

"ABC"	" " *(a null, empty, or zero length string)*
"‼WHAT A DAY‼"	" " *(a string with two blanks)*
" X Y Z "	

Blank spaces are significant within a string.

2-5.   VARIABLES

A variable is a name to which a value is assigned. This value may be changed during program execution. A reference to the variable acts as a reference to its current value. Variables are either string or numeric. Further, numeric variables are either simple or subscripted.

Simple numeric variables are a single letter (from A to Z) or a letter immediately followed by a digit (from 0 to 9):

A   A0
P   P5
X   X9

A variable of this type always contains a numeric value that is represented in the computer by a real floating-point number.

If a variable names an array, it must be subscripted. Only the alphabetic characters A through Z may be used to name an array. When a variable is subscripted, the variable name is followed by one or two subscript values enclosed in parentheses. If there are two subscripts, they are separated by a comma. A subscript may be an integer constant or variable, or any expression that is evaluated to an integer value:

A(1)   A (N,M)
P(1,1)   P (Q5,N/2)
X(N+1)   X (10,10)

A simple variable and a subscripted variable may have the same name with no implied relation between the two. For example, a simple variable named A is totally distinct from a subscripted variable named A (1,1).

Simple numeric variables can be used without being declared. Subscripted variables must be declared with a DIM statement (see Section III) if the array dimensions are greater than 10 rows, or 10 rows and 10 columns. The first subscript is always the row number, the second the column number. The subscript expressions must result in a value between 1 and the maximum number of rows and columns.

A variable may also contain a string of characters. This type of variable, a string array, is identified by a variable name consisting of a letter and $:

A$ P$

The value of a string variable is always a string of characters, possibly null or zero length. If the string array contains a single character, it need not be declared with a DIM statement (see Section III). String arrays differ from numeric arrays in that they have only one dimension. You may optionally use two subscripts which refer to the first and last characters in the substring you want to reference (See Section IV, String Arrays). You may also use one subscript to refer to the first character of the substring. In this case, the last character of the substring will be the last character of the string. Examples of subscripted string array names (substrings) are:

A$(1,3)   Z$(N,N+M)   A$(10)

2-6.   FUNCTIONS

A function names an operation that is performed using one or more parameter values to produce a single value result. A numeric function is identified by a three-letter name followed by one or more formal parameters enclosed in parentheses. If there is more than one, the parameters are separated by commas. The number and type of the parameters depends on the particular function. The formal parameters in the function definition are replaced by actual parameters when the function is used.

Since a function results in a single value, a system-defined function (see Section V) can be used anywhere in an expression where a constant or variable can be used. To use a function, the function name followed by actual parameters in parentheses (known as a function call) is placed in an expression. The resulting value is used in the evaluation of the expression.

Examples of common functions:

SQR(x)　　　　　where x is a numeric expression that results in a value ≥ 0. When called, it returns the square root of x. For instance, if N=2, SQR(N+2) = 2.

ABS(x)　　　　　where x is any numeric expression. When called, it returns the absolute value of x. For instance, ABS(-33) = 33.

BASIC provides many built-in functions that perform common operations such as finding the sine, taking the square root, or finding the absolute value of a number. The available functions are listed in Section V. In addition, you may define and name your own functions should you need to repeat a particular operation. How to write functions is described in Section V, Functions.

2-7.　OPERATORS

An operator performs a mathematical or logical operation on one or two values resulting in a single value. Generally, an operator is between two values, but there are unary operators that precede a single value. For instance, the minus sign in A - B is a binary operator that results in subtraction of B from A; the minus sign in -A is a unary operator indicating that A is to be negated.

The combination of one or two operands with an operator forms an expression. The operands that appear in an expression can be constants, variables, functions, or other expressions.

Operators may be divided into types depending on the kind of operation performed. The main types are arithmetic, relational, and logical (or Boolean) operators.

The arithmetic operators are:

+	Add (or if unary, positive)	A + B or +A
−	Subract (or if unary, negative)	A − B or −A
*	Multiply	A × B
/	Divide	A ÷ B
↑ or ∧	Exponentiate	A^B

In an expression, the arithmetic operators cause an arithmetic operation resulting in a single numeric value.

The relational operators are:

=	Equal	A = B
<	Less than	A < B
>	Greater than	A > B
<=	Less than or equal to	$A \leq B$
>=	Greater than or equal to	$A \geq B$
<> or #	Not equal	$A \neq B$

When relational operators are evaluated in an expression they return the value 1 if the relation is found to be true, or the value 0 if the relation is false. For instance, A = B is evaluated as 1 if A and B are equal in value, as 0 if they are unequal.

Logical or Boolean operators are:

AND	Logical "and"	A AND B
OR	Logical "or"	A OR B
NOT	Logical complement	NOT A

Like the relational operators, the evaluation of an expression using logical operators results in the value 1 if the expression is true, the value 0 if the expression is false.

Logical operators are evaluated as follows:

A AND B	= 1 (true) if A and B are both $\neq$ 0; = 0 (false) if A = 0 or B = 0
A OR B	= 1 (true) if A $\neq$ 0 or B $\neq$ 0; = 0 (false) if both A and B = 0
NOT A	= 1 (true) if A = 0; = 0 (false) if A $\neq$ 0

2-8.   EVALUATING EXPRESSIONS

An expression is evaluated by replacing each variable with its value, evaluating any function calls and performing the operations indicated by the operators. The order in which operations are performed is determined by the hierarchy of operators:

↑ or ∧	(highest)
NOT	
* /	
+ −	
Relational (=, <, >, <=, >=, <>)	
AND	
OR	(lowest)

The operator at the highest level is performed first followed by any other operators in the hierarchy shown above. If operators are at the same level, the order is from left to right. Parentheses can be used to override this order. Operations enclosed in parentheses are performed before any operations outside the parentheses. When parentheses are nested, operations within the innermost pair are performed first.

For instance: 5 + 6*7 is evaluated as 5 + (6 × 7) = 47

7/14*2/5 is evaluated as ((7/14)×2)/5 = .2

If A=1, B=2, C=3, D=3.14, E=0

then: A+B*C is evaluated as A + (B*C) = 7

A*B+C is evaluated as (A*B) + C = 5

A+B−C is evaluated as (A+B)−C = 0

(A+B)*C is evaluated as (A+B)*C = 9

When a unary operator immediately follows another operator of higher precedence, the unary operator assumes the same precedence as the preceding operator. For instance,

B ↑ −B ↑ C is evaluated as $(B^{-B})^C$ = 1/64 or .015625

In a relation, the relational operator determines whether the relation is equal to 1 (true) or 0 (false):

(A*B) < (A−C/3) is evaluated as 0 (false) since A*B=2 which is not less than A−C/3=0

In a logical expression, other operators are evaluated first for values of zero (false) or non-zero (true). The logical operators determine whether the entire expression is equal to 0 (false) or 1 (true):

E AND A−C/3   is evaluated as 0 (false) since both terms in the expression are equal to zero (false).

A+B AND A*B   is evaluated as 1 (true) since both terms in the expression are different from zero (true).

A=B OR C=SIN(D)   is evaluated as 0 (false) since both expressions are false (0).

A OR E   is evaluated as 1 (true) since one term of the expression (A) is not equal to zero.

NOT E   is evaluated as 1 (true) since E=0.

This section describes statements used in writing a Real-Time BASIC program. Statements must be preceded by a line number and are terminated by pressing the RETURN key when entered. Statements are executed in numeric sequence, but may be entered in any sequence.

Unlike COBOL, FORTRAN, and other programming languages, BASIC statements are interpreted at the time they are entered; thus a compile stage is not required. Invalid statements are immediately rejected. Statements are not executed, however, until the program is executed with the RUN command (see Section IX).

Table 3-1. lists some statements used in writing a program and briefly describes each. Detailed explanations of each statement are provided in the remainder of the section. Additional statements related to specific programming objectives are introduced and explained in subsequent sections of this part of the manual. A complete list of Real-Time BASIC statements and their uses is provided in Appendix A.

3-1.   LET

This statement assigns a value to one or more variables. The value may be in the form of an expression, a constant, a string, or another variable of the same type.

Format

> When the value of the expression is assigned to a single variable, the formats are:
>
> [LET] *variable = expression*
>
> When the same value is to be assigned to more than one variable, the formats are:
>
> [LET] *variable = variable = . . . = variable = expression*

In this statement, the equal sign is an assignment operator. It does not indicate equality, but is a signal that the value on the right of the assignment operator be assigned to the variable on the left. If any ambiguity exists between the relational operator "=" and the assignment operator, the equal sign is treated as a relational operator.

Table 3-1. Statements

STATEMENTS	FUNCTION
LET	Assigns the value of an expression to a variable. The word LET may be omitted.
REM	Introduces remarks and comments in the program listing.
GOTO	Transfers control to a specified statement.
GOTO . . . OF	Multibranch GOTO transfers control to one of a list of statements, depending on the value of an integer expression.
END/STOP	END indicates the last program statement and terminates execution of the current program. STOP terminates execution of the current program.
FOR . . . NEXT	Allows repetition of a group of statements between FOR and NEXT. The number of repetitions is determined by the initial and final values of a FOR variable, and an optional STEP specification.
IF . . . THEN	Evaluates a conditional expression and specifies action to be taken if condition is true.
PRINT	Prints the contents of a list of numeric or string expressions on the list device, or to a specified file.
READ/DATA/RESTORE	Assigns constants and string literals from one or more DATA statements to the variables specified in the READ statement. Treats contents of all DATA statements as a single data list.
INPUT	Requests user input to one or more variables by printing a prompt and accepts string or numeric data from the terminal.
DIM	Defines the size of arrays.
COM	Allows a program to store data in memory for retrieval by a subsequent BASIC program.
PAUSE	Stops program execution without terminating the program.
WAIT	Causes an executing program to stop for a specified number of milliseconds before continuing.

When a variable to be assigned a value contains subscripts, these are evaluated first from left to right, then the expression is evaluated and the resulting value moved to the variable.

If a value is assigned to more than one variable, the assignment is made from right to left. For instance, in the statement A=B=C=2, first C is assigned the value 2, then B is assigned the current value of C, and finally A is assigned the value of B.

Examples

```
10 LET A = 5.02
20 A=5.02
```

The variable A is assigned the value 5.02. Statements 10 and 20 have the same result.

```
30 X = Y7 = Z = Z1 = 0
```

Each variable X, Y7, Z, and Z1 is set to zero. This is a simple method for initializing variables at the start of a program.

```
35 LET M=2
40 LET A(M) = N = 9
```

First M is assigned the value 2 in line 35. In line 40 N is assigned the value 9, then the array element A(2) is assigned the value 9.

```
50 N = 0
60 LET N = N+1
70 LET A(N) = N
```

Statements 50 through 70 set the array element A(1) to 1. By repeating statements 60 and 70, each array can be set to the value of its subscript.

3-2. REM

This statement allows the insertion of a line of remarks in the listing of the program. The remarks do not affect program execution.

Format

REM *any characters*

Like other statements, REM must be preceded by a statement number.

The remarks introduced by REM are saved as part of the Real-Time BASIC program, and printed when the program is listed or punched. They are, however, ignored when the program is executed.

Remarks are easier to read if REM is followed by spaces, or a punctuation mark as in the examples.

Examples

```
>LIST
   10  REM: THIS IS AN EXAMPLE
   20  REM: OF REM STATEMENTS.
   30  REM -- ANY CHARACTERS MAY FOLLOW REM: "//**!!&&&,ETC.
   40  REM...REM STATEMENTS ARE NOT EXECUTED
>
```

3-3. GOTO

GOTO overrides the normal sequential order of statement execution by transferring control to a specified statement. The statement to which control transfers must be an existing statement in the current program.

Format

GOTO *statement number label*

GOTO *integer expression* OF *statement number label* [, *statement number label*, . . .]

GOTO may have a single *statement number label*, or may be multi-branched with more than one label. If the multi-branch GOTO is used, the value of the *integer expression* determines the label in the list to which control transfers. It is rounded to the nearest integer. GOTO may be entered as GO TO.

If the GOTO transfers to a statement that cannot be executed (such as REM or DIM), control passes to the next sequential statement after that statement. GOTO cannot transfer into or out of a function definition (see Section V). If it should transfer to the DEF statement, control passes to the line following the function definition.

The statement number labels in a multi-branch GOTO are selected by numbering them sequentially starting with 1, such that the first label is selected if the value of the expression is 1, the second label if the expression equals 2, and so forth. If the value of the expression is less than 1 or greater than the number of labels in the list, then the GOTO is ignored and control transfers to the statement immediately following GOTO.

Examples

```
50  GOTO 100
60  GOTO A OF 100, 200, 300
```

The first statement sends the sequence of execution to line number 100. The second statement directs control to either line number 100, 200, or 300 depending on the current value of A

The example below shows a simple GOTO in line 200 and a multi-branch GOTO in line 600.

```
>LIST
   100   LET I=0
   200   GOTO 600
   300   PRINT I
   400   REM THE VALUE OF I IS ZERO
   500   LET I=I+1
   600   GOTO I+1 OF 300,500,800
   700   REM THE FINAL VALUE OF I IS 2
   800   PRINT I
   900   END
>RUN
0
2
```

When run, the program prints the initial value of I and the final value of I.

3-4.   END/STOP

The END and STOP statements are used to terminate execution of a program.

Format

```
END

STOP
```

The END statement consists of the word END; the STOP statement of the word STOP.

END and STOP have identical functions; the only difference is that the highest numbered statement in a program must be an END statement. STOP may be used simply to halt program execution at a given point.

Examples

```
200 IF A # 27.5 THEN 350

   •

   •

300 STOP

   •

350 LET A = 27.5

   •

   •

500 IF B # A THEN 9999

   •

   •

9999 END
```

3-5.   FOR . . . NEXT

The looping statements FOR and NEXT allow repetition of a group of statements. The FOR statement precedes the statements to be repeated, and the NEXT statement directly follows them. The number of times the statements are repeated is determined by the value of a simple numeric variable specified in the FOR statement.

Format

FOR *variable = initial expression* TO *final expression* [STEP *step expression*]

The *variable* is set to the value resulting from the *initial expression*. When the value of the *variable* passes the value of the *final expression*, the looping stops. If STEP is specified, the *variable* is incremented by the value resulting from the *step expression* each time the group of statements is repeated. This value can be positive or negative, but should not be zero. If a *step expression* is not specified, the variable is incremented by 1.

The NEXT statement terminates the loop:

NEXT *variable*

The *variable* following NEXT must be the same as the *variable* after the corresponding FOR.

When FOR is executed, the variable is assigned an initial value resulting from the expression after the equal sign,and the final value and any step value are evaluated. Then the following steps occur:

1. The value of the FOR variable is compared to the final value; if it exceeds the final value (or is less when the STEP value is negative), control skips to the statement following NEXT.

2. All statements between the FOR statement and the NEXT statement are executed.

3. The FOR variable is incremented by 1, or if specified, by the STEP value.

4. Return to step 1.

Your program should not execute the statements in a FOR loop except through a FOR statement. Transferring control into the middle of a loop can produce undesirable results.

FOR loops can be nested if one FOR loop is completely contained within another. They must not overlap.

Examples

Each time the FOR statement executes, a value for R is entered and the area of a circle with that radius is computed and printed.

```
>LIST
  10   REM : RADIUS EXAMPLE
  20      FOR A=1 TO 5
  30      INPUT R
  40      PRINT "AREA OF CIRCLE WITH RADIUS ";R;" IS ";3.14159*R↑2
  50      NEXT A
  60   END
>RUN
?1
AREA OF CIRCLE WITH RADIUS 1          IS 3.14159
?2
AREA OF CIRCLE WITH RADIUS 2          IS 12.5664
?4
AREA OF CIRCLE WITH RADIUS 4          IS 50.2654
?8
AREA OF CIRCLE WITH RADIUS 8          IS 201.062
?16
AREA OF CIRCLE WITH RADIUS 16         IS 804.247

BASIC READY
```

The FOR loop executes six times, decreasing the value of X by 1 each time:

```
>LIST
  10      FOR X=0 TO -5 STEP -1
  20      PRINT X-5
  30      NEXT X
  40    END
>RUN
-5
-6
-7
-8
-9
-10

BASIC READY
>
```

3-6.   IF . . . THEN

IF . . . THEN statements are used to test for specified conditions and to specify program action depending on the test results. When a condition is found by the program to be true, then program action indicated by the statement is performed. When a condition is found by the program to be untrue, program action simply continues to the next statement.

Format

IF *expression* THEN $\begin{array}{l}\textit{statement}\\\textit{statement number label}\end{array}$

The IF . . . THEN statement relationship is often described as a conditional transfer. Possible statement transfers that may be used with the IF . . . THEN condition are:

```
IF. . . CALL
        CHAIN
        GOSUB
        GOTO
        INPUT
        LET
        PAUSE
        PRINT
        PRINT #
        READ
        READ #
        RESTORE
        RETURN
        STOP
        WAIT
```

The word THEN is omitted from the statement in the above operations.

3-8

Because numbers are not always represented exactly in the computer, the = operator should be used carefully in IF . . . THEN statements. Whenever possible, < = or > = should be used instead of =.

Examples

```
10 IF A=B THEN 30


10 IF A=B PRINT C
```

In the following example, if X > 10, the message in statement 40 is executed. Otherwise, the message in statement 60 is executed. Note that the relational operator is optional in logical evaluations.

```
>LIST
  10   LET N=10
  20   READ #1;X
  30   IF X <= N THEN 60
  40   PRINT "X IS MORE THAN";N
  50   GOTO 80
  60   PRINT "X IS LESS THAN OR EQUAL TO";N
  70   GOTO 20
  80   END
```

3-7.   PRINT

PRINT causes data to be output at the terminal. The data to be output is specified in a print list following PRINT.

Format

PRINT [*print list*]

The *print list* consists of items separated by commas or semicolons. The list may be followed by a comma or a semicolon. If the list is omitted, PRINT causes a skip to the next line. Items in the list may be numeric expressions, numeric or string variables, string literals, or tabbing functions.

The contents of the print list is printed. If there is more than one item in the print list, commas or semicolons must separate the items. The choice of a comma or semicolon affects the output format.

The output line is divided into six consecutive fields: five of 15 characters and one of 5 characters, for a total of 80 characters. The fields begin in columns 1, 16, 31, 46, 61 and 76. When a comma separates items, each item is printed starting at the beginning of a field. When a semicolon separates items, each item is printed immediately following the preceding item. In either case, if there is not enough room left in the line to print the entire item, printing of the item begins on the next line.

Statements

The separator between items can be omitted if one or both of the items is a quoted string. In this case, a semicolon is inserted automatically.

A carriage return and linefeed are output after PRINT has executed, unless the output list is terminated by a comma or semicolon. In this case, the next PRINT statement begins on the same line.

If an expression appears in the print list, it is evaluated and the result is printed. Any variable must have been assigned a value before it is printed. Each character between quotes in a string constant is printed.

See Section VII, Files, for information about other forms of the PRINT statement.

Examples

When items are separated by commas, they are printed in up to five fields per line; separated by semicolons, they directly follow one another. In the example below, the items are numeric, so each item is assigned a minimum of six characters.

```
>LIST
   10   LET  A=B=C=D=E=15
   20   LET  A1=B1=C1=D1=E1=20
   30   PRINT  A,B,C1,C
   40   PRINT  A;B;C1;C;D;E;A1;D1;E1
   50   PRINT  A,B;C,D
   60   END
>RUN
15                  15                  20                  15
15       15       20       15       15       15       20       20       20
15                  15       15       15
```

In the example below, the first PRINT statement evaluates and then prints three expressions. The second PRINT skips a line. The third and fourth PRINT statements combine a string constant with a numeric expression. No fields are used in the print line for string constants unless a comma appears as separator. The fourth PRINT statement prints output on the same line as the third because the third statement is terminated by a comma.

```
>LIST
   10   LET  A=B=C=D=E=15
   20   LET  A1=B1=C1=D1=E1=20
   30   PRINT  A*B,B/C/D1+30,A+B
   40   PRINT
   50   PRINT  "A*B =";A*B,
   60   PRINT  "THE  SUM  OF  A  AND  B  IS ";A+B
   70   END
>RUN
225                 30.05               30

A*B =225            THE  SUM  OF  A  AND  B  IS  30
```

3-7a. PRINT USING

PRINT USING statements are used to direct output to specific devices and to format that output. The LU# of the output device is specified in the statement. The format of the data to be output is specified in the format part of the string following PRINT USING, and the data is specified in a print list following the format part.

Format

PRINT [#*lu*] USING format part [;*using list*]

Parameter

lu   an optional numeric constant or variable which may be the LU# of the device, or the type 0 file in the sixth position in the FILES statement. "*lu*" may not be a standard file.

format part   either a format string (represented as a literal string or a string variable) or a statement number referencing an IMAGE statement.

using list   an optional parameter list which may include; numeric expressions, string variables, or print function (TAB), separated by commas. These commas are delimiters only and have no formatting function. Note that string variables, not string expressions, are allowed.

NOTE

Use of the carriage control operators #, + and − on non-terminal devices such as line printers may produce results that are different from those on a terminal. This is caused by mechanical or programatic differences in the hardware or software.

GROUPS. A *group* of one or more *format specifications* may be enclosed in parentheses which must be preceded by a *repetition factor* between 1 and 72 inclusive (e.g., 2(/AX,D/) is equivalent to /AX,D/ //AX,D/). Within the parentheses, the specifications must be separated by commas or slashes and the *group must be set off from other specifications by a comma or slashes, just as if it were a single specification. Groups can be nested two levels deep.*

EXECUTION OF THE PRINT USING STATEMENT. Execution of the PRINT USING statement commences by examining the *format string*. The *carriage control* character, if present, is noted for termination processing, then each *format specification* is examined.

If the specification is either a string or a numeric specification, the next item from the *using list* is printed according to the specification. If the *using list* has already been exhausted or is not present, the statement terminates. If the item does not agree with the specification (i.e., string vs. numeric), an error message is printed and the program execution terminates.

If the specification is literal, the specified number of blanks (or the contents of the literal string) is simply printed; the *using list* is not examined.

If the the end of the *format string* is reached before the end of the *using list*, processing continues from the beginning of the *format string* but after the optional *carriage control* character (if the *format string* contains no string or numeric specifications, the statement terminates).

When all items from the *using list* have been printed the statement terminates (any remaining literal specifications are processed if the end of the *format string* has not been reached for the first time). Termination consists of printing, carriage return, and linefeed, modified by the *carriage control* character.

If the *format string* is empty or contains only blanks, output consists of only a carriage return, and linefeed.

Examples

10   PRINT #6 USING 100A$, (2+X), B$

        •

        •

        •

100   IMAGE 3DX,DD.D//SD.DDE

50   PRINT #17 USING "#,3(DD.D2X)";Z1,Z2,Z3
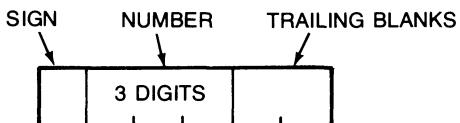
See Chapter 19 for additional information on formatted output.

3-8.   NUMERIC OUTPUT FORMATS
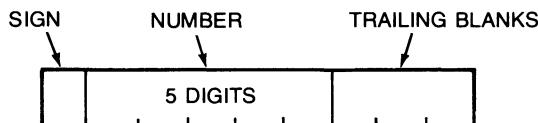
Numeric quantities are left justified in a field whose width is determined by the magnitude of the item. The width includes a position at the left of the number for a possible sign and at least one position to the right containing blanks. The width is always a multiple of three; the minimum width is six characters.

Integers

An integer with a magnitude less than 1000 requires a field width of six characters:

SIGN   NUMBER   TRAILING BLANKS

```
3 DIGITS
```

An integer with a magnitude between 1000 and 32767 inclusive requires a field width of nine characters:

SIGN   NUMBER   TRAILING BLANKS

```
5 DIGITS
```

Examples of integers:

The integers below are less than 1000 and greater than −1000:

```
>LIST
  10    PRINT 1;999;30;-300;+295
  20    END
>RUN
1       999    30      -300   295
```

These integers are between 1000 and 32767 or between −1000 and −32767:

```
>LIST
  10    PRINT 1000;+32751;-32767;32767
  20    END
>RUN
1000         32751      -32767     32767
```

These integers are mixed in magnitude, but none are greater than 32767 or less than −32767:
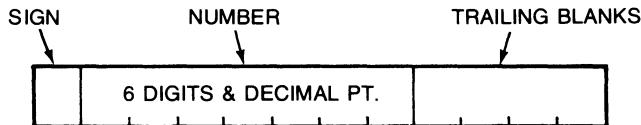
```
>LIST
  10    PRINT 1;1000;999;+32751;20;-32767;-300;25687;+286;5000
  20    END
>RUN
1       1000      999    32751      20      -32767     -300   25687      286
5000
```

If an integer has a negative sign it is printed; a positive sign is not printed.

Fixed-Point Numbers

A fixed point number requires a field width of 12 positions. If the magnitude of the number is greater than or equal to .09999995 and less than 999999.5, or is less than .1 but can be printed with six significant digits, the number is printed as a fixed-point number with a sign. Trailing zeros are not printed, but a trailing decimal point is printed to show the number is not exact. The number is left-justified in the field with trailing blanks. The sign is printed only if it is negative.
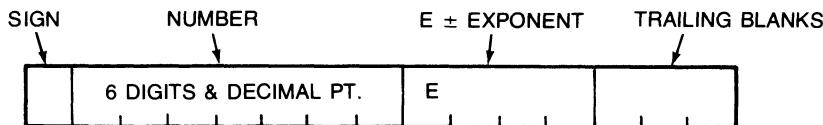
Examples of fixed-point numbers:

```
>LIST
   10   PRINT 999999.;.1;.000044
   20   END
>RUN
999999.      .1              .000044
```

Floating-Point Numbers

Any number, integer or fixed-point, with a magnitude greater than the magnitude of the numbers presented above, is printed as a floating-point number using a total field width of 15 positions:

Examples of floating-point numbers:

```
BASIC READY
>10 PRINT 2345673;.0000044
>20 END
>RUN
2.34563E+06      4.40000E-06

BASIC READY
>10 PRINT 23456789;.00000044
>20 END
>RUN
2.34563E+07      4.40000E-07

BASIC READY
>10 PRINT .00003943;.0000257895
>20 END
>RUN
3.94300E-05      2.57895E-05
```

3-9.   TAB FUNCTION

The TAB function moves the print position to a specified column.

Format

TAB *(integer expression)*

The print position is moved to the column specified by the *integer expression*. Print positions are numbered from 0 to 71. If the print position must be moved to the left because the *integer expression* is less than the current position, nothing is done. If the expression is greater than 71, the print position is moved to the beginning of the next line.

3-10.   READ/DATA/RESTORE

Together, the READ, DATA, and RESTORE statements provide a means to input data to a BASIC program. The READ statement reads data specified in DATA statements into variables specified in the READ statement. RESTORE allows the same data to be read again.

Format

READ *variable list*

DATA *constant* [, *constant*, ...]

RESTORE [*statement number label*]

Parameters

variable list	list of variables separated by commas.
constant	numeric or string constant.
statement number label	identifies a DATA statement.

Constants in the DATA statement are assigned to variables in the READ statement according to their order; the first constant to the first variable, and so forth.

When a READ statement is executed, each variable is assigned a new value from the constant list in a DATA statement. RESTORE allows the first constant to be assigned again when READ is next executed or, if a label is specified, the first constant in the specified DATA statement.

More than one DATA statement can be specified. All the constants in the combined DATA statements comprise a data list. The list starts with the DATA statement having the lowest statement label and continues to the statement with the highest label. DATA statements can be anywhere in the program; they need not precede the READ statement, nor need they be consecutive. DATA statements do not execute, but merely specify data.

If a variable is numeric, the next item in the data list must be numeric; if a variable is a string, the next item in the data list must be a string constant. It is possible to determine the type of the next item with the TYP function (see Section V).

A pointer is kept in the data list showing which constant is the next to be assigned to a variable. The RUN command sets the pointer at the first DATA statement. It is advanced consecutively through the data list as constants are assigned. The RESTORE statement can be used to access data constants in a non-serial manner by specifying a particular DATA statement to which the pointer is to be moved.

When the RESTORE statement specifies a label, the pointer is moved to the first constant in the specified statement. If the statement is not a DATA statement, the pointer is moved to the first following DATA statement. When no label is specified, the pointer is restored to the first constant of the first DATA statement in the program.

Examples

The data in statement 10 is read in statement 20 and printed in statement 30:

```
>LIST
   10   DATA 3,5,7
   20   READ A,B,C
   30   PRINT A,B,C
   40   END
>RUN
3                 5                 7
```

Note the use of RESTORE in this example. It permits the second READ to read the same data into a second set of variables:

```
>LIST
   10   DIM A$[3],B$[3]
   20   DATA 3,5,7
   30   READ A,B,C
   40   READ A$,B$
   50   DATA "ABC","DEF"
   60   RESTORE
   70   READ D,E,F
   80   PRINT A$;B$,A;B;C;D;E;F
   90   END
>RUN
ABCDEF          3     5     7     3     5     7
```

3-11.   INPUT

The INPUT statement allows you to input data to your program from the terminal.

Format

INPUT *variable list*

Parameters
variable list list of variables separated by commas.

The INPUT statement requests data to be input from your terminal for subsequent assignment to a variable. When the INPUT statement is encountered, the program comes to a halt and a question mark is printed on the terminal. The program does not continue execution until the input requirements are satisfied.

Only one question mark is printed for each INPUT statement. The statements:

```
10 INPUT A, B2, C5, D, E, F, G
```

and

```
20 INPUT X
```

each cause a single question mark to be printed. Note that the question mark generated by statement 10 requires seven input items, separated by commas, while that generated by statement 20 requires only a single input item. Failure to include commas between input items will result in corrupt input data.

When you run the program, if you enter data of the wrong type or other invalid input, two question marks (??) are printed. You may then type the correct input data.

If you want to terminate the program and return control to the BASIC Interpreter, type Control Q (Q^c). If you are running from a multi point terminal, type QQ.

Example

```
>LIST
  10     FOR M=1 TO 2
  20     INPUT A
  30     INPUT A1,B2,C3,Z0,Z9,E5
  40     PRINT "WHAT VALUE SHOULD BE ASSIGNED TO R ";
  50     INPUT R
  60     PRINT A;A1;B2;C3;Z0;Z9;E5;"R= ";R
  70     NEXT M
  80 END
>RUN

?1
?2,3,4,5,6,7
WHAT VALUE SHOULD BE ASSIGNED TO R ?27
1      2      3      4      5      6      7      R= 27
?1.5
?2.5,3.5,4.5,6.,7.2
?8.1
WHAT VALUE SHOULD BE ASSIGNED TO R ?-99
1.5          2.5          3.5          4.5          6      7.2
8.1          R= -99
```

3-12. DIM

The DIM (dimension) statement defines the size of an array. DIM statements may also be used with strings (see Section IV).

Format

DIM *X(integer)*[, ...]

DIM *X(integer,integer)*[, . . .]

Parameters

X   array name (A through Z)

integer   dimension of array. (The first *integer* refers to rows and the second to columns).

The DIM statement defines the size of an array. 255 is the maximum dimension allowed. If a variable is subscripted and has not been defined in a DIM or COM statement, the size of the array is assumed to be 10. If the reference is to a two dimensional array, the array is assumed to be 10 by 10. An array may be dimensioned only once. More than one array can be named in a DIM statement; they are separated by commas.

There is no requirement to use all of the space reserved when you define the array. The maximum array size depends only upon the maximum available memory in the computer. The DIM statement can appear anywhere in a program and is not executed.

There is no way to initialize an array before execution. Values must be loaded by FOR loops or by reading from peripheral devices.

Examples

```
>LIST
  10   DIM F[2,3]
  20     FOR I=1 TO 2
  30       FOR J=1 TO 3
  40       LET F[I,J]=1
  50       NEXT J
  60     NEXT I
  70   END
>RUN
```

The size of the F array is defined and the array is initialized to contain all ones.

3-13. COM

The COM statement is used to pass data values between programs. Variables specified in a COM statement are placed in a common area so that values assigned to these variables in one program will be retained when transferring to another program with CHAIN. COM areas must be equal in size for CHAINed or INVOKE'd programs so that common will remain properly aligned.

Format

COM *variable list*

Parameter *variable list*   list of string or array variables

---

COM is an array which is placed in a known fixed location in memory. Upon completion of the first program and the loading of the second program, the first location in the COM area is aligned with the first location of the second load.

Numeric bounds for arrays and strings are specified as in a DIM statement. Because a variable cannot be defined in two places at once, if the variable appears in a COM statement, it cannot also be defined in a DIM statement. An example of how the COM statement might appear in two successive programs follows.

	First Program 10 COM A(7)	Second Program 10 COM D(1),C(2),B(4)
Position in Memory	First Program	Second Program
xxx1	A(1)	D(1)
xxx2	A(2)	C(1)
xxx3	A(3)	C(2)
xxx4	A(4)	B(1)
xxx5	A(5)	B(2)
xxx6	A(6)	B(3)
xxx7	A(7)	B(4)

Remember, it is your responsibility to ensure proper access of common areas.

Common areas are not initialized to UNDEFINED as arrays declared in DIM statements are. You must not use Common area arrays before initialization or your results will be erroneous.

3-14. PAUSE

The PAUSE statement is used to stop the execution of a program without terminating the program.

Format

PAUSE [n]

Parameter

n   optional parameter. If used, the number n will be printed after PAUSE when the statement is executed.

The PAUSE statement stops a running program without terminating it, that is, without sending it to end of program. When a PAUSE statement is encountered and executed, the program is halted and the PAUSE is printed on the terminal. If you wish the program to continue, type GO, otherwise type Control Q (Q^c) thereby instructing the program to terminate and returning control to the BASIC Interpreter. BASIC is unable to execute real-time tasks during the time that a program is halted by a PAUSE statement.

3-15. WAIT

The WAIT statement is used to introduce a program delay. When a WAIT statement is encountered, program execution is stopped for the number of milliseconds specified, then continued automatically.

Format

WAIT *(number of milliseconds)*

The WAIT statement introduces a program delay which allows instruments to achieve a steady state. The number following the word WAIT is the desired delay in milliseconds. Hence the statement:

WAIT (1000)

will delay the program one full second. The range of the number of milliseconds that the program can wait is from 0 to 32767: the maximum delay is therefore 32.767 seconds.

The time delay produced by WAIT is not precise.

Example

```
>LIST
   10   LET Y=5000
   20   LET Z=1
   30   PRINT #Z;"STATEMENT 20"
   40   WAIT (Y)
   50   PRINT #Z;"STATEMENT 40"
   60   GOTO 20
   70   END
>RUN
```

A string is a set of characters delimited with quotation marks, such as "DDDDDE" or 45T,#". Real Time Multi-User BASIC contains special variables and language elements for manipulating string quantities. This section explains how to use the string features of BASIC. There is little difference in the form of statements referencing numeric quantities and those referencing strings. One important difference, however, is the use of subscripts which is explained later.

Lower-case alphabetic characters can be input from or output to user terminals having this capability. When lower-case characters are output to a terminal not capable of printing them, most terminals will print such characters as the upper-case equivalent. Lower-case characters are automatically converted to upper-case by the system, except when they occur in strings or REM statements. Lower-case characters in strings used as file names in ASSIGN statements or program names in CHAIN statements are also converted to upper-case when used.

The examples and comments in this section emphasize modifications in statement form or other special considerations in handling strings.

If you are familiar with the concepts "string", "string variable", and "substring", skip directly to paragraph 4-5.

4-1.   STRING

A string is a set of characters enclosed by quotation marks or the null string (no characters). See paragraph 4-5 for detailed information about string dimensions.

Typical Strings: "ABCDEFGHIJKLMNOP"

"12345"

"BOB AND TOM"
"MARCH 13, 1970"

Null String: " "

Quotation marks cannot be used within a string because quotation marks are used as string delimiters.

Apostrophes and control characters are legal as string characters.

A null string has no value, as distinguished from a blank space which has a value.

4-1

Strings are manipulated in string variables. For example:

100 A\$ = "THIS IS A STRING"
 ↑   ↑
 string   *string*
 variable

200 B\$ = A\$(1,10)
 ↑   ↑
 string   *substring*
 variable *(defined later)*

300 C\$ = ""
 ↑   ↑
 string   *null string*
 variable

4-2. STRING VARIABLE

A string variable consists of a single letter (A to Z) followed by a \$, and is used to store strings. A\$,Z\$,M\$ are typical string variables.

String variables must be declared before being used if they contain strings longer than one character. See the String DIM statement, paragraph 4-5. When a string variable is declared, its "physical" length is set. The "physical" length is the maximum size string that the variable can accommodate. For example:

```
710 DIM A$(72),B$(20),C$(50)
```

During execution of a program, the "logical" length of a string variable varies. The "logical" length of the variable is the actual number of characters that the string variable contains at any point. For example:

```
100   DIM A$[72]                    Sets physical length of A$
200   LET A$="SAMPLE STRING"        Logical length of A$ is 13
300   LET A$="LONGER SAMPLE STRING" Logical length of A$ is now 20
```

4-3. SUBSTRING

A substring is a single character or a set of contiguous characters from within a string variable. The substring is defined by a subscript string variable.

A substring is defined by subscripts placed after the string variable. Characters within a string are numbered from the left starting with one. Subscripts must be positive, non-zero, and less than 256. Non-integer subscripts are rounded to the nearest integer.

Two subscripts, separated by a comma, specify the first and last characters of the substring. For example:

```
100   DIM Z$[72]
200   LET Z$="ABCDEFGH"
300   PRINT Z$[2,6]
```

prints the substring

BCDEF

A single subscript specifies the first character of the substring and implies that all characters following are part of the substring. This convention also holds true if there is a zero specified for the second subscript. For example:

```
300   PRINT Z$[3]
```

prints the substring

CDEFGH

Two equal subscripts specify a single character substring. For example:

```
>300   PRINT Z$(2,2)
```

Prints the substring

B

If subscripts specify a substring larger than the physical length of the original string, blanks are appended.

4-4.   STRINGS AND SUBSTRINGS

A string can be made into a null string. This is done by assigning it the value of a substring whose second subscript is one less than its first. For example:

```
100 A$ = B$(6,5)                        A$ now contains a null string.
```

This is the only case in which a smaller second subscript is acceptable in a substring.

Substrings can become strings. For example:

```
100 A$ = "ABCDEFGH"
200 B$ = A$(3,5)
300 PRINT B$
```

Strings

prints the string

    CDE

because the substring of A$ is now a string in B$.

Substrings can be used as string variables to change characters within a larger string. For example:

```
100 A$ = "ABCDEFGH"
200 A$(3,5) = "123"
300 PRINT A$
```

prints the string

    AB123FGH

Strings, substrings, and string variables can be used with relational operators. They are compared and ordered as entries are in a dictionary. See Appendix C for the ranking of non-alphabetic characters. For example:

```
100 IF A$ = B$ THEN 2000
200 IF A$ <= "TEST" THEN 3000
300 IF A$(5,6) >= B$(7,8) THEN 4000
```

See the STRING IF statement description in this section.

4-5.   STRING DIM

Format

DIM *string variable (number of characters in string)*

The string DIM statement reserves storage space for strings longer than 1 character; also for arrays.

The number of characters specified for a string in its DIM statement must be expressed as an integer from 1 to 255.

Each string having more than 1 character must be mentioned in a DIM statement before it is used in the program.

Strings not mentioned in a DIM statement are assumed to have a length of 1 character.

The length mentioned in the DIM statement specifies the maximum number of characters which may be assigned; the actual number of characters assigned may be smaller than this number. See the LEN Function, paragraph 4-11, for further details.

4-4

Array dimension specifications may be used in the same DIM statement as string dimensions (example statement 45 below).

Example

```
35 DIM A$(255), B$(60)
40 DIM Z$(10)
45 DIM N$(2), R(5,5), P$(8)
```

4-6.   STRING ASSIGNMENT

Format

$$[LET] \quad \begin{matrix} string\ variable \\ substring\ variable \end{matrix} = \begin{matrix} \text{"string literal"} \\ string\ variable \\ substring\ variable \end{matrix}$$

The string assignment statement establishes a value for a string; the value may be a literal value in quotation marks, or a string or substring value.

An input line may contain a total of 80 characters. One string assignment statement is allowed per input line. Thus, a string may contain 80 characters minus assignment statement elements such as the statement number, string variable name, equal sign, quotation marks, and any blanks inserted between these elements. String variables having more than 1 character must be mentioned in a DIM statement (see Paragraph 4-5).

Special purpose characters, such as A^c, H^c, D^c, Y^c or quotation marks may not be string characters.

If the source string is longer than the destination string, the source string is truncated at the appropriate point.

Example

```
200 LET A$ = "TEXT OF STRING"
210 B$ = "*** TEXT !!!"
220 LET C$ = A$(1,4)
230 D$ = B$(4)
240 F$(3,3)=N$
```

Strings must be concatenated if they are to include more characters than the limit imposed by the 80-character line. For example:

```
5   DIM A$(255)
100  A$(1,50) = "50 characters"
110  A$(51,100) = "50 characters"
120  A$(101,150) = "50 characters"
130  A$(151,200) = "50 characters"
140  A$(201,255) = "55 characters"
```

4-7.   STRING INPUT

Format

```
INPUT  string variable
       substring variable · · ·
```

The string INPUT statement allows string values to be entered from the user terminal.

Placing a single string variable in an INPUT statement allows the string value to be entered without enclosing it in quotation marks.

If multiple string variables are used in an INPUT statement, each string value must be enclosed in quotation marks, and the values separated by commas. The same convention is true for substring values. Mixed string and numeric values must also be separated by commas. Failure to include the quotation marks and/or commas will result in corrupt input data.

If a substring subscript extends beyond the boundaries of the input string, the appropriate number of blanks are appended.

Numeric variables may be used in the same INPUT statement as string variables (example statement 55 below).

Example

```
50 INPUT R$
55 INPUT A$,B$, C9,   D8
60 INPUT A$(1,5)
65 INPUT B$(3)
```

4-8.   PRINTING STRINGS

Format

```
PRINT  string variable     [  string variable          ]
       substring variable  [,  substring variable  , · · ·]
```

A string PRINT statement causes the current value of the specified string or substring variable to be output to the user's terminal device. The terminal device may be any ASCII output device.

String and numeric values may be mixed in a PRINT statement (example statements 115 and 125 below).

Specifying only one substring parameter causes the entire substring to be printed. For instance, in the example below, if the value of B3 = 642 and C$ = "WHAT IS YOUR NAME?", example statement 120 prints:

WHAT IS

while statement 115 prints

YOUR NAME?END OF STRING 642

Numeric and string values may be "packed" in PRINT statements without using a "semicolon", as in example statement 115.

Example

```
105 PRINT A$
110 PRINT A$, B$, Z$
115 PRINT C$(8) "END OF STRING" B3
120 PRINT C$(1,7)
125 PRINT "THE TOTAL IS: ";X5
```

Each string PRINT statement will accept a maximum of 80 characters. If the strings and string variables in a PRINT statement total more than 80 characters, the line printer will truncate after 80 characters, with no error message returned.

4-9.   READING STRINGS

Format

$$\text{READ}\quad \begin{array}{l} string\ variable \\ substring\ variable \end{array} \quad \left[\begin{array}{l} , \\ , \end{array} \begin{array}{l} string\ variable \\ substring\ variable \end{array} \quad , \quad \cdots \right]$$

A string READ statement causes the value of a specified string or substring variable to be read from a DATA statement.

A string variable (to be assigned more than 1 character) must be mentioned in a DIM statement before attempting to READ its value.

String or substring values read from a DATA statement must be enclosed in quotation marks, and separated by commas. See paragraph 4-12 in this section.

Only the number of characters specified in the DIM statement may be assigned to a string. Blanks are appended to substrings extending beyond the string dimensions.

Mixed string and numeric values may be read (example statement 310 below); see TYP (X), paragraph 5-1, for a description of a data type check which may be used with DATA statements.

Example

```
300 READ C$
305 READ X$, Y$, Z$
310 READ Y$(5), A,B,C5,N$
315 READ Y$(1,4)
```

4-10. STRING IF

Format

> IF *string var. relational oper. string var.* THEN *statement number label*
> *statement*

A string IF statement compares two strings. If the specified condition is true, control is transferred to the statement number specified or the statement is executed. Statements allowed with IF are listed in paragraph 3-6.

Strings are compared one character at a time, from left to right; the first difference determines the relation. If one string ends before a difference is found, the shorter string is considered the smaller one.

Characters are compared by their ASCII representation.

If substring subscripts extend beyond the length of the string, null characters (rather than blanks) are appended.

String compares may appear only in IF. . .THEN statements and not in conjunction with logical operators.

Strings may not use Boolean expressions.

Example

```
340 IF C$<D$ THEN 800
350 IF C$>D$ THEN 900
360 IF C$=D$ THEN 1000
370 IF N$(3,5)<R$(9) THEN 500
380 IF A$(10)="END" THEN 400
390 IF A$#B$ PRINT A$
```

4-11. LEN FUNCTION

Format

> *statement type* LEN *(string variable)*. . .

The LEN function supplies the current (logical) length of the specified string, in number of characters.

DIM merely specifies a maximum string length. The LEN function allows you to check the actual number of characters currently assigned to a string variable.

Example

```
469 PRINT LEN(A$)
479 PRINT LEN(X$)
489 PRINT "TEXT"; LEN(A$); B$, C
499 IF LEN(P$) #5 THEN 600
509 LET X$(LEN(X$)+1) = "ADDITIONAL SUBSTRING"


    •
    •
    •
    •


600 STOP
609 PRINT "STRING LENGTH = "; LEN(P$)
```

4-12. STRINGS IN DATA STATEMENTS

Format

DATA *"string literal"* [, *string literal"*. . .]

The DATA statement specifies data in a program (numeric values may also be used as data).

String values must be enclosed by quotation marks and separated by commas.

String and numeric values may be mixed in a single DATA statement. They must be separated by commas (example 520 below).

A DATA statement input line may contain a total of 80 characters. Thus, a string literal may contain 80 characters minus the statement number, the word DATA, quotation marks, any blanks or commas and other string literals included within the input line.

Example

```
500 DATA "NOW IS THE TIME."
510 DATA "HOW", "ARE", "YOU,"
520 DATA 5.172, "NAME?", 6.47,5071
```

4-13.   PRINTING STRINGS ON FILES

Format

```
                                    string variable
    PRINT # filenumber , record number ; substring variable [, . . .]
                                    "string literal"
```

The PRINT # statement prints string or substring variables or string literals on a file.

String and numeric variables may be mixed in a single file or record within a file (example statement 360 below).

The formula for determining the number of words required for storage of a string on a file is:

$$1 + \frac{number\ of\ characters\ in\ string}{2} \quad \text{if the number of characters is even;}$$

$$1 + \frac{number\ of\ characters\ in\ string\ +\ 1}{2} \quad \text{if the number of characters is odd.}$$

If the file number is not equal to a file position as defined in a FILES statement, the output will go to the logical unit of the same number. When printing to the line printer using PRINT #6, BASIC inserts a leading space in column 1. The PRINT USING (Section 3-7a) or PRINT to a type 0 file should be used to control the lineprinter.

Example

```
350 PRINT #5; "THIS IS A STRING."
355 PRINT #3; C$, B$, X$, Y$, D$
360 PRINT #7,3; X$, P$, "TEXT", 27.5,R7
365 PRINT #N,R; P$, N, A(5,5), "TEXT"
```

4-14.   READING STRINGS FROM FILES

Format

```
                                       string variable        string variable
    READ # file number [, record number] ; substring variable  'substring variable  [, . . .]
```

The READ # statement reads string and substring values from a file.

String and numeric values may be mixed in a file and in a READ number statement; they must be separated by commas.

Example

```
450  READ #5;  C$,B$,X$,Y$,D$

460  READ #8;  X$,P$,X,P

470  READ #N;  P$,N,A(5,7)
```

If the file number is not equal to a file position number as defined in a FILES statement, input will be read from the logical unit of that number.

4-15. DECIMAL STRING ARITHMETIC ROUTINES

You may perform arithmetic operations on decimal strings and substrings through BASIC program calls to the Decimal String Arithmetic package. Five of the Decimal String Arithmetic routines are callable from BASIC. These routines are SADD, SSUB, SMPY, SDIV, and SEDIT. Detailed information about these routines is contained in the Decimal String Arithmetic Routines manual. The part number for this manual is listed in the Documentation Map at the front of this manual.

A function is the mathematical relationship between two variables, X and Y, for example, that returns a single value of Y for each value of X. The independent variable is called an argument; the dependent variable is the function value. To illustrate, in the statement:

    100 LET Y = SQR(X)

X is the argument; the function value is the square root of X; and Y takes the value of the positive root.

Two types of functions are used in Multi-User Real-Time BASIC: system defined functions and user-defined functions.

5-1. SYSTEM-DEFINED FUNCTIONS

Real-Time BASIC provides a variety of functions that perform common operations such as finding the sine, taking the square root, or finding the absolute value of a number. The resulting value of a function is always numeric and can be used in the evaluation of an expression. Available system-defined functions are listed below:

ABS(x)   The ABS function gives the absolute value of the expression (x).

ATN(x)   ATN is the arctangent function. ATN returns the angular argument of x in radians adjusted to the appropriate quadrant.

COS(x)   The COS function returns the cosine of x expressed in radians.

EXP(x)   EXP gives the value of the constant e raised to the power of the expression (x).

IERR(x)   This function returns the error code value which may have been set by a user-defined subroutine or function. See Section VI. x is a dummy argument.

INT(x)   The integer function, INT, provides the truncated value of x; $x < 32767$.

LEN(x\$)   Determines length (no. of characters) in character string identified by string variable x\$. See Section IV.

LOG(x)   Gives base 10 logarithm of variable or expression.

LN(x)   LN provides the logarithm of a positive expression to the base e.

OCT(x)   This function prints the octal equivalent of an integer value. The maximum possible range of the returned variable is $0\text{-}177777_8$. If x is outside the range of -32768 to 32767, 77777_8 is returned.

RND(x)   RND generates a random number greater than or equal to zero and less than 1. The argument x may have any value. A sequence of random numbers is repeatable upon each run if the argument is positive. A random sequence can be achieved upon each run by two RND calls. The first call is issued in the negative value for x, called a seed, followed by another call with a positive x. If the same seed is used for each run, the same random sequence is repeated.

SERR(x)   Sets the error code which may be queried with IERR(x). See Section VI.

SGN(x)   SGN returns 1 for $x > 0$, 0 for $x = 0$, and -1 for $x < 0$.

SIN(x)   The SIN function gives the sine of x expressed in radians.

SQR(x)   SQR provides the square root of x. x must be greater than zero.

SWR(x)   The SWR function returns the logical value, one or zero, of the Switch Register bit position specified by x (range = 0 through 15).

TAB(x)   The TAB function is used to advance the print position the number of positions specified by x. x may be equal to 0 through 71. See Section III.

TAN(x)   The TAN function returns the tangent of x expressed in radians.

TIM(x)   The TIM function returns the current minute, hour, day or year.

                    x = 0, TIM(x) = current minutes (0 to 59)
                    x = 1, TIM(x) = current hour (0 to 23)
                    x = 2, TIM(x) = current day (1 to 366)
                    x = 3, TIM(x) = current year (four digits).
                    x = -1, TIM(x) = current seconds (0 to 59)
                    x = -2, TIM(x) = current tens of milliseconds.

TYP(x)   The TYP function determines the type of the next data item in the specified file. The three possible reponses are: 1 = next item is a number, 2 = next item is a character string, 3 = next item is "end of file", 4 = next item is "end of record". If x is zero, the TYP function references the DATA statements and returns the following response: 1 = number, 2 = string, 3 = "out of data" condition.

5-2.   USER-DEFINED FUNCTIONS

A user-defined function is one that you define for use in your program. It is called and used the same way that a system-defined function is. The DEF statement is used to define a new function, that is to equate the function to a mathematic expression.

Format

DEF FNx(y) = *expression*

Parameters

x   stands for a letter (A-Z) that completes the name of the function. Only 26 user-defined functions may be specified: FNA through FNZ.

y   stands for the variable to which the function is to be applied. Any number, string, or variable may be used in this position.

expression   provides a formula such as X*X or X ↑TAN(X). Whenever the function is called in the program, this formula will be evaluated.

Example

```
>10 DEF FNA(Y)=Y/10
>20 PRINT FNA(100)
>30 END
>RUN
10
```

When FNA (100) is called for in statement 20, the formula defined for FNA is evaluated to determine the value printed. Note that the results of the function may be used in computation:

```
35 LET X = FNA(M1) + 14 -FNA(12)
```

An operand in the program may be used in the defining expression, however, such circular definitions as the one below cause infinite looping.

```
10 DEF FNA(Y) = FNB(Y)+1
20 DEF FNB(X) = FNA(X)-1
```

It is often preferable to make use of the same procedure several times within a program. Rather than re-writing the procedure each time it is to be used, you can simply refer to a given segment of code (a subroutine) whenever that segment is needed. The GOSUB/RETURN statement sequence is used when a subroutine is located within your own program. The CHAIN statement is used when you want to execute another program.

There are also times when the inclusion of subroutines outside of your program is desirable. In this case, the CALL statement is required. External subroutines are completely separate from your program and from the BASIC Interpreter. They are disc resident programs or parts of programs, accessed via a memory directory, and must have been specified by a special subroutine configuration process as described in Section XIV.

6-1. GOSUB/RETURN

GOSUB transfers control to the beginning of a simple subroutine. A subroutine consists of a collection of statements that may be executed from more than one location in the program. In a simple subroutine, there is no explicit indication in the program as to which statements constitute the subroutine. A RETURN statement in the subroutine returns control to the statement following the GOSUB statement.

Format

GOSUB *statement number label*

GOSUB *integer expression* OF *statement number label* [, *statement number label*, . . .]

RETURN

GOSUB may have a single *statement number label,* or may be multi-branched with more than one label separated by commas. In a multi-branch GOSUB, the particular label to which control transfers is determined by the value of the *integer expression* which is rounded to the nearest integer. The RETURN statement consists simply of the word RETURN.

A single-branch GOSUB transfers control to the statement indicated by the label. A multi-branch GOSUB transfers to the statement label determined by the value of the integer expression. As in a multi-branch GOTO, if the value of the expression is less than 1 or greater than the length of the list, no transfer takes place.

When the sequence of control within the subroutine reaches a RETURN statement, control returns to the statement following the GOSUB statement. RETURN statements may be used at any desired exit point in a subroutine. There may be more than one RETURN statement per GOSUB.

Within a subroutine, another subroutine can be called. This is known as nesting. When a RETURN is executed, control transfers back to the statement following the last GOSUB executed. Up to 20 GOSUB statements can occur without an intervening RETURN; more than this causes a terminating error.
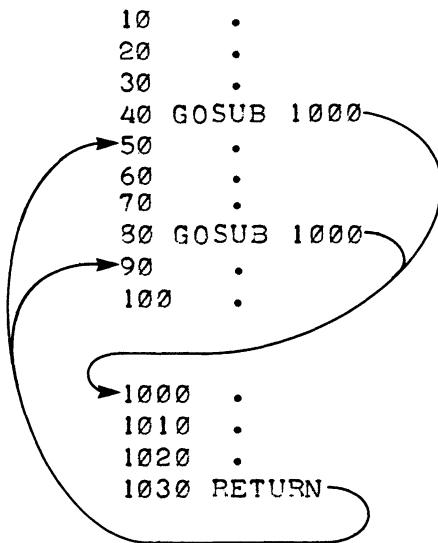
Examples

In the first example, line 20 contains a simple GOSUB statement; the subroutine is in lines 50 through 70, with RETURN in line 70.

```
>LIST
   10   LET B=90
   20   GOSUB 50
   30   PRINT "SINE OF B IS ";A
   40   GOTO 80
   50   REM: THIS IS THE START OF THE SUBROUTINE
   60   LET A=SIN(B)
   70   RETURN
   80   REM: PROGRAM CONTINUES WITH NEXT STATEMENT
   90   END
>RUN
SINE OF B IS .893993
```
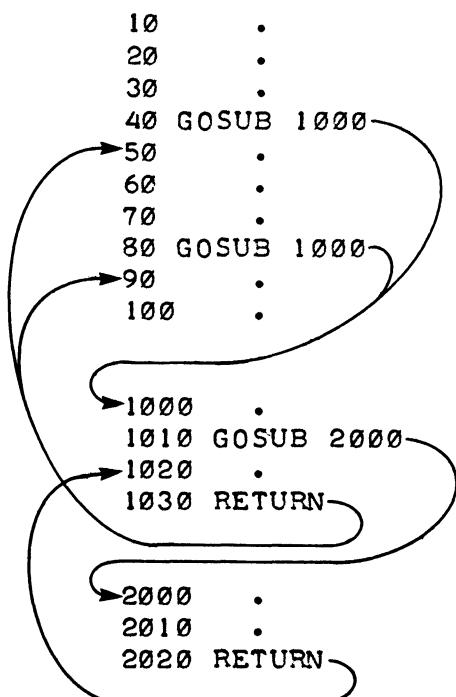
The GOSUB statement can follow the subroutine to which it transfers as in the example below.

```
>LIST
   10   LET B=90
   20   GOSUB 110
   30   REM: THIS IS THE START OF SUBROUTINE
   40   LET A=SIN(B)
   50   RETURN
   60   REM: OTHER STATEMENTS CAN APPEAR HERE
   70   REM: THEY WILL NOT BE EXECUTED
   80   LET A=24
   90   LET B=50
  100   PRINT A;B
  110   GOSUB 30
  120   PRINT A
  130   REM: A SHOULD EQUAL .893993
  140   PRINT B
  150   REM: B SHOULD EQUAL 90
  160   END
>RUN
.893993
90
```

It is also possible for any one subroutine to be called from several places in the coding of any one program. The logical flow of this situation looks like this:

```
10      •
20      •
30      •
40  GOSUB 1000
50  •
60  •
70  •
80  GOSUB 1000
90  •
100     •


1000    •
1010    •
1020    •
1030  RETURN
```

Taking the same situation one step further, it is permissable for a subroutine to, in turn, call another subroutine:

```
10       •
20       •
30       •
40  GOSUB 1000
50   •
60   •
70   •
80  GOSUB 1000
90   •
100      •


1000     •
1010  GOSUB 2000
1020     •
1030  RETURN


2000     •
2010     •
2020  RETURN
```

Subroutines

Subroutines should be entered only with GOSUB statements rather than GO TO's to avoid unexpected RETURN errors (which cause the program to stop execution).

This sequence shows logically nested GOSUB's:

```
10 INPUT
20 GOSUB 100
•
•
•
100 IF C>0 THEN 120
110 LET C=-C
120 GOSUB 200
130 RETURN
•
•
•
200 LET A=SQR(C)
210 LET C=SQR(A)
220 RETURN
300 END
```

The order in which this program is executed is:

when $C > 0$:
10
20
100
120
200
210
220
130
statements after 20

when $C <= 0$:
10
20
100
110
120
200
210
220
130
statements after 20

6-2.   CHAIN

The CHAIN statement terminates the current program and begins execution of another program, optionally starting at a specified statement number.

Format

CHAIN $\begin{matrix} string\ variable \\ string\ literal \end{matrix}$ [, *statement number label*]

The *string variable* or *literal* is the name of a Real-Time BASIC program that is in your library. This may be a fully qualified file name (see Section VII, Files). If the optional *statement number label* is present, execution begins at the first executable statement at or after the label; the exact label need not be present in the called program. If omitted, execution begins at the first executable statement in the called program.

CHAIN calls the program identified by the string expression, and it replaces the current program. When the program called by CHAIN finishes execution, it terminates and does not automatically return to the calling program. The called program may call another program, including the original calling program, with the CHAIN statement.

Chained programs will leave common data files open, thus reducing the overhead needed to close and re-open the file during the next chain operation.

Only variables declared in a COM statement are saved during a CHAIN operation. All variables and arrays of the current program that were not declared in COM are lost when the new program begins execution. All programs must contain the same size COM area and the same number of file positions in the FILES statements so that common will be properly aligned. Files declared in CHAINed programs correspond to files declared in the main program. Logical unit cannot be defined in a CHAINed program unless it is already declared in the main program's FILES statement. Refer to the FILES statement, Section 7-3, for examples.

If the programs are CSAVEd, the time required to execute the CHAIN statement is reduced.

```
>LIST
   10   REM..PROGRAM MAIN
   20   LET X=200
   30   LET A=X'3
   40   PRINT "A= ";A
   50   PRINT "LEAVE MAIN AND ENTER SUBA AT LINE 30"
   60   CHAIN "SUBA",30
   70   END

>LIST
   10   REM..PROGRAM SUBA
   20   PRINT "THIS STATEMENT IS NOT EXECUTED"
   30   PRINT "ENTER SUBA - LINE 30"
   40   LET B=125
   50   LET C=B'2
   60   PRINT "C= ";C
   70   PRINT "END OF SUB A - TERMINATE HERE"
   80   END
>SAVE SUBA
>RUN MAIN
A= 3.00000E+06
LEAVE MAIN AND ENTER SUBA AT LINE 30
ENTER SUBA - LINE 30
C= 15625
END OF SUB A - TERMINATE HERE
```

The main program, MAIN, calls program SUBA with a CHAIN command in line 50. Execution of SUBA begins in line 30, and execution terminates with the last line of SUBA. None of the variable values from MAIN are saved following execution of CHAIN.

6-3.   CALL

The CALL statement is used to identify and execute an external subroutine at a given point within a program. CALL is optional, you may simply use the subroutine name and parameter list. After the subroutine executes, control returns to the statement following the CALL unless there is a FAIL return.

Format

[CALL]*subroutine name(parameter list)* [FAIL: *statement*]

Parameters

subroutine name   name of the routine as entered into the system during system genera-
          tion or when loaded on-line.

parameter list   list of variables or constants to be passed to the subroutine or varia-
          bles into which the subroutine places information for the calling
          program. Spaces are not allowed between the subroutine name and
          the left parenthesis.

FAIL: *statement*   optional subroutine failure return. See paragraph 6-4.

To execute the subroutine calling sequence, you need to determine the following:

* the name of the subroutine
* the number of parameters in the call
* the meaning of the contents of each parameter
* the values acceptable in each parameter.

Usually this information is provided in the documentation supplied with the subroutine.

A CALL statement is rejected by the interpreter unless the TABLES command has been given to specify the legal set of subroutine names.

Examples

Figures 6-1 and 6-2 contain examples of routines written in FORTRAN which may be called from BASIC.

Constant numbers, string literals, and expressions cannot be used as parameter values when calling a subroutine if the parameter is defined as a return variable (type V, see Section XIV).

```
0001  FTN,L,M
0002        INTEGER FUNCTION NUM(I)
0003  C
0004  C
0005  C
0006  C
0007  C THIS FUNCTION RETURNS THE NUMERIC VALUE OF THE FIRST CHARACTER
0008  C OF THE STRING EXPRESSION ACCORDING TO THE STANDARD CHARACTER CODE.
0009  C
0010  C   FOR EXAMPLE:
0011  C
0012  C        10 PRINT NUM("A")
0013  C        20 END
0014  C
0015  C        >RUN
0016  C
0017  C        65
0018  C
0019  C
0020  C
0021  C THE FUNCTION'S DESCRIPTION THAT MUST BE INPUT TO THE TABLE
0022  C GENERATOR TO CREATE THE PROPER ENTRY IN THE BRANCH AND MNEMONIC
0023  C TABLE IS AS FOLLOWS:
0024  C
0025  C        NUM(RA), OV=NN, INTG, ENT=NUM, FIL=FILxx
0026  C
0027  C   WHERE: RA    INDICATES REAL PARAMETER (STRINGS ARE ALWAYS REAL)
0028  C          NN    INDICATES THE OVERLAY NUMBER
0029  C          FILX  INDICATES THE FILE NAME OF THE RELOCATABLE FOR
0030  C                THIS FUNCTION.
0031  C
0032  C
0033  C
0034        DIMENSION I(2)
0035  C
0036  C   RIGHT JUSTIFY CHARACTER BY DIVIDING
0037  C
0038  C   RIGHT HALF OF THE FIRST WORD OF A STRING IS THE CHARACTER COUNT
0039  C   AND MUST NOT BE DISTURBED.
0040  C
0041        NUM =I(2)/256
0042        RETURN
0043        END
```

Figure 6-1. Preparing a FORTRAN Function for Use by BASIC Program

```
0001   FTN,L,M
0002          SUBROUTINE CHRS(I,J)
0003   C
0004   C
0005   C
0006   C
0007   C    THIS SUBROUTINE CAUSES THE NUMERIC VALUE OF THE FIRST PARAMETER
0008   C    TO REPLACE THE FIRST CHARACTER OF THE SECOND PARAMETER WHICH
0009   C    IS A STRING VARIABLE.
0010   C
0011   C
0012   C       FOR EXAMPLE:
0013   C
0014   C           10 DIM A$(10)
0015   C           20 AS="YBCDE"
0016   C           30 CHRS(65,A$)
0017   C           40 PRINT A$
0018   C           50 END
0019   C
0020   C           >RUN
0021   C
0022   C           ABCDE
0023   C
0024   C
0025   C
0026   C    THE FUNCTION DESCRIPTION THAT MUST BE INPUT TO THE TABLE
0027   C    GENERATOR TO CREATE THE PROPER ENTRY IN THE BRANCH AND MNEMONIC
0028   C    TABLE IS AS FOLLOW:
0029   C
0030   C           CHRS(I,RVA), END=CHRS
0031   C
0032   C           WHERE: I     INDICATES AN ITEGER VARIABLE PASSED TO 'CHRS'
0033   C                  RVA   INDICATES A REAL VARIABLE (STRINGS ARE ALWAYS
0034   C                        SPECIFIED AS REAL) RETURNED FROM 'CHRS'.
0035   C
0036   C
0037   C
0038   C
0039   C
0040   C
0041   C
0042          DIMENSION J(2)
0043   C
0044   C    PLACE CHARACTER IN FIRST CHARACTER POSITION OF STRING 'J'
0045   C
0046   C    THE RIGHT HALF OF THE FIRST WORD OF A STRING IS THE CHARACTER
0047   C    COUNT AND MUST NOT BE DISTURBED.
0048   C
0049          J(2)=IAND(J(2),377B)
0050          J(2)=IOR(I*256,J(2))
0051          RETURN
0052          END
```

Figure 6-2. Preparing a FORTRAN Subroutine for Use by BASIC Program

6-4.   THE FAIL ERROR OPTION

Some of the externally defined subroutines supplied with the Real-Time BASIC Interpreter make error checks at execution time. For example, the TRNON routine checks both the time schedule table and the trap table for overflow before adding a new entry. If an execution time error is detected, an appropriate error message is printed, the ERRCD flag is set, the program is aborted, and the BASIC Interpreter returns to command input mode.

You may avoid aborting your program by using the FAIL option as part of the subroutine call statement. Any statement which can appear in an IF statement can be added to the end of a subroutine CALL statement following the word FAIL:.

For example:

        100 CALL TRNON(2000,122536)FAIL: GO TO 9000

If the called subroutine detects an error during execution, the error message is printed but the Interpreter executes the statement following FAIL: instead of aborting the program. The error message format is:

        ERROR n IN LINE xxx where n is the ERRCD value.

If ERRCD equals zero, the FAIL statement is not executed.

The FAIL: option may be used with the following routines:

SETP	
TRNON	Task
START	Control
ENABL	Statements
DSABL	
RDBIT	
RDWRD	
WRBIT	
WRWRD	HP 6940
DAC	Calls
MPNRM	
SENSE	
AISQV	
SGAIN	
RGAIN	HP 2313
AOV	Calls
NORM	
PACER	

You can use the FAIL ERROR option in subroutines you write for yourself, as well as in the HP subroutines listed above. In your own subroutines you also have the option of processing errors without printing out the error messages. There is no way of avoiding the error message when you are using the HP subroutines, because these routines call an internal error message routine.

After execution of a CALL statement, in either an HP subroutine or your own, the interpreter checks the value of ERRCD. If this value is non-zero, and you have not included the FAIL ERROR option, subroutine CALSB prints the error message:

SUB. OR FUNCT. TERMINATED ABNORMALLY IN LINE *xxx*

and the program aborts. Subroutine CALSB is the parameter-passing linkage between BASIC overlays and the subroutine you write in BASIC language. The loader attaches CALSB to your BASIC routine.

If you do not, in your subroutine, set a non-zero value in ERRCD, there will be no error message and no abort. If, on the other hand, you do set ERRCD non-zero, you will get the above error message, and the program will abort. If you set ERRCD non-zero, and also include the FAIL ERROR option in your CALL statement, the program will not abort and you can additionally interrogate ERRCD in your program:

```
10 CALL subroutine FAIL: GO TO 100
       .
       .
       .
100 I = IERR(0)
110 PRINT I
```

In statement 100, IERR is the function that interrogates ERRCD. The PRINT statement allows you to check what may have caused the error in the execution of your routine.

The following is a sample subroutine you might use to set a value in ERRCD:

```
ASMB,R,L
        NAM PASS,7          Subroutine to set ERRCD
        ENT PASS
        EXT .ENTR,ERRCD
ICODE   BSS 1
PASS    NOP
        JSB .ENTR
        DEF ICODE
        LDA ICODE,I
        STA ERRCD
        JMP PASS,I
        END
```

Alternatively, you can use subroutine ERROR to print a BASIC-type error message:

```
ASMB,R,L
        NAM PASS,7          Subroutine to set ERRCD
        ENT PASS
        EXT .ENTR,ERRCD,ERROR
ICODE BSS 1
PASS  NOP
        JSB .ENTR
        DEF ICODE
        LDA ICODE,I
        STA ERRCD
        JSB ERROR
        DEF *+3
        DEF ERRCD
        DEF MESS
        JMP PASS,I
MESS  DEC 4
        ASC 2,TEST
        END
```

Using the ERROR subroutine prints an error message of the form:

ERROR TEST–*number* IN LINE *xxx*

where *number* is the value of ERRCD. The following is a FORTRAN subroutine that uses PASS, and a sample BASIC program to call it:

```
FTN4,L
        SUBROUTINE TRYIT (IFLAG,ICODE)
C IF IFLAG #0, THEN SET ERRCD TO ICODE
        IF (IFLAG.EQ.0) GO TO 900
        CALL PASS (ICODE)
900     RETURN
        END

10 PRINT "INPUT A,N (A#0: TAKE ERR EXIT/N=ERRCD)";
20 INPUT A,N
30 CALL TRYIT (A,N) FAIL: GO TO 50
40 STOP
50 PRINT "ERRCD IS NOW ";IERR(0)
60 END
```

Notice that ERRCD is examined in BASIC with the function IERR, which has a single dummy parameter, not used but necessary. Also notice that ERRCD may be set in the BASIC program by the function SERR, which also has a single dummy parameter.

6-5.   THE IERR FUNCTION

Since the action desired may depend on which error occurred, the function IERR is supplied to interrogate the ERRCD flag. It is a BASIC function and must be used as an operand in an expression. It returns the value of ERRCD. IERR requires one dummy parameter which is ignored. Any call to another external subroutine or execution of a PRINT statement resets the value of IERR(x).

Example

```
100 CALL TRNON(2000,124515)FAIL:GOTO 9000
 •
 •
 •
9000 IF IERR(X) = 1 GOTO 9100
9010 IF IERR(X) = 2 GOTO 9200
```

Specify task 2000 to be executed at 12:45 and 15 sec. If error, go to 9000.

If error is 1, go to 9100.

If error is 2, go to 9200.

6-6.   THE SERR FUNCTION

You may use the SERR function to set the ERRCD flag to a particular value in a subroutine. For example, the statement:

        110 I= SERR(N)   (I is a dummy variable)
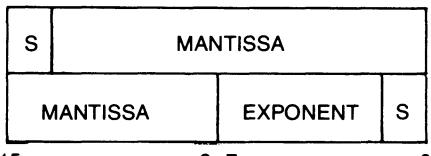
sets the ERRCD flag to the value of N. After execution of your subroutine you can examine the error code by using the IERR function. The value of I is unchanged.

The CALL statement initializes ERRCD to 0, however, you should initialize it at the beginning of your program and reset it to zero after you have detected an error in a routine and taken appropriate action to avoid leaving it set in case there are no more CALLs. You initialize the error code as follows:
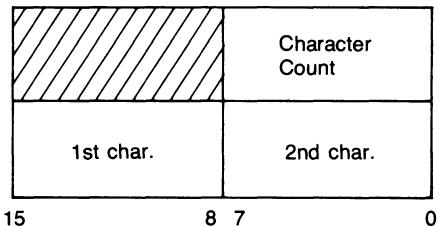
        10 I= SERR (0)

6-7.   PARAMETER CONVERSION

BASIC has two data types: number (real) and string. The format of real data is:

S	MANTISSA		
MANTISSA		EXPONENT	S

    15   8 7   0

and for string is:

////////	Character Count
1st char.	2nd char.

    15   8 7   0

The leftmost half of the 1st word may contain information used internally by BASIC.

If you want to pass parameters to or from external subroutines, you must be aware of the internal representation of these two data types. BASIC converts real data variables and arrays to integer and vice versa, and thus provides you with the ability to transfer data between BASIC and subroutines that use integer type data. For example, if the subroutine passes an integer array to BASIC, you must specify that parameter as an integer returned array when generating the Branch and Mnemonic Tables. (See Section XIV.)

Some intrinsic RTE and File Manager subroutines use string parameters but do not follow the BASIC string format convention. If you want to call one of these subroutines, you must first write an interface routine to convert the BASIC string to the necessary format. Figure 6-3 contains a FORTRAN routine which converts the program name to the FORTRAN string format before calling the File Manager EXEC subroutine.

```
0001    FTN,L,M
0002            SUBROUTINE EXEC9(I)
0003    C
0004    C
0005    C
0006    C
0007    C THIS SUBROUTINE CALLS THE RTE 'EXEC' TO SCHEDULE A PROGRAM WITH WAIT
0008    C
0009    C   FOR EXAMPLE:
0010    C
0011    C      10 DIM A$(6)
0012    C      20 PRINT "INPUT PROGRAM NAME";
0013    C      30 INPUT A$
0014    C      40 CALL EXEC9(A$)                    BASIC program
0015    C      50 END
0016    C
0017    C      >RUN
0018    C
0019    C      INPUT PROGRAM NAME?PROGA
0020    C
0021    C
0022    C
0023    C THE SUBROUTINE'S DESCRIPTION THAT MUST BE INPUT TO THE TABLE
0024    C GENERATOR TO CREATE THE PROPER ENTRY IN THE BRANCH AND MNEMONIC
0025    C TABLE IS AS FOLLOWS:
0026    C
0027    C    EXEC9(RA),   OV=NN , INTG, ENT=EXEC9, FIL=EXEC9R
0028    C
0029    C      WHERE:  RA         INDICATES REAL ARRAY PARAMETER( STRINGS
0030    C                         ARE ALWAYS SPECIFIED AS REAL )
0031    C              NN         INDICATES THE OVERLAY NUMBER
0032    C              EXEC9R     INDICATES THE FILE NAME OF THIS SUBROUTINE'S
0033    C                         RELOCATABLE.
0034    C
0035    C
0036        DIMENSION I(4)
0037    C
0038        CALL EXEC(9,I(2))                       FORTRAN Subroutine
0039        RETURN
0040        END
```

Figure 6-3. FORTRAN Subroutine to Convert String Parameter

6-3a. INVOKE

The INVOKE statement is used to schedule a second BASIC program from a calling program. The called program may also call another program and so on. When the currently executing, called program terminates, control is returned to its calling program.

Format

INVOKE *string variable* / *string literal* [,*statement number label*]

> The *string variable* or *literal* is the name of a Real-Time BASIC program or the type 6 RTE program that has been saved with the File Manager 'SP' command. This may be a fully qualified file name (see Section VII, Files). If the optional *statement number label* is present, execution begins at the first executable statement at or after the label; the exact label need not be present in the called program. If omitted, execution begins at the first executable statement in the called program.

INVOKE calls the program identified by the string expression, and it stores the current program on disc. When the program called by INVOKE finishes execution, it terminates and automatically returns to the calling program. The called program may call another program, including the original calling program, with the INVOKE statement.

Basic data files remain open when one program INVOKES another.

Only variables declared in a COM statement are saved during a INVOKE operation. All variables and arrays of the current program that were not declared in COM are lost when the new program begins execution. All programs must contain the same size COM area and the same number of file positions in the FILES statements so that common will be properly aligned. Files declared in INVOKE'd programs correspond to files declared in the main program. Logical units cannot be defined in a INVOKE'd program unless it is already declared in the main program's FILES statement. Refer to the FILES statement, Section 7-3, for examples.

If the programs are CSAVEd, the time required to execute the INVOKE statement is reduced.

Any TRAPS previously enabled will remain enabled.

The calling program is saved on the disc on program tracks in the system area of the disc. The levels deep of INVOKE is determined by the size of the calling program and the number of available system tracks. When the available system tracks are used up, the message "BASIC WAITING TRACKS" is printed. The copy of Multi-User BASIC being used by the operator is disc track suspended until system tracks become available, or the copy of Multi-User BASIC being used by the operator is terminated with an OF command (this is not recommended).

When BASIC invokes a program, the LU's of your terminal and list device are passed. If the devices are interactive, the echo bit (K-bit) is set.

The non-BASIC program that is INVOKE'd must exist as a type six file for BASIC to schedule it. This means that after the program to be INVOKE'd is loaded, it must be SP'd with the File Manager SP command. For example, if EXMPL is the non-BASIC program to be INVOKE'd first load the program then type,

    :SP, EXMPL

EXMPL is now stored as a type six file. BASIC first 'opens' the INVOKE'd program to determine its type. This type could be ASCII (type three or four) and BASIC assumes a BASIC program is to be INVOKE'd. If it is type six, then BASIC schedules the non-BASIC program.

When the user breaks the INVOKE'd program with the system break command (*BR,BASIC), BASIC returns control to the origianl program.

For situations that require permanent data storage external to a particular program, Real-Time BASIC provides a data file capability. This capability allows flexible direct manipulation of large volumes of data stored in files.

The simplest approach to files is to treat them as serial storage devices. Visualize a file as a list of data items, ordered serially. You can read the data in a file and write data to a file with your programs quite easily without worrying about the internal structure of the file. Several programs may access the same file along with yours. Each program uses its own data pointer to mark its position in the file, and functions independently of the other programs.

You may also envision files as structured data bases, internally organized as a collection of records — each record consisting of 128 16-bit words. Thus each record of a file may hold up to 64 numerical quantities. A string data item occupies $1 + (n+1)/2$ words where n is its length in characters.

To use a file you should be familiar with the CREATE and PURGE commands and with the statements listed below:

- FILES
- READ#
- PRINT#
- ASSIGN
- IF END

These commands and statements as applicable to files are defined in the remainder of this section.

7-1.   FILE CHARACTERISTICS

In order to create and use files, you must understand the following characteristics of Real-Time BASIC files. The conventions for file creation are the same as RTE File Manager conventions.

- A file name may contain from 1 to 6 characters. The first character may not be a number. Leading and trailing blanks are ignored. Embedded blanks are not allowed. Any printable ASCII character except the plus (+), hyphen or minus (−), comma (,), and colon (:) may be used.

- A file may be assigned a security code to control read/write access. The security code may be a number between −32767 and +32767. A positive code write protects the file. When accessing the file, you may supply a positive or negative version of the positive security code in order to write on the file. A negative security code both read and write protects the file. You must provide the negative code to read or write on a file protected by a negative code. If you do not want to protect the file, assign a zero security code.

  Two ASCII characters may be used in lieu of a positive security code. The first character may not be a number.

- Each file is assigned a type number. For a complete description of all file types, see the Batch-Spool Monitor Reference Manual. The types you will be using with BASIC are: type 0, type 1, type 4, and type 10.

  A type 0 file defines an I/O device. You must create type 0 files with the File Manager CR command. After you create a type 0 file, you can use the file name to reference the device it defines.

  A type 1 file contains data. You must create this type of file with the Real-Time BASIC CREATE command if you are using it with BASIC programs.

  A type 4 file is created when you SAVE a program. You may also create a type 4 file with the File Manager or Interactive Editor and store source programs or commands in it.

  A type 10 file is created when you CSAVE a program.

- When you create or access a file, you can specify the cartridge reference. The cartridge reference can be a positive integer corresponding to the label of a currently mounted cartridge or a negative logical unit number referencing a disc. The file will be created on or accessed from the specified cartridge. If you specify a zero, the cartridges are accessed in the order in which they appear in the File Manager Cartridge Directory.

7-2.   CREATE AND PURGE

The CREATE command is used to create a file for use by a program and the PURGE command is used to remove a file. These commands are described in Section IX, paragraphs 9-6 and 9-7.

7-3.   FILES STATEMENT

Every file that is to be accessed by a program must be identified in the program's FILE statement.

Format

FILES *filename*$_1$ [, *filename*$_2$, . . .   *filename*$_n$ [*:security*[*:cartridge*]]]

Parameters

filename$_n$ name of file to be referred to by number corresponding to position in FILES statement, or an asterisk indicating file will be assigned later or zero indicating position refers to a logical unit.

security   optional security code which may be supplied with each filename.

cartridge   optional cartridge reference which may be supplied with each filename (either positive cartridge reference number or negative LU number).

The FILES statement declares which files will be used in a program. Up to four FILES statements can appear in a program, but only 16 files total can be declared. The files are assigned numbers (from 1 to 16) in the order in which they are declared in the program. In the examples below MATH is file #1 and #9, FILE27 is #7, and DATA is #10. File position #3 will be assigned to a filename with the ASSIGN statement. File position #4 is specified as zero to indicate LU4.

Example

```
10 FILES MATH, SCORE, *, 0, NAMES
20 FILES GRP, FILE27, SAMPLE:JS:10
30 FILES MATH, DATA
```

These numbers are used in the program to reference the files. For instance, the statement:

```
100 PRINT #2;A
```

would print the value of A onto the file named SCORE. This feature allows most programming to be done independently of files to be used. The FILES statement may be added any time before running the program.

All programs which are INVOKE'd or CHAINed must contain the same size COM area and the same number of file positions in the FILES statements so that common will be properly aligned. An asterisk in the FILES statement can only be assigned within the program containing that FILES statement. Therefore, an asterisk in the main program cannot be assigned to a file by a CHAINed or INVOKE'd program. Also an asterisk in a CHAINed/INVOKE'd program is the only way to open a new file.

Example

Main program creates the files and declares 3 files:

```
CREATE A,1
CREATE B,1
CREATE C,1
CREATE G,1
10 FILES A,B,C
20 PRINT "MAIN PROGRAM"
30 INVOKE "SUB1"
      .
      .
      .
      .
90 END
```

SUB1 program is invoked and defines file G:

```
 5 REM** FILES A&B ARE THE SAME FILES AS IN THE MAIN
 6 REM** FILE 3 WILL BE ASSIGNED TO FILE G
10 FILES A,B,*
20 PRINT "SUB1"
30 ASSIGN "G",3,J
      .
      .
      .
50 PRINT #3; "GGGG", END    file G has "GGGG" written into the first record.
      .
      .
      .
90 END
```

Files

Main program has 3 files:

```
CREATE A,1
CREATE B,1
CREATE C,1
10 FILES A,*,C
20 ASSIGN "B",2,J
30 INVOKE "SUB2"
        .
        .
        .
90 END
```

SUB2 program is invoked. Although it uses file names A1, A2, A3, they correspond to files A, B, C in the main program:

```
10 FILES    A1,A2,A3
20 PRINT "SUB2"
        .
        .
        .
90 END
```

A CHAINed/INVOKE'd program uses the zero as a file position holder only. Whatever files or LU's the main program has defined correspond in the CHAINed/INVOKE'd program's FILES statement.

Example

Main program has 3 files and LU 4 declared:

```
10 FILES A,B,C,0
20 PRINT "MAIN"
30 INVOKE "SUB3"
        .
        .
        .
90 END
```

SUB3 program is invoked:

```
10 FILES 0,0,0,0          The zeroes define files A, B, C and LU 4 from the
20 PRINT "SUB3"           main program to be in SUB3.
        .
        .
        .
40 REM** WRITE INTO FILE 1 WHICH IS FILE A
41 REM** FROM THE MAIN PROGRAM WHEN SUB3 IS INVOKED,
42 REM** OTHERWISE FILE 1 IS LU1
45 PRINT #1, "THIS IS SUB3"
        .
        .
        .
90 END
```

7-4. ASSIGN STATEMENT

The ASSIGN statement is used to change the file referred to by a specified file number during the execution of a program.

Format

ASSIGN *filename* [*:security*[*:cartridge*]] *, file number, return variable*

Parameters

filename	the name of a file, (i.e., a literal string of up to six characters) enclosed in quotes, or a string variable containing a literal string.
security	optional security code. Must be supplied if file was created with a security code.
cartridge	optional positive cartridge reference number or negative LU number.
file number	a number, variable, or expression whose value is between 1 and 16, indicating a file's position. The file number should not exceed the number of files declared in the FILES statement.
return variable	the value returned to this variable when the statement is executed. Values are summarized in Table 7-1.

When the ASSIGN statement is executed, the named file replaces the file previously referenced by the file number in the statement. Subsequent file references using this number will apply to the new file. Data written to the old file will be intact.

Example

```
10 ASSIGN "NEW:AJ:-44",1,J
20 ASSIGN A$, 3, B1
30 ASSIGN "NEWFL", S2, J
40 ASSIGN "$F2", 6, C
```

After each file is assigned you should test the value of B1, J, and C for error conditions as summarized in Table 7-1.

Each file that is assigned (opened) requires 144 words of user memory space.

Table 7-1. ASSIGN Return Variable Values

RETURN VARIABLE	MEANING
−1	Disc is inoperable.
−5	File number is out of range. It does not correspond to any of the positions in the FILES statement.
−6	File not found.
−7	Invalid security code.
−8	File currently open to a program that demands exclusive use of the file or to 7 programs (the maximum allowed).
−13	Cartridge is locked.
−32	Specified disc not found.
−33	No more room on disc.

7-5.   IF END # . . . . THEN STATEMENT

The IF END statement sets a flag for a specified file so that if and when an end-of-file condition occurs in reading and writing the file, control is transferred to a specified statement. If the flag is not set, an end-of-file condition causes the program to terminate.

Format

IF END *# file number* THEN *statement number label*

Parameters

file number   a number, variable, or expression whose value is between 1 and 16, indicating a file's position.

statement number label   the number of the statement to which control will transfer.

When an end-of-file condition occurs during execution of a READ # statement, the IF END statement transfers control to the statement specified. The IF END statement remains in effect until another IF END for the same file changes it, or until an ASSIGN statement containing the same file number is executed.

An end-of-file condition occurs when a file read operation encounters an end-of-file mark or the physical end of file, or when a file write operation encounters the physical end of the file.

Example

    10 IF END #3 THEN 125

7-6.   RESTORING THE DATA POINTER

A READ # statement may be used to reposition the data pointer to the start of a file. The statement can be used for any file accessed by the program.

Format

READ # *file number*, 1

Parameter

file number   a number, variable, or expression whose value is between 1 and 16, indicating the file's position in the FILES statements.

When the statement is executed, the data pointer is set to point to the beginning of the first record. A serial read or print will begin at that position. Do not use PRINT #1, 1 to reset the pointer because this will delete the content of record 1.

Example

```
300 READ #1,1
```
          The pointer for file position #1 is reset.

7-7.   SERIAL FILE READ STATEMENT

The READ # statement can be used to read items from a file into numeric or string variables. The first item read is the item following the current position of the data pointer.

Format

READ # $\begin{array}{c} \textit{file number} \\ \textit{lu} \end{array}$; *variable list*

Parameters

file number   a number, variable, or expression whose value is between 1 and 16, indicating a file's position. An expression is rounded to the nearest integer.

lu   if a file position corresponding to the *file number* does not exist in the program or the file position contains a zero (i.e. FILES 0,FILA), the number is interpreted as a logical unit number (*lu*).

variable list   a series of variables separated by commas. The rules governing this list are the same as those described for the READ statement described in Section III.

The READ # statement fills the variables in the list from the designated file beginning with the data item referenced by the data pointer and moving from record to record as necessary. A subsequent file read will start with the first data item not used by the prior read; record boundaries of portions of records not containing data are ignored.

Strings and numbers may be intermixed in the data list, but the types in the item list and data list must correspond in kind and order. The destination for a string value must be a string variable; the destination for a numeric value must be a numeric variable.

If the file named in the FILES statement is a type 0 file, the data is read from the device corresponding to the file. See Section IX for more information about type 0 files.

It is possible to check the type of the next data item with the TYP function described later in this section.

When an attempt is made to read beyond a logical or physical end-of-file, an end-of-file condition occurs. Unless an IF END statement transfers control to another statement in the program, the program terminates.

Example

```
400 READ #2;A,B(3),R(1,3)
```

7-8.   READING A RECORD

The READ # statement may also be used to read items from a designated record into a numeric or string variable. This is called a direct read.

Format

READ # *file number, record number; variable list*
Parameters
file number   same as for READ #, paragraph 7-7.
record number   a number, variable, or expression indicating on which record the list is to be printed. Ignored if *file number* refers to LU number or type 0 file.
variable list   see paragraph 7-7.

Execution of a read record statement fills the variables in the variable list from the designated record only, starting at the beginning of the record. Encountering an end-of-record mark or the physical end-of-record will generate an end-of-file condition and terminate the program unless an IF END statement is provided.

The file pointer is positioned at the start of the record initially but may be left in the middle of the record following the read operation. In this case, the remainder of the record can be read by a subsequent serial read.

When manipulated on a record-by-record basis, a file appears very much as a collection of subfiles which consists of records. The ability to reference any record of the file directly allows you to partition your data and alter any group without disturbing or having to read or rewrite the rest of the file.

Example

```
75 READ #1,3; A, B1, C(I,J+2)
80 READ # I,J                    Positions file pointer to record J without reading
                                 any data.
```

7-9.   SERIAL FILE PRINT STATEMENT

The serial file PRINT # statement writes data items on a file, starting at the current position of the pointer. The items may be numeric or string expressions.

Format

$$\text{PRINT} \ \# \ \begin{matrix} \textit{file number} \\ \textit{lu} \end{matrix} \ ; \ \textit{print list} \ [, \text{END}]$$

Parameters

file number	a number, variable, or expression whose value is between 1 and 16, indicating a file's position in the FILES statement.
lu	if a file position corresponding to the *file number* does not exist in the program or the file position contains a zero (i.e. FILES 0,FILA), the number is interpreted as a logical unit number *(lu)*.
print list	series of numeric expressions, numeric or string variables, or string literals.
END	optional constant which prints EOF on the file.

The PRINT # statement performs essentially the same operation as the ordinary PRINT statement, except that data is written to a file or logical unit rather than to a terminal. No line formatting of files takes place, the comma and semicolon act only as delimiters and may not be used as actual data unless the *lu* number refers to a teleprinter or lineprinter. (See the PRINT statement, paragraph 3-7.) When printing to the line printer using PRINT #6, BASIC inserts a leading space in column 1. The PRINT USING (Section 3-7a) or PRINT to a type 0 file should be used to control the line printer.

Writing of the first value begins wherever the file pointer is positioned and new records are used as necessary. Writing onto a file overlays whatever may have been in that area, including end-of-file marks. To ensure that a file actually ends with the last item written, the special constant END may be placed at the end of the print list. END is significant only when it is the last item written on a file.

The END constant should not be used when the PRINT# statement refers to an LU. The interpreter cannot distinguish the difference between a file number or an LU when the line is evaluated by the parsing routine. Therefore, a run-time error will occur.

If printing is attempted beyond the physical end of the file, an end-of-file condition occurs. The IF END statement can be used to specify action in this case, or the program will terminate.

Since character strings vary in length and each string must be wholly contained within a record, some space in each record may be left unused. You can calculate the number of words occupied by any string with the formula described in paragraph 1-6.

If the file named in the FILES statement is a type 0 file, the data is printed to the device corresponding to the file. See Section IX for more information about type 0 files.

You should not do a serial read after PRINT without resetting the pointer. For information about modifying records, see paragraph 7-12.

```
20 FILES FILA
30 PRINT #1; A,"STRING",A$
```
The value of A, the string "STRING", and A$ are printed on FILA.

```
20 FILES LINEP
   .
   .
   .
60 PRINT #1; "SUMMARY"
```
LINEP is a type 0 file corresponding to the line printer.

The string literal SUMMARY is printed on the line printer.

```
100 FILES FILEA,FILB,FILC,0,FILD
    .
    .
    .
150 PRINT #4; A,B,C
```
The variables A, B, C are punched on the paper tape punch, LU 4.

```
200 PRINT#(I+J);  2,42,A,B,C,D(3,5),END
```
The items are printed on the file in position I+J followed by an end-of-file mark.

To advance a page (top-of-form) on a line printer, set up a type 0 file to the line printer as follows:

```
10 FILES 0,LP
20 PRINT #2;"1"
30 END
```

where: LP is a type 0 file for the line printer.

7-10. PRINTING A RECORD

The PRINT # statement may also be used to write items to a designated record.

Format

PRINT # *file number, record number; print list*

Parameters

file number	see paragraph 7-9.
record number	a number, variable, or expression indicating on which record the list is to be printed. Ignored if *file number* refers to LU number or type 0 file.
print list	see paragraph 7-9.

Execution of a print record statement performs the same task as serial PRINT with the exception that the operation is limited to one record. The entire list of data must fit in one record. An attempt to write more than one record or more than the record can hold results in an end-of-file condition. The write begins at the beginning of the record, which is scratched of previous data. Again, the use of the IF END statement will avoid program termination at the end of the write.

Example

```
200 PRINT #3,4; A,B,C,D
210 PRINT #J,9
```
                                    Positions file pointer to record 9 on file named in
                                    position J.

7-11.   TYP FUNCTION

The TYP function returns the type of the next data item for a particular file. This function is used in conjunction with file READ statements since the variables into which the data is read must be string if the item is string; numeric if the item is numeric.

Format

TYP *(file number)*

Parameter

file number   a number, variable, or expression whose value is between 1 and 16, indicating a file's position.

The value returned by TYP depends on the type of the next data item in the file.

TYP(x)	MEANING
1	number
2	string
3	end-of-file
4	end-of-record

If the *file number* is greater than zero, the file is treated as a serial file. This means that the value 4 is never returned as end-of-record marks are skipped in a serial file.

If the *file number* is less than zero, the file is treated as a direct file and any of the values of TYP may be returned. Since the file number is based on the absolute value of the expression, an expression equal to -1 means that file number 1 is examined and treated as a direct file.

If the *file number* is zero, TYP returns a result based on the current position of the pointer to the DATA statements (see READ/DATA/RESTORE description, paragraph 3-10). The value 4 is never returned, and 3 means end-of-data.

7-12.   MODIFYING RECORDS

If you print an item to a record, all items following that one are no longer in the record. It is necessary to read and save, either in memory or on another record, all items following the one you want to modify and to copy these items back on the record after printing the new item. You must also be sure that the item you are modifying is not replaced with data which requires more space in the record than is available. This can occur if you replace a string with a longer string.

If you are modifying an item in a serial file, you must use this same procedure but copy all items in the file which follow the item you are modifying.

Multi-User Real-Time BASIC is itself a program, and as such must be made available for use on your system. Procedures for loading BASIC are provided in Appendix D. Once BASIC is loaded and ready for use, you need simply schedule BASIC for operation as described below.

8-1. SCHEDULING BASIC

The command to start the BASIC Interpreter is:

$$:RUN,BASIC \quad \genfrac{}{}{0pt}{}{[,p1[,p2[,p3[,p4[,p5]]]]]}{[,console[,list[,input[,output[,error\ list]]]]]}$$

console	—is the logical unit number of the keyboard device you are using. Default is LU 1, the system console, or the *lu* of your MTM terminal.
list	—is the logical unit number of the list device you want to use. Default is LU 1, the system console, or the *lu* of your MTM terminal.
input	—is the logical unit number of the input device you want to use. Default is LU 5, the paper tape reader.
output	—is the logical unit number of the output device you want to use. Default is LU 4, the paper tape punch.
error list	—is the logical unit number of the error message logging device you want to use. Default is LU 1, the system console, or the *lu* of your MTM terminal.
p1, p2, p3, p4, p5	—optional parameter string that you may pass to the BASIC program. This string may be obtained by executing the program shown in Figure 8-1.

If you omit the optional parameters, the default devices specified above will automatically be used by BASIC. You must type a comma in place of the omitted parameter if you specify subsequent parameters. If you wish to alter the devices used, enter the logical unit numbers of the devices you prefer.

BASIC may be scheduled from either the RTE Operating System, which prompts with an asterisk (*), or from the File Manager, which prompts with a colon(:).

The following is an example of the commands used to schedule BASIC starting with the Operating System and proceeding through the File Manager to BASIC.

```
*RUN,FMGR
:RUN,BASIC
BASIC READY
>
```

There may be more than one copy of BASIC on your system. See Appendix D for more information about copies of BASIC.

```
ASMB,R,L,C
      NAM GETPR,3
*
*THIS SUBROUTINE IS USED TO RETRIEVE THE PARAMETERS PASSED TO BASIC
*VIA THE :RU,BASIC,P1,P2,P3,P4,P5 COMMAND.  ALL PARAMETERS FOLLOWING
*THE SECOND COMMA ARE PASSED BACK TO THE BASIC PROGRAM, UNLESS 'NOW'
*IS USED THEN ALL PARAMETERS PAST THE THIRD COMMA.  ANY CHARACTERS
*AFTER PARAMETER P5 USED IN THE RUN COMMAND (UP TO A TOTAL OF 20
*CHARACTERS) ARE ALSO RETRIEVED BY THE GETPR PROGRAM.
*
*THE TABLE GENERATION SPECIFICATION IS AS FOLLOWS:
*
*      GETPR(RVA), OV=NN, ENT=GETPR, FIL=FILENAME
*
*THE CALLING SEQUENCE FROM BASIC IS SHOWN IN THE FOLLOWING EXAMPLE:
*
*      :RU,BASIC,1,6,HERE I AM
*      BASIC READY                  \
*                                    \
*      >10 DIM A$(20)                 \
*      >20 CALL GETPR(A$)         >     GETPR CALLING SEQUENCE
*      >30 PRINT A$               /
*      >40 END                    /
*
*      >RUN                        \
*                                   \
*      1,6,HERE I AM              >     RESULT OF PROGRAM EXECUTION
*      BASIC READY                /
*      >                          /
*
*NOTE: THIS SUBROUTINE MUST BE CALLED BEFORE ANY OTHER SUBROUTINE
*      ELSE THE 'RUN' PARAMETERS WILL BE LOST FOREVER AND EVER.
*
*
      ENT GETPR
      EXT GETST,.ENTR
STR   NOP
GETPR NOP
      JSB .ENTR
      DEF STR
*
      LDA STR
      INA                   SET UP STRING
      STA TEMP                 ADDRESS
      JSB GETST             GET
      DEF *+4                  PARAMETER
      DEF TEMP,I
      DEF M100
      DEF NLOG
*
      LDA NLOG              SET UP
      STA STR,I                STRING COUNT
      JMP GETPR,I           RETURN
*
NLOG  BSS 1
TEMP  BSS 1
M100  DEC -100
*
      END
```

Figure 8-1. Sample Program, Parameter String Retrieval

8-2. USING BASIC

BASIC prompts for commands and program statements with the greater than symbol(>).The first command you should enter, if you plan to use subroutines in your program, is the TABLES command. This command declares the names of two tables used by BASIC, the Branch and Mnemonic Tables.

To specify the Branch and Mnemonic tables enter:

TABLES *branch filename* [:*SC*[:*CR*]], *mnemonic filename* [:*SC*[:*CR*]]

branch filename	is the name of the file, created by the RTE BASIC Table Generator,which contains subroutine branch information.
mnemonic filename	is the name of the file, created by the RTE BASIC Table Generator, which contains subroutine mnemonics.
SC	is an optional security code parameter.
CR	is an optional positive cartridge number or negative logical unit number.

Procedures for table generation are provided in Section XIV.

BASIC programs can be stored as File Manager files with the BASIC SAVE command and reloaded with the LOAD and RUN commands. The LOAD command does not execute the program, but merely makes it available for execution or editing. The RUN command runs the current program if you do not provide a file name with it, or it loads and runs the program in the file name provided. In BASIC the file name and program name are the same. Sections IX and X provide a complete description of the many operating commands available to you with Real-Time BASIC.

8-3. START UP OPTIONS

You may input commands to BASIC from the console keyboard or from an ASCII disc file previously prepared by using the RTE Interactive Editor, EDITR, or the File Manager, FMGR. If you input commands from a disc file, you may choose from two methods of initiating BASIC:

- you may start BASIC from the console as shown below, or

- you may schedule BASIC from another program. The program may be written in FORTRAN, ALGOL, Assembly language, or BASIC.

The ASCII command file can contain any legal BASIC commands. It may not, however, contain any BASIC statements. Following is an example of an ASCII command file:

>TABLES BRT,MNT	*Sets up Branch and Mnemonic Tables.*
>CREATE DATA1,20	*Creates a data file.*
>RUN PROGA	*Runs program PROGA.*
>LIST LP	*Lists program to type 0 file,LP, corresponding to line printer.*
>BYE	*Terminates Real-Time BASIC.*

To start BASIC with a command file from the console, input the following command to the File Manager:

:RU,BASIC,*fl,na,me*[,*console,list*]

fl,na,me is the name of the file containing the commands. Commas must be used to separate every two letters. The name is actually *flname*.

console is the logical unit number of the terminal you are using. Error messages will be output at the specified terminal. The default is LU 1, the system console.

list is the logical unit number of the list device you want to use. The default is LU 1, the system console.

To initiate BASIC from another program, use the following calling sequences.

Assembly language:

```
JSB EXEC                    Transfer control to RTE.
DEF *+7
DEF .9                      Request code (.10 if schedule without wait).
DEF BASIC                   Name of BASIC.
DEF P1  ⎫
DEF P2  ⎪
DEF P3  ⎬                   Parameters.
DEF P4  ⎪
DEF P5  ⎭

                            Continue execution of program.
.9  DEC 9                   Schedule with wait.
BASIC ASC 3,BASIC           Name of BASIC Interpreter.
P1 ASC 1,FN ⎫
P2 ASC 1,AM ⎬               Command file name.
P3 ASC 1,E  ⎭
P4 DEC 1                    Console logical unit number.
P5 DEC 6                    List logical unit number.
```

FORTRAN

```
                            Store name of BASIC in integer
DIMENSION NAME(3)           array name.
DATA NAME/2HBA,2HSI,2HC /
CALL EXEC(9,NAME,2HFN,2HAM,2HE ,1,6)   Transfer control to RTE.
                            FNAME is the command file
                            name. LU 1 is the console, LU 6
                            the list device.
```

For additional information on scheduling BASIC from programs see the RTE-II or RTE-III Programming and Operating Manual and the Batch-Spool Monitor Reference Manual.

OPERATOR COMMANDS

This section describes the Multi-User Real-Time BASIC commands. Unlike the statements discussed in earlier sections, commands are not part of a program, nor are they preceded by line numbers. When entered, a command is executed immediately.

Table 9-1 lists and defines the various operator commands available to you with Multi-User Real-Time BASIC. Detailed explanations of most of the commands are provided in the remainder of the section. Additional commands used in specific situations such as program debugging are introduced and explained in subsequent sections. The TABLES command is described in Section VIII. A complete list of the commands and their uses is provided in Appendix A.

File characteristics (filename, security, cartridge) are described in Section 7-1.

9-1.   LOAD

The LOAD command enables you to load all or a portion of a source program or a semi-compiled program from a specified file.

Format

LOAD [*limits*] [*filename* [*:security* [*:cartridge*]]]

Parameters

limits	beginning and ending line numbers of the portion of the program you want loaded separated by a comma. Limits are omitted if the entire program is to be loaded.
filename	name of file containing the program or type 0 file corresponding to a device from which the program is to be loaded. The default is LU 5 or the LU number specified as the input parameter in the RUN,BASIC command.
security	optional security code. Must use if program saved with security code.
cartridge	optional cartridge reference (label or LU number).

The LOAD command reads in all program statements between and including the line numbers specified as limits. If limits are not specified, the entire program is loaded.

Once loaded, a program is ready for execution or editing.

Examples

```
>LOAD                          Loads from LU 5 by default.
>LOAD 150,250 CARD             Loads from a file named CARD.
```

9-1

When a syntax error is encountered during the loading of a program, BASIC prints an error and returns to the command mode.

The correct procedure for recovering from syntax errors during LOAD is illustrated by the following example:

Given the file TEST::1132

```
10 LET A=1
20 LTE B=A
30 LET C=B
99 END
```

The load procedure is:

```
>LOAD TEST::1132
MISSING ASSIGNMENT OPERATOR IN LINE 20
      20 LTE B=A
>20 LET B=A
>MERGE TEST::1132,21

BASIC READY
>LIST
10 LET A=1
20 LET B=A
30 LET C=B
99 END

>
```

Table 9-1. Operator Commands

COMMAND	FUNCTION
LOAD	Loads a source program or a semi-compiled program from a file.
SAVE	Stores a program on disc as a source program.
CSAVE	Stores a program on disc in semi-compiled format.
MERGE	Merges a source program with a program in memory.
REPLACE	Replaces a source program on disc.
DELETE	Deletes a program from memory.
CREATE	Creates a data file on a device.
PURGE	Deletes a program or file from disc.
RENAME	Removes the name of a file on the disc and replaces it with a new name.
RESEQ	Renumbers the statements in a program.
RUN	Loads and executes a program.
LOCK	Locks a peripheral device to your program.
UNLOCK	Unlocks a locked device.
BYE	Terminates the use of BASIC.
LIST	Lists a program onto a file.
*BR,BASIC	Breaks execution of a program.
CALLS	Lists all of the mnemonics in the current Mnemonics Table.
TABLES	Specifies Branch and Mnemonic Table names. (Described in Section VIII also.)
REWIND	Rewinds magnetic tape.*
SKIPF	Skips to end of file on magnetic tape.*
BACKF	Backspaces to end of file on magnetic tape.*
WEOF	Writes end of file on magnetic tape.*

*These commands are described in Section XIII.

9-2. SAVE/CSAVE

The SAVE command stores the BASIC program currently in memory on the disc. CSAVE saves a program in semi-compiled form for faster loading.

Format

SAVE   [*limits*] [*filename* [*:security* [*:cartridge*]]]

CSAVE [*filename* [*:security* [*:cartridge*]]]

Parameters

limits   beginning and ending line numbers of the program to be saved. If specified, limits must be separated by a comma. If no limits are specified, the entire program currently in memory is saved.

filename   name of the file in which the program is to be saved. If the file already exists, an error message is printed. If the file does not exist, it is created and the program is saved. The file is closed when the operation is completed. If no *filename* is specified, the program will be output to LU 4 (for SAVE only) or the LU number specified as the output parameter in the RUN,BASIC command.

security   optional security code.

cartridge   optional cartridge reference (label or LU number).

The SAVE and CSAVE commands store program statements on the disc. The SAVE command may store only the statements between and including the line numbers specified. If limits are not specified, the entire program is stored.

A SAVEd program can be edited with the Interactive Editor and can also be loaded by BASIC. CSAVE is used for faster loading and particularly where several large programs are CHAINed together. A CSAVEd program should always be backed up by a SAVEd version because it is in binary format. If any modification of the Interpreter or a new system generation takes place, the CSAVEd program becomes unloadable by the new version of BASIC or the RTE system. CSAVE is used generally for production programs where no modification is being made and speed of loading and chaining is important. A program cannot be CSAVEd to an LU or type 0 file.

Rules for naming files are given in paragraph 7-1.

Once stored, the program can be loaded and executed as needed.

When a program containing external subroutines is CSAVEd, the names of the Branch and Mnemonic Tables are stored with the CSAVEd program. Therefore, if there is more than one copy of that program, be aware that each version is related to a different set of Branch and Mnemonic Tables and overlays.

Examples

>SAVE	*The current source program is output on LU 4.*
>SAVE 100,260	*Lines 100 through 260 of the current program are output to LU 4.*
>SAVE PROGA	*The current program is saved in file PROGA.*
>CSAVE PROGB	*Semi-compiles and saves program in file PROGB.*
>CSAVE	*Output semi-compiled form of current program to LU 4.*

9-3. MERGE

The MERGE command merges a source program or portion of a source program with a program in memory.

Format

MERGE [*limits*] [*filename* [*:security* [*:cartridge*]]]

Parameters

limits beginning and ending line numbers of the program to be merged. Limits must be separated by a comma. If no limits are specified the entire program is merged.

filename name of the program to be merged. Program name may be omitted, if program is to be loaded from the standard input device (LU 5).

security optional security code. Must be supplied if program saved with security code.

cartridge cartridge reference (label or LU number).

MERGE merges a BASIC source program with a program in memory. If any line numbers of the program being entered match those of the program in memory, the like numbered statements of the program in memory will not be replaced. MERGE is useful if you want to load a program with syntax errors. You can retype the statements which are in error and edit them, if necessary, and then merge in the rest of the program. The corrected statements will not be replaced by the ones in error.

Examples

>MERGE	*Merge source statements from LU 5.*
>MERGE 120,190	*Merge source statements numbered 120 through 190 from LU 5.*
>MERGE PROGA	*Merge source statements from file named PROGA.*

9-4.   REPLACE

The REPLACE command enables you to replace a source or semi-compiled program on disc with the program (or part of the program) currently in memory. The entire file on disc is replaced in either case.

Format

REPLACE [*limits*] [*filename* [*:security* [*:cartridge*]]]

Parameters

limits   beginning and ending line numbers of the program in memory you want to store in the file. Limits must be separated by a comma. If no limits are specified, the entire program is stored.

filename   name of the program on disc which will be replaced. If no name is specified, the command executes like the SAVE command with output to the LU number specified as the output parameter in the RUN,BASIC command.

security   optional security code.

cartridge   optional cartridge reference (label or LU).

This command replaces an entire program stored on disc with a program or part of a program currently in memory. The name of this disc program is not changed by the operation.

Once the operation is completed, the program may be loaded from disc to memory and executed.

Examples

>REPLACE   *Program is output to a logical unit.*

>REPLACE 250,350 PROGS   *Lines 250 through 350 of the program in memory*
                                  replace all of the PROGS file.

9-5.   DELETE

The DELETE command enables you to remove a program or part of a program from memory.

Format

DELETE [*limits*]

Parameter

limits   beginning and ending line numbers of the portion of the program to be deleted. Limits must be separated by a comma. If limits are not specified, the entire program is removed.

DELETE effectively erases the program currently in memory. Versions of the same program stored on disc are not affected by the DELETE operation. The limits parameter allows you to delete specific portions of a program in memory.

The DELETE command is the only one which may be abbreviated; you may use DEL when entering this command.

A single statement may be deleted by simply typing the statement number and pressing RETURN.

Examples

>DELETE   *Deletes the current program from memory.*

>DELETE 50,425   *Deletes lines 50 through 425 of the current program.*

>DEL   *Deletes the current program from memory.*

>45   *Deletes line 45.*

9-6. CREATE

The CREATE command enables you to create a data file (type 1) on a specified disc cartridge.

Format

CREATE *filename* [*:security* [*:cartridge*]] [*,file length*]

Parameters

filename   name of file to be created. Must conform to the same naming conventions as imposed on programs, e.g. no more than six characters in length, and so forth.

security   optional security code to assign to file being created to prevent unauthorized access to it.

cartridge   optional cartridge reference (label or LU number) specifying on which cartridge the file is to be created.

file length   number of 128 word records in file. Restricted only by the amount of disc available on the specified cartridge or all cartridges if none is specified. The default file length is one 128 word record.

CREATE initializes a file with an end-of-file mark, that is, as an empty file containing only an end-of-file indicator. If there is not room on the disc for the file, an error message is printed. You can terminate BASIC with BYE and then use the File Manager PK (pack) command to pack the disc and/or purge unused files to recover contiguous space on the disc. See the Batch-Spool Monitor Reference Manual for information about the PK command.

Examples

```
>CREATE FILEA,10          Creates a file named FILEA containing 10
                          records.
>CREATE FL101:386,20      Creates a file named FL101 of length 20 with
                          security code 386.
```

9-7. PURGE

The PURGE command enables you to remove a program or data file stored on disc.

Format

PURGE *filename* [:*security* [:*cartridge*]]

Parameters

filename   name of the file to be purged.

security   security code of file to be purged. Must be supplied if the file was created with a security code.

cartridge   optional cartridge reference (label or LU number).

This command removes the version of a program (or data file) stored on disc. Only the entire program can be purged. A version of the program currently in memory is not affected by the PURGE command.

Once purged, a program is no longer available for loading and/or execution. Purging a file does not make more space available on the disc. To do so you must use the File Manager PK (pack) command. See the Batch-Spool Monitor Reference Manual for information about this command.

Examples

```
>PURGE FL101:386          Purges file named FL101 with security code 386.
>PURGE PROGA              Purges the file containing PROGA.
>PURGE FILEA              Purges the data file named FILEA.
```

9-8.   RENAME

The RENAME command enables you to establish a new name for a program or data file.

Format

RENAME *old filename* [*:security* [*:cartridge*]] *,new filename*

Parameters

old filename   name of the file to be changed.

security   optional security code of file to be changed. Must be supplied if the file was created with a security code.

cartridge   optional cartridge reference (label or LU number).

new filename   name to be assigned to the file.

This command removes the name of a file on disc and replaces it with a new name. Once the RENAME operation is completed, the program may be loaded and executed by its newly assigned name only. The old name can no longer be used for the program or data file, but may be assigned to another file.

Rules for naming files are given in paragraph 7-1.

Example

>RENAME PROGA,PROGZ   *The file named PROGA is now named PROGZ.*

9-9.   RESEQ

The RESEQ command renumbers a range of statement numbers in specified increments.

Format

RESEQ [*new initial number* [*,increment* [*,first old number* [*,last old number*]]]]

Parameters

new initial number   number to be assigned to the first statement. Default = 10.

increment   interval between new statement numbers. Default = 10.

first old number   number of first statement to be renumbered. If omitted, renumbering begins with the first statement.

last old number   number of last statement to be renumbered. If omitted, renumbering terminates with the last program statement.

Any parameter may be omitted, but all parameters following it must also be omitted.

RESEQ can not be used to change the order of statements in a program. When negative statement numbers are used in TRAP statements, RESEQ will resequence but it cannot put back the negative sign. A warning message is issued. Remember to edit those TRAP statements which originally had negative statement numbers.

Examples

>RESEQ ←————————————————— *Renumber entire program in increments of 10.*

>RESEQ 20,5,15,123 ←———————— *Statements 15 through 123 are renumbered in increments of 5 beginning with 20.*

>RESEQ 110,5 ←——————————————— *The entire program is renumbered in increments of 5 beginning with the number 110.*

9-10. RUN

The RUN command enables you to load and execute a program or portion of a program in one operation.

Format

RUN [*limits*] [*filename* [:*security* [:*cartridge*]]]

Parameters

limits   beginning and ending line numbers of the portion of the program to be executed. Limits must be separated by a comma. If limits are not specified, the entire program is executed.

filename   name of the program (filename) to be loaded and executed. Name is specified only if the program is not currently in memory.

security   optional security code. Must be used if program was saved with security code.

cartridge   optional cartridge reference (label or LU number).

This command loads and executes a program or portion of a program. It can be used for a program already in memory or a program stored on disc, obviating the LOAD command.

Examples

>RUN   *Executes the program currently in memory.*

>RUN PROGA   *Loads PROGA if not in memory. Executes the program in either case.*

>RUN 50,75,PROGB   *Loads and executes statements 50 through 75 in PROGB.*

9-11. LOCK/UNLOCK

The LOCK command allows you to use a peripheral device exclusive of all other system users. The UNLOCK command returns a device to common availability.

Format

```
    LOCK      lu
    UNLOCK [lu]


    Parameter

    lu                  logical unit number of the peripheral device to be locked or unlocked. lu is
                        optional with the UNLOCK command. If not specified all locked devices
                        are unlocked.
```

The LOCK command locks a specified peripheral device to your version of BASIC, so that no other program can access that device while the lock is in effect. UNLOCK relinquishes your control of the device.

BASIC automatically unlocks all peripherals at termination of the BASIC Interpreter.

If you attempt to use a device locked by another user, an appropriate error message is generated.

Examples

>LOCK 6   *Locks the line printer, LU 6.*

>UNLOCK   *Unlocks all locked devices.*

9-12. BYE

The BYE command is used to terminate execution of the BASIC Interpreter.

Format

```
    BYE
```

BASIC is terminated immediately upon entry of the BYE command. Control is returned to the RTE Operating System or the program that scheduled BASIC (i.e. FMGR or some other calling program). You should always use BYE (not *OFF,BASIC,1) to terminate your BASIC session so that your files will be properly closed.

9-13. LIST

The LIST command enables you to copy a program or portion of a program onto a specified device.

Format

LIST [*limits*] [*filename* [:*security* [:*cartridge*]]]

Parameters

limits   beginning and ending line numbers of the portion of the program to be listed. Limits must be separated by a comma. If no limits are specified, the entire program currently in memory is listed. If one value is given, just one line is printed.

filename   name of the type 0 file corresponding to the device on which the program is to be listed. Default is LU 1 or the LU specified as the list parameter in the RUN,BASIC command.

security   optional security code.

cartridge   optional cartridge reference (label or LU number).

The LIST command enables you to list the program in memory at your terminal or onto a specified device. The program is listed in ascending sequence according to line numbers. You may request a partial list by specifying line number limits. FOR. . .NEXT statements are indented 2 spaces each for easy identification of program loops.

Examples

>LIST	*Lists the current program in memory.*
>LIST 120,180,FILEB	*Lists statements 120 through 180 from program in FILEB.*
>LIST FILEB	*Lists entire program in FILEB.*
>LIST 120	*List statement 120 only.*

9-14.   *BR,BASIC

The *BR,BASIC command permits you to interrupt an executing BASIC program or the listing of a program. It is used as follows:

- When a program is executing and you wish to interrupt it, press any key at your terminal.

- An asterisk (*) or *lu* > is printed.

- Enter BR,BASIC or BR and the name of the copy of BASIC you are using.

Control is returned to the Interpreter at this point and the message OPERATOR TERMINATION IN LINE xxx is printed on the console.

To break execution of a program which is waiting for a response to an INPUT,READ # *lu*, or PAUSE statement, type Control Q (Q^c) and press RETURN. If you are running from a multipoint terminal, type QQ.

Examples

*BR,BASIC	*Break the BASIC program.*
13>BR,BASCA	*Break a copy of the BASIC program named BASCA.*

9-15.   CALLS

The CALLS command prints a list of all the mnemonics in the current Mnemonics Table including the name, form (subroutine/function), type of parameters, and overlay number of each subroutine or function.

Format

```
CALLS
```

There are no attendant parameters to this command.

Example

```
        Parameter types          Overlay Number
  >CALL S
  IRSET(I,I)  F  0
  IEOR(I,I)  F  0
  OR(I,I)  F  0
  AND(I,I)  F  0
  NOT(I)  F  0                        } F = function
  ISHFT(I,I)  F  0
  IBTST(I,I)  F  0
  IBCLR(I,I)  F  0
  ISETC(RA)  F  0
  DAC(I,R)  S  0
  MPNRM  S  0
  RDWRD(I,IV)  S  0
  WRWRD(I,I)  S  0
  RDBIT(I,I,IV)  S  0
  WRBIT(I,I,I)  S  0
  SENSE(I,I,I,I)  S  0
  AISQV(I,I,RVA,IV)  S  1
  AIRDV(I,RA,RVA,IV)  S  1
  PACER(I,I,I)  S  1
  NORM  S  1
  SGAIN(I,R)  S  1
  RGAIN(I,RV)  S  1
  AOV(I,RA,RA,IV)  S  1
  MTTRD(I,RVA,I,IV,IV)  S  2
  MTTRT(I,RA,I,IV,IV)  S  2
  MTTPT(I,I,I)  S  2
  MTTFS(I,I)  S  2
  SFACT(R,R)  S  3                   } S = subroutine
  FACT(R,R)  S  3
  WHERE(RV,RV)  S  3
  PLOT(R,R,I)  S  3
  LLEFT  S  3
  URITE  S  3
  PLTLU(I)  S  3
  AXIS(R,R,RA,R,R,R,R)  S  3
  NUMB(R,R,R,R,R,I)  S  3
  SYMB(R,R,R,RA,R,I)  S  3
  LINES(RA,RA,I,I,I,R)  S  3
  SCALE(RVA,R,I,I)  S  3
  TIME(RV)  S  4
  SETP(I,I)  S  4
  START(I,R)  S  4
  DSABL(I)  S  4
  ENABL(I)  S  4
  TRNON(I,R)  S  4
```

9-16. TABLES

The TABLES command specifies which Branch and Mnemonic tables to use. This command is necessary if you intend to use external subroutines written in other languages. The Branch and Mnemonic Tables are created when running the BASIC Table Generator described in Section 14.

Format

TABLES *branch filename* [:*SC*[:*CR*]], *mnemonic filename* [:*SC*[:*CR*]]

branch filename	is the name of the file, created by the RTE BASIC Table Generator,which contains subroutine branch information.
mnemonic filename	is the name of the file, created by the RTE BASIC Table Generator, which contains subroutine mnemonics.
SC	is an optional security code parameter.
CR	is an optional positive cartridge number or negative logical unit number.

DEBUGGING COMMANDS

Multi-User Real-Time BASIC provides commands that enable you to debug a program while it is running. The path of execution through a program can be traced; program execution can be interrupted at critical points; and variables can be displayed and modified. In addition, subroutine calls can be displayed and simulated.

Real-Time BASIC commands used to perform the above tasks are defined in this section. Table 10-1 lists the commands, briefly states their functions, and delineates applicable use limitations.

Table 10-1. Debugging Commands

COMMAND	FUNCTION	USAGE LIMITS
TRACE	Turns on tracing of program or selected statements in the program.	Normal command mode or during a break.
UNTRACE	Turns off trace facility.	None.
BREAK	Causes program execution to be interrupted at from 1 to 4 specified break points.	None.
UNBREAK	Turns off break points set by break command.	None.
RESUME	Resumes program execution following a break point or a call simulation.	During break only.
ABORT	Aborts program execution following a break point.	During break only.
SIM	Simulates a subroutine call: causes a program to list a subroutine call when encountered, and suspends program execution.	None.
UNSIM	Turns off simulation facility.	None.
SHOW	Displays values of variables specified. Follows BREAK or SIM.	During break only.
SET	Permits entry of new variable content. Follows BREAK or SIM.	During break only.

10-1.   TRACE/UNTRACE

The TRACE command is used to turn on the BASIC facility to trace a program's execution from statement to statement. The UNTRACE command is used to turn the trace facility off.

Format

TRACE   [*limits*]
UNTRACE

Parameter

limits   beginning and ending line numbers of the statements within the program
                                to be traced. If omitted, the entire program is traced. If one statement is to
                                be traced, enter the line number of the statement.

The RUN command must be given to initiate the trace operation.

Once tracing begins, a message is printed for each traced statement executed. This message consists of an asterisk, the word TRACE, and the line number of the statement. You can follow the activity of the program by noting the statements listed as they are executed. If you use the TRACE command a second time, the new limits replace the existing ones.

The UNTRACE command may be given at any time a BASIC program is not executing.

Examples

```
>TRACE
>RUN
```
*Request that all of the current program state-
ments be traced.*

```
>TRACE 955
>RUN
```
Request that statement 955 be traced.

```
>TRACE 75,200
>RUN
*TRACE 75
*TRACE 80
    •
    •
    •
```
Statements 75 through 200 are traced.

10-2.   BREAK/UNBREAK

The BREAK command enables you to specify points within your program where program execution is to be suspended. A breakpoint can be a line number or range of line numbers. The UNBREAK command is used to eliminate scheduled breaks.

Format

```
BREAK     statement number label[,statement number label,....statement number label]

UNBREAK statement number label[,statement number label,....statement number label]

BREAK

Parameters

statement number label      execution suspends just before this statement. At least one
                            number must be specified, as many as four may be specified
                            separated by commas.
```

Once a breakpoint is reached, execution can only be resumed by entering a RESUME command. BREAK prompts for commands with two greater than (>) symbols.

To eliminate all breakpoints, enter the command UNBREAK prior to resuming program execution.

To eliminate specific breakpoints, enter the command UNBREAK and the line numbers of the breakpoints to be deleted.

To obtain a list of currently set breakpoints, enter BREAK with no parameters.

Legal commands that may be entered only during a break are: ABORT, RESUME, SET, and SHOW. Commands that may be entered during a break as well as other times include: BREAK/UNBREAK, CALLS, SIM/UNSIM, and TRACE/UNTRACE. If you enter any other command during a break, the system issues an error message. If you enter more than four statement number labels without using UNBREAK, an error message is printed.

Examples

>BREAK 30,70,90,100 >RUN	*Breakpoints are defined for lines 30,70,90, and 100.*
BREAK 30	*Breakpoint 30 is reached.*
>RESUME	*Resume execution by typing RESUME.*
BREAK 70	*Breakpoint 70 is reached.*
>UNBREAK 90	*Delete breakpoint 90.*
>RESUME	*Resume execution.*
BREAK 100	*Breakpoint 100 is reached.*
>	
BREAK 30,0,90,100	*Indicates breakpoints are set at statements 30, 90, and 100.*

10-3. RESUME

The RESUME command terminates current debugging activity and resumes execution of the suspended program. This command may be entered only during a break period, or after a subroutine simulation suspension.

Format

```
RESUME
RES


There are no attendant parameters to this command.
```

The RESUME command restarts the program at the location printed when the break occurred. You may abbreviate the command to RES.

Example

`>BREAK 25,100`	*Set breakpoints 25 and 100.*
`*BREAK 25`	*Breakpoint 25 is reached, program is suspended.*
`>>RESUME`	*Type RESUME command to resume execution of program.*
`*BREAK 100`	*Breakpoint 100 is reached.*

10-4.   ABORT

The ABORT command terminates a suspended program and returns the BASIC Interpreter to a non-executing program state. ABORT may be entered only during a break period or following a subroutine simulation suspension.

Format

```
ABORT


There are no attendant parameters for this command.
```

When ABORT is entered, the break period is ended and the run terminated. You may now enter any command legal during normal BASIC operation, but cannot enter commands legal only during a break period. Break points and trace are unchanged.

Example

`>BREAK 25,100`	*Set breakpoints 25 and 100.*
`*BREAK 25`	*Breakpoint reached, execution suspended.*
`>>ABORT`	*Type ABORT to terminate program execution.*
`BASIC READY` `>`	*BASIC is ready for next command.*

10-5.   SIM/UNSIM

The SIM command simulates a subroutine call. This command is identical to a breakpoint in as much as program execution is suspended just prior to the execution of the statement. The UNSIM command is used to turn off the simulation.

Format

```
SIM
UNSIM


There are no attendant parameters to these commands.
```

SIM is entered prior to program execution (or resumption). When a subroutine call is encountered, BASIC lists the subroutine call and suspends the program. The suspension message printed contains an asterisk, the statement number, the word CALL, the subroutine name, and pertinent variables. To illustrate:

    * 125 CALL AISQV (A,B(1),4,N)

You may display or modify variables at this point by using the SHOW and SET commands. To continue execution of the program, enter the RESUME command. Execution continues at the statement following the subroutine call.

To eliminate any further subroutine call simulation, enter UNSIM prior to resuming program execution.

To resume execution type RESUME; to terminate the program type ABORT.

Example

~ ...	*Request subroutine simulation.*
>RUN	*Execute the program.*
*125 CALL AISQV (A,B(1),4,N)	*BASIC prints information and suspends when*
>>SHOW B(3)	*subroutine is executed. Request value of B(3) variable.*
3.5	*BASIC prints the value.*
>>UNSIM	*Turn off subroutine simulation.*
>>RESUME	*Resume program execution.*

10-6.   SHOW

The SHOW command prints the values of the items specified. This command may be used only during a break period or subroutine call simulation.

Format

```
SHOW variable list


Parameters

variable list    numeric or string variables separated by commas. Expressions are not
                 allowed.
```

Example

`*BREAK 100`	*Set breakpoint 100.*
`>>SHOW X,Y,Z(10),A$`	*Request values of variables in list.*
`1.3    2.7    3.35544E+2    ABCE`	*BASIC prints values: X=1.3, Y=2.7, etc.*
`>>RESUME`	*Resume program execution by typing RESUME.*

10-7. SET

The SET command allows you to set any variable to a constant value. This command may be used only during a break period or following a subroutine call simulation.

Format

```
Set variable = constant [,variable=constant,. . .variable=constant]


Parameters

variable      numeric or string variable to be set to a value.

constant      new value to be given to the variable. Note:  Expressions are not allowed.
```

More than one variable may be set at a time. The equals sign must be used to equate the variable to a constant.

Example

`>RUN`	*Run the current program in memory.*
`*BREAK 100`	*Breakpoint 100 is reached.*
`>>SHOW A(20),B$(2,5)`	*Examine variables.*
`1.357    ABCD`	*BASIC prints values of variables.*
`>>SET A(20)=-1.357`	*Modify the value of variable A(20).*
`>>SET B$(2,5)=WXYZ`	*Modify the value of B$(2,5).*
`>>RESUME`	*Resume program execution.*

11-1. INTRODUCTION

Multi-User Real-Time BASIC is called *real-time* because the order of processing is governed by time or by the occurrence of external events rather than by a strict sequence defined in the program itself. Because these events can occur in a random order and require different amounts of processing, a real-time system must be capable of resolving conflicts between tasks.

A task is defined as a group of BASIC statements initiated by a call to one of the BASIC scheduling subroutines (e.g. START, TRAP, etc.) and terminated by a RETURN statement. Each task is uniquely identified by the line number of the first statement in the task. If a task is initiated by executing line 2000 of the BASIC program, that task will be represented in all scheduling calls as 2000. No blanks are allowed in the name or between the subroutine name and the left parenthesis preceding the parameter list.

> NOTE: Only one copy of the BASIC Interpreter can utilize TRAP and task
> scheduling subroutine calls at one time.

11-2. METHODS OF INITIATING TASKS

A task can be initiated in three different ways:

- at a specified time of day,
- after a specified delay,
- upon the occurrence of an external event.

To initiate a task at a given time of day, use the TRNON routine. For example:

```
100  CALL TRNON(2000,124505)
```

initiates task 2000 five seconds after 12:45 p.m.

The system clock must be set if it is to reflect the correct time-of-day. To do this, use the RTE operator command TM after loading the RTE Operating System.

To initiate a task after a delay, use the START routine. For example:

```
100  CALL START(2000,15)
```

executes task 2000 fifteen seconds after statement 100 is executed. This form of scheduling allows you to schedule a task repetitively. For example:

```
100    CALL  START(2000,15)      Schedule task 2000 in 15 seconds.
998    WAIT  (500)               Execute idle-loop.
999    GOTO  998
2000   CALL  START(2000,10)      Schedule task 2000 again in 10 seconds.
2010   PRINT "TASK 2000 AT 10 SECOND INTERVALS"   Execute task 2000.
2020   RETURN
3000   END
>RUN
TASK 2000 AT 10 SECOND INTERVALS
```

Lines 998 and 999 comprise an *idle-loop* and keep the program in execution mode indefinitely. When no task is executing, the program continuously cycles waiting for something to happen. All real-time programs should have some type of idle-loop. The idle-loop may be a useful statement such as a PRINT to continuously log the results of your calculations.

To terminate this example press any key on the system console or your terminal. The RTE system responds by printing an * or *lu* >. Type BR, BASIC (or BR, and the name of the copy of BASIC you are using) and press RETURN. The program will be aborted and Real-Time BASIC returns to command mode.

One common error occurs when a task reschedules itself every few seconds. For example, if a task reschedules itself every two seconds and the START statement is at the end of the task, the delay will be two seconds plus the amount of time necessary to execute the task statements which precede the START statement and the time required for all interim processing by the operating system. Therefore it is recommended that the statement used to reschedule the task be the first in the task in order to avoid the cumulative effect of the time delays.

If you give a series of consecutive task initiation commands, you should not specify a delay of zero (immediate start) for each one. The first task encountered must run to completion before another program statement can be executed. Subsequent tasks are delayed by the amount of time required to initiate preceding ones. To remedy this situation it is recommended that you schedule each task with a one second delay, thus allowing enough time for all of the initiating statements to be executed before any of the tasks is executed.

To initiate a task in response to an external event, the task must be associated with a trap number. For example:

```
100 TRAP 5 GOSUB 2000
```

associates task 2000 with trap number 5. Once the trap number is associated with the task, various events can refer to the trap number by using HP 6940 SENSE calls (for events sensed by the HP 6940) or a TTYS call (for an auxiliary teleprinter event).

Real-Time BASIC senses when an external event interrupt occurs, signals that the event has occurred, and records significant information about the event. The recording of an event takes place concurrently with the execution of a BASIC language statement. When statement execution completes, the BASIC Scheduler determines whether a task has been scheduled, compares its priority with the task currently executing and decides whether or not to suspend the current task in favor of the new one.

11-3.   PRIORITIES

Since a program may contain up to 16 tasks, more than one task may try to execute at the same time. To resolve these conflicts you may assign a priority number to each task. Priorities are established to provide some delineation between actions which must occur immediately and actions which have a more relaxed requirement. A task priority may be from 1 (the highest) to 99 (the lowest). A task of higher priority can interrupt the processing of a lower priority task if the interrupt scheduling the higher priority task occurs while the lower priority task is executing.

The following statement sets the priority of task 2000 to 50:

```
110 CALL SETP(2000,50)
```

Any task whose priority has not been specified is given a priority of 99.

11-4.   RESPONSE TIME

To determine how often and how quickly an event interrupt will be processed, you must consider the length of time required to complete execution of the BASIC statement being processed when the interrupt occurs. The amount of time required to process BASIC statements varies widely; a range of 0.5 to 3 milliseconds is typical. If it takes 3 ms to process a particular BASIC statement, and the event occurs after 1 ms has elapsed, then obviously the event will be noted but task execution will not begin for at least 2 ms. Since you do not know which statement will be executing when the interrupt occurs, the statement in your program with the longest execution time determines the maximum response time required to service an interrupt.

If you are doing real-time processing, it is recommended that you avoid statements which require operator intervention such as PAUSE, INPUT, and READ# *lu*. Interrupts may be lost while the program waits for a response from the operator.

Interrupt processing may also be delayed indefinitely or interrupts ignored as a result of the operator suspending the BASIC program with the RTE SUSPEND command or the operator entering RTE commands while BASIC is outputting to the system console or your terminal.

11-5.   THE BASIC SCHEDULER

The BASIC Scheduler keeps track of tasks and tells the Interpreter when to initiate execution of each task. The Scheduler maintains information passed to it by the task scheduling statements and uses it to make decisions concerning the tasks. The information includes the following:

* line number of the first statement in the task.
* priority of the task.
* trap number associated with the task.
* whether task is enabled or disabled.
* the current state of the task.

The BASIC Scheduler considers a task to be in one of five states at all times. For purposes of discussion, these states will be labeled A, B, C, D, and E. Refer to Figure 11-1.
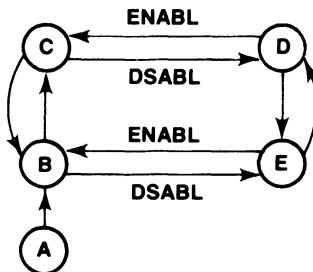
When the BASIC Interpreter completes execution of a line of BASIC code, it transfers control to the Scheduler. The Scheduler examines the real-time clock to see if any time scheduled tasks are ready for execution and puts them into State C (pending) or State D (pending/disabled). (Event scheduled tasks go to State C or D immediately upon occurrence of the event.) The Scheduler then determines which is the highest priority task in State C. If the priority of this task is higher than the currently executing task, the Scheduler puts it in State B (dormant) and tells the Interpreter to suspend the currently executing task and begin executing the highest priority pending task.

The BASIC Interpreter suspends the currently executing task and stores the line number of the next statement to be executed. It then stores the priority of the new task and begins executing the first line of that task. Task execution, once initiated, is independent of the Scheduler. The task can be interrupted but not disabled.

The Interpreter maintains the priority of the currently executing task which is determined when execution is initiated and does not change. It is independent of the priority kept by the Scheduler. Each time the Interpreter transfers control to the Scheduler after executing a statement, the priority is made available so the Scheduler can decide whether or not to suspend the currently executing task and initiate a new one. The Scheduler cannot change the priority of the currently executing task.

Once the Scheduler initiates a task, the execution of that task is controlled by the Interpreter. The Scheduler does not know which task is executing and is only concerned with which task should be initiated next. Thus, while a task is executing, it is possible that the same task may be initiated a second time. If its priority has been raised, the second initiation may interrupt the first execution of the task since the first retains the original priority.

State A — Undefined

A statement referencing the task has not yet been executed.

State B — Dormant

A statement referencing the task has been executed (the BASIC Interpreter has information about the task) but neither an external event nor the real-time clock has indicated that the task should be executed.

State C — Pending

An external event or the real-time clock indicate the task should be executed but execution has not yet begun. The task is actively vying for the computer resources. Execution is delayed until:

● the BASIC Interpreter completes execution of the current program statement, or

● a higher priority task completes execution.

State D — Pending/Disabled

A CALL DSABL has disabled the task which would otherwise be in State C (pending). The task is no longer vying for resources but once it is enabled, it will go directly to State C. A disabled task will not be initiated.

State E — Dormant/Disabled

A CALL DSABL has disabled the task which would otherwise be in State B (dormant). Once enabled, it will go directly to State B. If an interrupt indicates the task should be executed, it will go to State D (pending/disabled).

Figure 11-1. Task State Definitions

11-6. DSABL

The DSABL routine disables a specified task.

Format

CALL DSABL(*statement number label*) [FAIL: *statement*]

Parameter

statement number label	first statement number of the task to be disabled. If zero, all tasks are disabled. If negative, the task is removed from the task table (which controls the task), any previous scheduling is nullified.
FAIL: *statement*	optional error return statement.

The DSABL routine tells the BASIC Scheduler not to initiate a task or tasks. If the argument is positive, the task beginning with that statement is placed in State D or E as appropriate. If it is 0, all tasks are placed in State D or E. If the argument is negative, the task beginning with the statement number equal to the absolute value of the argument is placed in State A. Any task scheduling statement referring to a disabled task enables it (puts it in State B or C). If the task was in State A, any previous priority or trap number is lost.

Example

    350 CALL DSABL(2000)   *Disables task 2000.*

11-7. ENABL

The ENABL routine allows the scheduling of a previously disabled task.

Format

CALL ENABL(*statement number label*) [FAIL: *statement*]

Parameter

statement number label	first statement number of task to be enabled. If 0, all tasks are enabled.
FAIL: *statement*	optional error return statement.

A call to ENABL allows the initiation of a task which has been turned off previously by DSABL. A positive argument transfers the task from State D or E to State B or C. A zero argument transfers all tasks to State B or C. If the argument is negative, the SCHED-4 error message is printed and the ERRCD flag is set to 4. It may be interrogated with IERR.

Note that if a task has been disabled by a DSABL call, it may also be enabled by any other call (except DSABL).

You may use DSABL and ENABL to prevent a task from being interrupted. For example:

```
100 CALL START(1000,5)          Initiate task 1000 in 5 seconds.
    •
    •                           Disable all other tasks. Task 1000 is initiated
1000 CALL DSABL(0)              and then this statement is executed, it will not be
    •                           disabled. Processing proceeds uninterrupted to
    •                           1100.
1100 CALL ENABL(0)             Enable all tasks.
```

11-8. SETP

The SETP routine sets the priority of a task.

Format

CALL SETP(*statement number label, priority*) [FAIL: *statement*]

Parameters

statement number label	first statement in the task to have priority set.
priority	a number from 1 to 99. 1 is the highest priority and 99 is the lowest.
FAIL: *statement*	optional error return statement.

The priority of a task is used to resolve scheduling conflicts. Tasks with lower numbered priorities are selected for execution before tasks with higher numbered priorities. If no SETP call is made for a task its priority is 99.

Example

```
500 CALL SETP(1500,45)          Task 1500 is given priority 45.
```

11-9. START

The START routine schedules a task for processing after a specified delay.

Format

CALL START(*statement number label, sec*) [FAIL: *statement*]

Parameters

statement number label	first statement number of task to be initiated.
sec	number of seconds until execution is initiated.
FAIL: *statement*	optional error return statement.

If you CALL START with *sec* = 0 it is the same as using GOSUB, the task executes immediately and runs to completion. All subsequent tasks will be delayed by the execution time of this task unless they have a higher priority.

Example

```
655 CALL START(3300,35)        Task 3300 will be scheduled in 35 seconds.
```

11-10. TIME

The TIME routine returns the time according to the system real-time clock.

Format

CALL TIME(*time*)

Parameter

time   a variable equal to the time-of-day to the nearest tenth of a second. *time* is
            expressed as the number of seconds past midnight.

The following program converts the *time* parameter to hours, minutes, and seconds in the format
HH:MM:SS and prints the converted result every five seconds on LU 17.

```
>LIST
  10   LET L=17
  20   CALL START(100,6)
  29   WAIT (100)
  30   GOTO 29
 100   CALL START(100,5)
 110   CALL TIME(T1)
 120   LET S1=INT(T1/60)
 130   LET S=INT(T1-S1*60)
 140   LET H=INT(S1/60)
 150   LET M=INT(S1-H*60)
 160   PRINT #L;H;TAB(2);":";M;TAB(5);":";S
 170   RETURN
 180   END
>RUN
9      :49     :3
9      :49     :8
9      :49     :13
9      :49     :18
9      :49     :23
9      :49     :28
```

11-11. TRAP STATEMENT

The TRAP statement associates a trap number with a task which then may be associated with a hardware interrupt.

Format

TRAP *trapn* GOSUB statement number label

Parameters

trapn	trap number, a constant between 1 and 16 inclusive.
statement number label	first statement number of the associated task.

The trap number is a parameter in the HP 6940 SENSE routine, auxiliary teleprinter TTYS routine and the HP-IB SRQSN routine. For example:

CALL SENSE(*chan,nbit,bit,trapn*)

CALL TTYS(*lu,trapn*)

CALL SRQSN(*lu,trapn*)

When an interrupt to either of these routines occurs, the task associated with the *trapn* number is executed and the task is run. The TRAP association statement must already have been executed.

Only one trap number may be associated with each task and vice versa. Any attempt to associate more than one trap number to a statement number causes a SCHED-3 error, and the ERRCD flag to be set to 3. You may interrogate the ERRCD flag with IERR.

There are two methods of changing the association between a trap number and a task. Assume an association has been made as follows:

```
750 TRAP 5 GOSUB 1000
```

The first method is to simply assign a new task statement number as follows:

```
100 TRAP 5 GOSUB 2000
```

This forces the old task (1000) into State A (undefined, see figure 11-1), and nullifies any interrupts that have occurred to trap number 5. All future interrupts to trap number 5 will be transferred to statement 2000.

The second method is to use a negative statement number as follows:

```
100 TRAP 5 GOSUB -2000
```

The minus sign indicates you want to save any interrupts that have occurred to trap number 5. These interrupts will be transferred to statement 2000. The task at statement 1000 is forced to State A. All future interrupts to trap number 5 will be transferred to statement 2000.

If the error TRAP-1 is printed, the trap number is negative, the task was not found at syntax time, or the GOSUB part of the statement is missing.

After RESEQing TRAP statements with negative statements numbers, remember to put back the negative sign on those statements. The RESEQ command resequences properly but it can not put back the negative signs.

Examples

110 TRAP 3 GOSUB 2000	*After executing statement 110, trap number 3 is associated with task 2000 only.*

Do not change two values at once, you will get ambiguous results.

```
100 TRAP 3 GOSUB 1000
110 TRAP 4 GOSUB 2000
120 TRAP 3 GOSUB 2000
```
Statement 120 causes error SCHED-3 since task 2000 is associated with two trap numbers.

```
5    TRAP 7 GOSUB 170
10   SENSE(5,4,1,7)
•
•
•
160  GOTO 160
170  CALL WRBIT(2,4,1)
180  PRINT "RELAY CLOSED"
190  RETURN
```
Trap 7 transfers to statement 170.

A contact closure on bit 4 of channel 5 on a HP 6940 event sense card traps to statement 170.

This statement writes a bit on a channel.

A message is printed and control returns to the statement following the one completed before the interruption.

```
40   TRAP 7 GOSUB 960
50   SENSE(5,I,J,7)
60   CALL WRBIT(2,4,0)
•
•
•
960  PRINT "RELAY CLOSED"
970  TRAP 7 GOSUB 1000
980  CALL WRBIT(2,4,1)
990  RETURN
1000 CALL WRBIT(2,4,0)
1010 PRINT "RELAY OPEN"
1020 STOP
```
Set trap 7 to task 960.

First time SENSE interrupt occurs it traps to statement 960.

Task prints message.

Changes trap 7 so it is associated with task 1000.

The next time statement 50 is executed, the interrupt will trap to statement 1000.

11-12. TRNON

The TRNON routine executes a task at a specified time.

Format

CALL TRNON(*statement number label,time*) [FAIL: *statement*]

Parameters

statement number label   first statement of task to be initiated at specified time.

time   variable containing six digit number equal to HHMMSS, the time in hours, minutes, and seconds. Hours must be based on a 24 hour clock.

FAIL: *statement*   optional error return statement.

The call to TRNON starts a task when the real-time clock equals the *time* parameter.

Example

```
100 LET T=120015          Initialize T to 12 p.m. and 15 seconds.
110 LET I=5               Set increment = 5 seconds.
120 CALL TRNON(1000,T)    Schedule Task 1000 at 12 p.m. 15 sec.
999 GOTO 999        Idle-loop.
1000 LET T=T+I      Increment T by 5 seconds.        If sec = 60,
1010 IF T-INT(T/100)*100>=60 LET T=T+40     increment by 40 (minute by 1).
1020 IF T-INT(T/10000)*10000>=6000 LET T=T+4000
1030 IF T>=240000 LET T=T-240000          If min = 60, increment hr by 1
1040 CALL TRNON(1000,T)                    (min + sec by 4000).
                                          If hr = 24, reset to 0 hours.
.                   Reschedule task.
.
.                   Execute task code.
1900 RETURN         End of task.
```

11-13. TTYS

The TTYS routine allows an auxiliary teleprinter to interrupt the Real-Time BASIC system.

Format

CALL TTYS(*lu,trapn*)

Parameters

lu   logical unit number of the teleprinter.

trapn   trap number associated with a TRAP statement

A user at an auxiliary teleprinter can interrupt Real-Time BASIC by pressing any key and the trap associated with the logical unit number of the teleprinter will be executed. The routine does not service LU 1, the system console. If the error TTY-1 is printed, the logical unit number is less than 7. (LU 1 through 6 are reserved for standard devices.)

Example

```
10 TRAP 5 GOSUB 100
20 CALL TTYS(10,5)
29 WAIT(100)
30 GOTO 29
100 TIME(T)
110 PRINT#10;T
120 RETURN
```

Associate trap number 5 and task 100.

Define trap to be executed when interrupt occurs on LU 10. Insert idle-loop.

When a key closure generates an interrupt on LU 10, task 100 prints the number of seconds past midnight.

11-14. PROGRAM EXAMPLE

Figure 11-3 represents the structure of the working program shown in figure 11-4. The routine shown in figure 11-2 generates the data required by the program in figure 11-4.

Line 999 in figure 11-4 defines an idle-loop. In order to get some idea of how much time is spent in looping, a counter can be installed as part of the loop. Different variables are used for each task since one task may destroy the contents of variables used by another task. When a variable is to be shared by more than one task, you must analyze the implications of interrupts breaking into the statement sequence.

```
4 LET L4=4
5 LET L6=6
10 LET C1=20
20 LET C2=1
30 LET C3=10
40 LET C4=3
50 LET C5=5
60 LET C6=.1
70 LET M1=36
100 FOR I=1 TO 100
110 LET X=C6*I
120 GOSUB 1000
130 NEXT I
140 PRINT# L4;999
150 STOP
1000 LET Y=C1*COS(C2*X)+C3*SIN(C4*X)+C5-2*C5*RND(X)
1005 PRINT# L4;Y
1010 LET P=Y+M1
1020 IF P<M1 GOTO 1200
1030 IF P>M1 GOTO 1300
1040 PRINT# L6;TAB(M1);"X"
1050 RETURN
1200 PRINT# L6;TAB(P);"X";
1210 PRINT# L6;TAB(M1);"1"
1220 RETURN
1300 PRINT# L6;TAB(M1);"1";
1310 PRINT# L6;TAB(P);"X"
1390 RETURN
1400 END
```

```
27.8551      10.6984
25.2468       9.84983
26.9359       4.74023
32.7251       6.6409
32.4648       8.21229
26.0227       5.04531
23.1503       7.74416
23.0191      10.4607
12.6957      13.2719
12.1831       9.58012
 8.38185     19.0826
 3.45362     18.99
 -.724108    24.6351
-1.18706     22.214
-5.77463     29.8881
-7.25732     29.5723
-10.3784     28.426
-8.12697     21.0262
-15.2137     19.6844
-12.8658     16.6032
-6.25166     14.049
-5.85369     16.0063
-8.8297       6.51914
-4.75087      4.82018
-7.68372       .951558
-6.94039     -6.74977
-12.7432     -7.43395
-13.3699     -9.90004
-16.9722    -13.6216
-16.1321    -15.5273
-20.8142    -11.5115
-17.0646    -14.503
-22.8928     -9.28221
-24.363      -8.73984
-23.5851     -8.01055
-26.8436     -2.87477
-21.9709     -2.86945
-28.5017     -4.23474
-22.5021     -9.50069
-14.2186     -8.39849
-10.5363    -16.9804
-10.8942    -14.0062
 -.624686   -18.1544
 2.30019    -19.4267
 6.92991    -23.0569
 8.10668    -26.4572
12.2503     -26.6669
13.2217     -28.2566
15.5333     -23.6533
10.9137     999
 8.84168
```

NOTE: This routine generates the data at the right. The generated data is then fed into the routine following which demonstrates the scheduling capabilities of BASIC.

Figure 11-2. Task Scheduling Program Example (Part 1)

1-20 ASSIGN PERIPHERALS	
100-900 INITIALIZATION OF TASKS	
999 IDLE LOOP	
1000-1320 READING & PLOTTING DATA RETURN	TASK 1
2000-2150 PRINTING CALCULATION RESULTS ONTO TELETYPE RETURN	2
3000-3090 GIVE TIME AND SUMMARY CALCULATIONS WHEN KEYED RETURN	3
4000-4300 GIVE MESSAGE AT 8 AM RETURN	4
5000-5100 GIVE MESSAGE AT 12 NOON RETURN	5
6000-6100 GIVE MESSAGE AT 5 PM RETURN	6
7000-7030 GIVE MESSAGE AT 00:05 AM RETURN	7
8000-8120 BAD DATA PROCESSOR RETURN	8
9000-9060 CONVERT SECONDS TO HHMMSS RETURN	9

Figure 11-3.   Structure of Program Example in Figure 11-4.

```
1 LET L1=1
2 LET L2=11
3 LET L3=11
4 LET L4=4
5 LET L5=5
6 LET L6=6
10 LET L0=11
20 LET D=100
100 REM******** SET UP TASK 1000 ****
110 LET D1=1
120 LET A1=0
140 LET N1=0
160 SETP(1000,50)
170 START(1000,D1)
200 REM******** SET UP TASK 2000 ****
210 LET S2=0
220 LET N2=0
230 LET D2=10
240 SETP(2000,70)
250 START(2000,D2)
300 REM********SET UP TASK 3000 ****
310 SETP(3000,5)
320 TRAP 1 GOSUB 3000
330 TTYS(11,1)
400 REM*********SET UP TASK 4000 ****
420 SETP(4000,99)
430 TRNON(4000,80000)
500 REM*********SET UP TASK 5000 ****
510 SETP(5000,60)
520 TRNON(5000,120000)
600 REM*********SET UP TASK 6000 ****
610 SETP(6000,1)
620 TRNON(6000,170000)
700 REM*********SET UP TASK 7000 ****
710 TRNON(7000,105)
900 REM*********IDLE LOOP **********
999 GOTO 999
1000 REM
1002 REM ** TASK 1000 ** READ DATA AND PLOT IT*
1004 REM
1010 START(1000,D1)
1020 READ# L5;X1
1030 IF X1>36 OR X1<-36 GOTO 8000
1040 LET A1=A1+X1
1050 LET N1=N1+1
1060 LET P1=36+X1
1070 IF P1>36 GOTO 1200
1080 IF P1<36 GOTO 1300
1090 PRINT# L6;TAB(36);"X"
1100 RETURN
1200 PRINT# L6;TAB(36);"I";
1210 PRINT #L6;TAB(P1);"X"
1220 RETURN
1300 PRINT# L6;TAB(P1);"X";
1310 PRINT# L6;TAB(36);"I"
1320 RETURN
2000 REM
2002 REM ** TASK 2000 ** DATA COMPRESSION ROUTINE*
2004 REM
2010 START(2000,D2)
2015 GOSUB 9000
```

Initialize variables which assign outputs to specific devices.

Lines 100 through 900 schedule tasks 1000 through 7000.

Set priority = 50.

Tell Scheduler to initiate task 1000 D1 seconds from now.

Since D1=1, task 1000 executes 1 second after statement 170 executes. This allows time for all other tasks to be initiated.

Execute task 2000 in 10 seconds (D2=10.)

Set priority to 5.
Associate trap 1 with task 3000.
Associate LU 11 with trap 1. Pressing any key on LU 11 interrupts BASIC and starts task 3000.

Start task 4000 at 8 a.m.

Start task 5000 at 12 noon.

Start task 6000 at 5 p.m.

Start task 7000 at 1 min. 5 sec. after midnight.

Forces program to continue executing while waiting for tasks to be scheduled. Control returns here if no task is scheduled.

Note that task 1000 and 2000 will try to execute simultaneously. Task 1000 has a higher priority than task 2000 so will be scheduled first or may interrupt task 1000 which must wait.

The tasks defined in lines 1000 to 9060 are identical to subroutines.

Figure 11-4. Task Scheduling Program Example (Part 2)

```
2020 LET A2=A1/N1
2030 LET S2=S2+A1
2040 LET N2=N2+N1
2050 LET B2=A2/N2
2060 PRINT# L4;A2,N1,B2,N2,T
2070 LET A1=0
2080 LET N1=0
2100 PRINT#L1
2110 PRINT# L1;"AT TIME ";T;TAB(0);
2120 PRINT# L1;" THE AVERAGE IS ";B2;TAB(0);
2130 PRINT# L1;" FROM ";N2;TAB(0);" DATA POINTS."
2140 PRINT# L1;"THE AVERAGE FOR THE LAST PERIOD WAS ";A2
2145 PRINT# L1
2150 RETURN
3000 REM
3002 REM ** TASK 3000 ** EVENT SCHEDULED TASK **
3004 REM
3010 GOSUB 9000
3015 PRINT# L3
3020 PRINT# L3;"THE CURRENT TIME IS ";T;TAB(0);
3025 PRINT# L3;".  SO FAR WE HAVE"
3040 LET N3=N2+N1
3050 LET A3=(S2+A1)/N3
3070 PRINT# L3;N3;TAB(0);" DATA POINTS WITH AN AVERAGE OF ";A3
3080 PRINT# L3
3090 RETURN
4000 REM
4002 REM ** TASK 4000 **
4004 REM
4010 FOR I4=1 TO 8
4020 PRINT# L2;""
4030 WAIT(D)
4040 NEXT I4
4100 PRINT# L2;"GOOD MORNING "
4200 PRINT#L2;"I HOPE YOU HAD A GOOD NIGHTS REST "
4300 RETURN
5000 REM
5002 REM ** TASK 5000 **
5004 REM
5010 FOR I5=1 TO 12
5020 PRINT# L2;""
5030 WAIT(D)
5040 NEXT I5
5050 PRINT# L2;"TIME FOR LUNCH, LET'S EAT."
5100 RETURN
6000 REM
6002 REM ** TASK 6000 **
6004 REM
6010 FOR I6= 1 TO 5
6020 PRINT# L2;""
6030 WAIT(D)
6040 NEXT I6
6050 PRINT# L2;"TIME TO GO HOME, SEE YOU TOMORROW."
6100 RETURN
7000 REM
7002 REM ** TASK 7000 **
7004 REM
7010 PRINT# L2;"EITHER YOU ARE WORKING LATE, OR";
7020 PRINT# L2;" YOU FORGOT TO SET THE TIME"
7030 RETURN
```

Figure 11-4. Task Scheduling Program Example (Part 2) (Continued)

```
8000 REM
8002 REM ** THIS ROUTINE STOPS WHEN OUT OF DATA **
8004 REM
8010 PRINT "EITHER YOU HAVE RUN OUT OF DATA, OR YOU HAVE";
8020 PRINT " READ A BAD DATA POINT."
8120 STOP
9000 REM ** SUBROUTINE TO CONVERT TIME **
9005 TIME(T9)
9010 LET S9=INT(T9/60)
9020 LET S=T9-S9*60
9030 LET H=INT(S9/60)
9040 LET M=S9-H*60
9050 LET T=INT((H*100+M)*100+S)
9060 RETURN
9999 END
```

Figure 11-4. Task Scheduling Program Example (Part 2) (Continued)

11-15. TABLE PREPARATION

In order to use the task scheduling subroutines, you must add the names of the subroutines you want to use to the Branch and Mnemonic Tables. Generation of these tables is described in Section XIV. A list of the RTETG commands required to add the subroutines described in this section is given there.

11-16. ERROR MESSAGES

The following errors may result from execution of the task scheduling routines:

SCHED-2	Task table overflow.
SCHED-3	Impossible to resolve combination in TRAP statement.
SCHED-4	Attempt to ENABL or DSABL non-existent entry.
SCHED-5	Time schedule table overflow.
SCHED-6	Time to execute scheduled task, but its entry has been deleted from the Task Table.

In all cases the ERRCD flag is set to the error number. You may use the FAIL: option and IERR to interrogate the flag. See paragraphs 6-4 and 6-5.

Multi-User Real-Time BASIC performs all arithmetic operation in 32-bit floating point format (i.e., two 16-bit computer words). In instrument-related systems it is frequently necessary to manipulate the internal floating point number as though it were a 16-bit integer. This capability is especially important when the BASIC program communicates with instruments that require special bit patterns as input and/or output parameters. The following integer bit manipulation routines are designed to allow the BASIC programmer to perform inclusive OR, exclusive OR, NOT, AND, shift, set bit, clear bit, and test bit operations. These functions may be incorporated in the BASIC system at generation time by placing the proper name, entry point and parameter conversion in the Branch and Mnemonic table.

12-1. BIT MANIPULATION WORD FORMAT

Each word within the computer can be thought of as a 16-bit shift register. The bits are numbered from right to left as shown in Figure 12-1. Each set of three bits is weighted 4-2-1. For example, a bit pattern of 101 has an octal value of 5.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
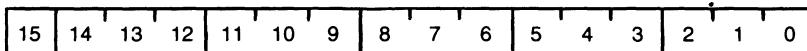
Figure 12-1. 16-Bit Word

For the purposes of bit manipulation, there are no sign bits or any significance other than a positional relationship.

In all cases of bit manipulation functions the operations are performed bit-by-bit. There is no carry from one set of bits to adjacent bits. Thus, in an OR function, for example, there is no carry from column-to-column.

12-2. AND

The AND function logically multiplies two words bit-by-bit.

Format

> A = AND *(arg1, arg2)*
>
> Parameters
>
> *arg1*   first value to be ANDed.
>
> *arg2*   second value to be ANDed.
>
> *result* A   returned value of the operation.

The AND statement requires that both of the values be "1" in order to have the product be "1". Refer to the following truth table:

arg1	0011
arg2	0101
result	0001

Example

```
10 X = ISETC("762")
20 Y = ISETC("543")
30 CALL AND(X,Y)
40 PRINT OCT(Z)
50 END
```

ISETC function described in paragraph 12-9.

Results

X	=762	=111-110-010
Y	=543	=101-100-011
Z	=542	=101-100-010

12-3. IBCLR (Bit Clear)

This function sets selected bit positions to zero. The position to be set to zero is determined by the number given in the *bit posit* parameter where the bits are referenced right to left starting with zero. Refer to Figure 12-1 for the bit positions. Note that only one bit can be "zeroed" at a time.

Format

A = IBCLR *(value, bit posit)*

Parameters

value   starting value.

bit posit   position of the bit to be cleared. The least significant bit is zero. If *bit posit* exceeds 15, the value of *result* is set to the starting value *(value)*.

result A   returned value.

Example

```
10 X = ISETC("767")
20 Z = IBCLR(X,8)
30 PRINT OCT(Z)
40 END
```

Results

X	=767	=111-110-111
		↓
Z	=367	=011-110-111

Bit position 8 is cleared (set to zero).

12-4.   IBSET (Bit Set)

This function sets selected bit positions to "1". The position to be set to one is determined by the number given in the *bit posit* parameter where the bits are referenced right to left starting with zero. Refer to Figure 12-1 for the bit positions. Note that only one bit can be set to "1" at a time.

Format

A = IBSET *(value, bit posit)*

Parameters

value   starting value.

bit posit   position of the bit to be set. The least significant bit is zero. If *bit posit* exceeds 15, the value *result,* is set to the starting value *(value)*.

result A   returned value.

Example

```
10 X = ISETC("452")
20 Z = IBSET(X,7)
30 PRINT OCT (Z)
90 END
```

Results

X	=452	=100-101-010
		↓
Z	=652	=110-101-010

Bit position 7 is set to 1.

12-5.   IBTST (Bit Test)

This function tests a selected bit in a word. It is used to return the value of a certain bit within a word without disturbing the original word.

Format

A = IBTST *(value, test posit)*

Parameters

value   value to be tested.

test posit   the position of the bit is to be tested. The least significant bit is zero. If *test posit* is greater than 15 or less than zero, *result* is set to zero.

result A   value of the tested bit (either a "1" or "0").

Example

```
10  Z = IBTST(X,15)
20  PRINT OCT(Z)
30  END
```

X = the value to be tested.

15 = the left-most bit, or bit 15.

Z = a "1" or "0" depending on the value of bit 15 in word X.

12-6.   IEOR

The IEOR function executes the modulo-two sum (Exclusive OR) between two words, bit-by-bit.

Format

A = IEOR (*arg1, arg2*)

Parameters

arg1   first value to be Exclusive OR'ed.

arg2   second value to be Exclusive OR'ed.

result A   returned value of the operation.

The exclusive OR operation is primarily used in compare operations. When performing the exclusive OR, the result will be "1" if one and only one of the bits is "1". Refer to the following truth table:

arg1	0011
arg2	0101
result	0110

Example

```
10  X = ISETC("564")
20  Y = ISETC("371")
30  Z = IEOR(X,Y)
40  PRINT OCT(Z)
50  END
```

Results

```
X  =564  =101-110-100
Y  =371  =011-111-001
Z  =615  =110-001-101
```

12-7.   NOT

This function complements a word bit-by-bit. It causes the complement of a value to appear in the result. That is, a "1" in *value* becomes a "0" in *result* and vice versa.

Format

```
A = NOT (value)

Parameters

value        value to be complemented.

result A     returned value of the operation.
```

Example

```
10 CALL ISETC("1762",X)
20 Z=NOT(X)
30 PRINT OCT(Z)
40 END
```

Results

X	= 1762	=0-000-001-111-110-010
Z	=176015	=1-111-110-000-001-101

12-8.   OR

The OR function executes the logical sum (Inclusive OR) between two words bit-by-bit.

Format

```
A = OR (arg1, arg2)

Parameters

arg1         first value to be Inclusive OR'ed.

arg2         second value to be Inclusive OR'ed.

result A     returned value of the operation.
```

The Inclusive OR operation is primarily used in compare operations. When performing the Inclusive OR, the result is "1" if either of the bits in argument 1 or 2 is "1". Refer to the following truth table.

arg1	0011
arg2	0101
result	0111

Example

```
10 X = ISETC("461")
20 Y = ISETC("577")
30 Z = IOR(X,Y)
40 PRINT OCT(Z)
50 END
```

Results

X	=461	=100-110-001
Y	=577	=101-111-111
Z	=577	=101-111-111

12-9. ISETC (Set to Octal)

ISETC converts the octal number to its floating point equivalent. Therefore, whenever using the PRINT statement to print the number, use the OCT option. Otherwise, the decimal equivalent of the number is printed. For example, if you use the octal number 177777, and the PRINT statement without the OCT option, the decimal number −1 is printed.

Format

$$A = \text{ISETC}\begin{pmatrix} \text{"}octal\ numb\text{"} \\ string\ variable \end{pmatrix}$$

Parameters

octal numb	six-character or less octal number enclosed in quotes (" ") which is to be set into *variable*. Any characters other than 0 through 7 will cause an error 26.
string variable	variable containing a string of numeric characters.
variable A	parameter that receives the octal number.

Example

```
10 X = ISETC("27765")
20 PRINT OCT(X)
30 END
```

Results

X is printed as 27765.

12-10. ISHFT (Register Shift)

This function shifts the bit contents of a variable left or right a given number of positions. It shifts the entire word as though the word were a shift register. The bits shifted out of the word are lost. Replacement bits coming into the word are zeros.

Format

A = ISHFT (*value, shift numb*)

Parameters

value	argument to be shifted.
shift numb	direction of shift and number of positions.
numb <0	shift right "n" positions.
numb >0	shift left "n" positions.
numb =0	no shifting.
numb >±15	result is set to zero.
result A	returned value of the operation.

Example

```
10 X = ISETC("276")        Shifts all bits two positions left.
20 Z = ISHFT(X,2)
30 PRINT OCT(Z)
40 END
```

Results

```
X   = 276    =010-111-110
Z   =1370    =001-011-111-000
```

12-11. BRANCH AND MNEMONIC TABLE PREPARATION

In order to use the routines described in this section, you must enter the names of the functions and other information in the Branch and Mnemonic Tables. The procedure for doing this is described in Section XIV with a list of the precise commands required.

MAGNETIC TAPE I/O

The magnetic tape drive enhances BASIC's effectiveness by providing sorting, manipulation, storage, and retrieval capabilities. The magnetic tape drive allows the system to save information that is too voluminous for storing in memory, on disc, or on paper tape. There are two primary advantages to using magnetic tape over paper tape:

- The speed and capacity of a magnetic tape.
- The ability to rewind and reread material solely on command from the computer.

Data is written to the tape drive as a set of contiguous data called a *record*. Each tape record is a direct result of a command from the program to write a record onto a magnetic tape. Sets of consecutive records all associated with the same logical program and function are called *files*. It is possible to have more than one file on a single tape, and, except for the fact that they are on the same physical reel, they do not necessarily have to be related. At the end of each of the files, the program can write a special record called an end-of-file (EOF) which signifies the end of a file.

You should not mix READ and WRITE commands as you progress through a tape. Since records are not spaced a precise distance apart, extraneous data may be left on the tape if you write a record and then read the record which follows it. Always write onto a tape consecutively and then rewind and read it later. It is unacceptable to update a tape by replacing records.

13-1. MAGNETIC TAPE OPERATOR COMMANDS

There are several operator commands available that allow you to manipulate the tape drive from the system console. The command format consists of a control word and file name.

Format

Command	Purpose
REWIND *mag tape unit filename*	Rewind the magnetic tape all the way back to the beginning.
WEOF *mag tape unit filename*	Write an end-of-file mark on the tape.
SKIPF *mag tape unit filename*	Skip to the end of the current file and stop at the beginning of the next file on the magnetic tape.
BACKF *mag tape unit filename*	Backspace past the previous file mark and stop.

Parameter

mag tape unit filename	name of a type 0 file created by File Manager CR command and corresponding to a magnetic tape unit. For example: :CR,MT,8,BO creates a file named MT corresponding to LU 8. See the Batch-Spool Monitor Reference manual for more information.

13-1

13-2.   MAGNETIC TAPE CALLS

Descriptions of all magnetic tape calls available in Real-Time BASIC are provided in the remainder of this section. Figure 13-2 contains a sample program which uses the routines described here.

13-3.   MTTRT

The MTTRT routine writes a record onto a tape.

Format

CALL MTTRT(*lu,array,numb,eof,length*)

Parameters

lu	logical unit number of magnetic tape device.
array	first element of an array of data to be written.
numb	number of variables requested to be transferred from memory to the tape.
eof	dummy variable always set to zero on return.
length	actual number of variables transferred from memory to tape.

The number of variables requested to be transferred is always repeated in the *length* parameter since a tape WRITE always writes the specified number of variables. If there is a problem with writing on a bad portion of tape, the statement skips and rewrites on fresh tape.

Example

```
355  CALL MTTRT(13,A(1),100,  E,  N)    Writes 100 elements of array A on LU 13.
```

13-4.   MTTRD

The MTTRD routine reads a record from a tape into an array.

Format

CALL MTTRD(*lu,array,numb,eof,length*)

Parameters

lu	logical unit number of the magnetic tape device.
array	first element of an array into which data is read.
numb	number of variables requested to be read from the tape.
eof	variable set to 1 if EOF is encountered during a READ operation, set to 0 otherwise.
length	actual number of variables transferred from the tape to memory. Useful when the record read is shorter than the length specified by *numb*.

The routine transfers up to *numb* variables. If the record length is smaller than *numb*, only the record is transferred. If the record is larger than *numb*, the remaining data in the record is lost.

The *length* parameter provides the actual number of variables transferred to the array. *Length* can never exceed *numb*; therefore, if *length* = *numb*, the record may have been too long.

After the last record in a file is read, the next READ returns an end-of-file indication (if there is one) in parameter *eof*.

Example

```
500  CALL MTTRD(12,B(1),200,E,N)     Read 200 elements into array B from LU 12.
```

13-5.   MTTPT

The MTTPT routine positions the tape forward or backward a certain number of files and/or records.

Format

CALL MTTPT(*lu,fspace,rspace*) Parameters *lu* — logical unit number of the magnetic tape device. *fspace* — number of files to skip. If positive, skips forward, if negative, skips backward. *rspace* — number of records to skip. If positive skips forward, if negative, skips backward.

Forward positioning is accomplished by reading files or records until the number you request have been skipped or an EOF is read.
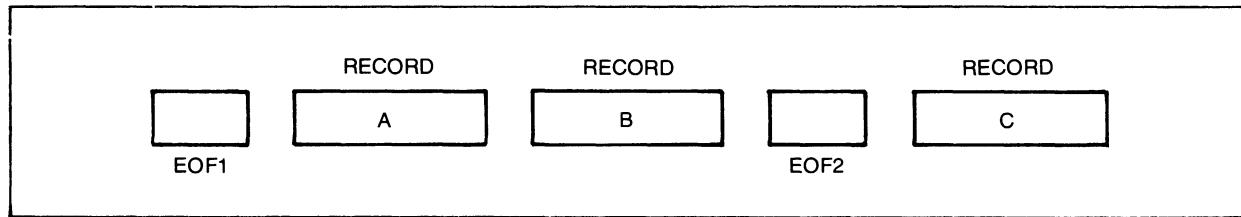
Figure 13-1.   Record Positioning Example Using MTTPT.

Examples (Refer to figure 13-1)

1. File position is immediately after record C, and *fspace* = −1 (backspace 1 file). File position moves to immediately after record B before EOF2.

2. File position is immediately after record C; *fspace* = −2 (backspace 2 files) and *rspace* = 1 (forward space 1 record). File position moves to immediately after EOF1 before record A.

3. File position is immediately after record C, and *rspace* = −3 (backspace 3 records). File position moves between record A and B.

13-6. MTTFS

The MTTFS routine writes an end-of-file and rewinds the tape.

Format

CALL MTTFS(*lu,func*)

Parameters

lu   logical unit number of the magnetic tape device.

func   specific tape drive control or function:

   0 = WRITE a 4 inch gap.

   1 = WRITE an EOF mark.

   2 = REWIND the tape but leave the tape, when finished, available for use again.

   3 = REWIND the tape and when the rewind is complete, make the tape device no longer available to the system (rewind and unload).

Examples

435 CALL MTTFS(12,1)   *WRITE an EOF on LU 12.*

860 CALL MTTFS(11,2)   *REWIND LU 11. Control returns to program immediately.*

755 CALL MTTFS(11,3)   *REWIND LU 11 and make tape device unavailable.*

950 CALL MTTFS(10,0)   *Erase 4 inches of tape on LU 10.*

13-7. TAPE MANIPULATION ERRORS

The following message is given when tape errors occur:

   ERROR MAGTP-*n* IN LINE *nnnn*

If n equals:

   1   An illegal logical unit number has been specified for a tape.

   2   An illegal request for tape manipulation has been made.

   3   A WRITE request has been given but the write-ring (file-protect-ring) is not in place on the reel.

The line in the BASIC program where the error occurred is supplied as *nnnn*.

13-8.   BRANCH AND MNEMONIC TABLE ENTRIES

During system generation, the routines described in this section must be entered in the Branch and Mnemonic Tables if the magnetic tape drive calls are to be used. The procedure for doing this is described in Section XIV.

13-9.   Sample Program Using Magnetic Tape

Figure 13-2 contains a sample program which demonstrates some of the previously described routines. The tape is rewound, file records are written on it, it is rewound again and then read until an end-of-file is encountered. The tape is then backspaced twice to allow the last data record to be reread.

```
>LIST
    10   REM - MAGNETIC TAPE EXAMPLE
    20   DIM V[100],W[100]
    30   REM - REWIND THE TAPE UNIT TO LOAD POINT
    40   CALL MTTFS(8,2)
    50   REM - WRITE FIVE RECORDS
    60     FOR I=1 TO 5
    70       CALL MTTRT(8,V[1],100,X,E)
    80     NEXT I
    90   REM - WRITE END-OF-FILE MARK
   100   CALL MTTFS(8,1)
   110   REM - REWIND THE TAPE UNIT TO LOAD POINT
   120   CALL MTTFS(8,2)
   130   REM - NOW READ ALL THE RECORDS
   140   CALL MTTRD(8,W[1],100,E,N)
   150   IF E=1 THEN 170
   160   GOTO 130
   170   REM - - BACKSPACE TWO RECORDS
   180   CALL MTTPT(8,0,-2)
   190   END
>
```

Figure 13-2. Tape Control Sample Program

In order to call external subroutines and functions written in BASIC, FORTRAN, ALGOL, or Assembly language, you must use the RTE Table Generator to define and generate two tables, the Branch and Mnemonic Tables, and create overlays which contain the actual subroutines.

When a specific subroutine is called, the module or overlay is loaded from the disc into a foreground partition of memory. Remember that BASIC resides in a background partition. Therefore, BASIC and the overlay reside in memory concurrently, at least initially. After control is passed to the subroutine, the BASIC Interpreter may be swapped out to the disc if necessary. All of the information that allows BASIC to access the correct overlay is contained in the Branch and Mnemonic Tables. Additional information is contained in the overlay which is used to pass control to the correct subroutine in the overlay.

BASIC uses the Branch and Mnemonic Tables to transfer program execution from BASIC to the subroutine or function and back. It converts parameters and locates the subroutine overlay and the subroutine within the overlay.

Figure 14-1 illustrates the relationship of the BASIC Interpreter and a subroutine overlay in memory.
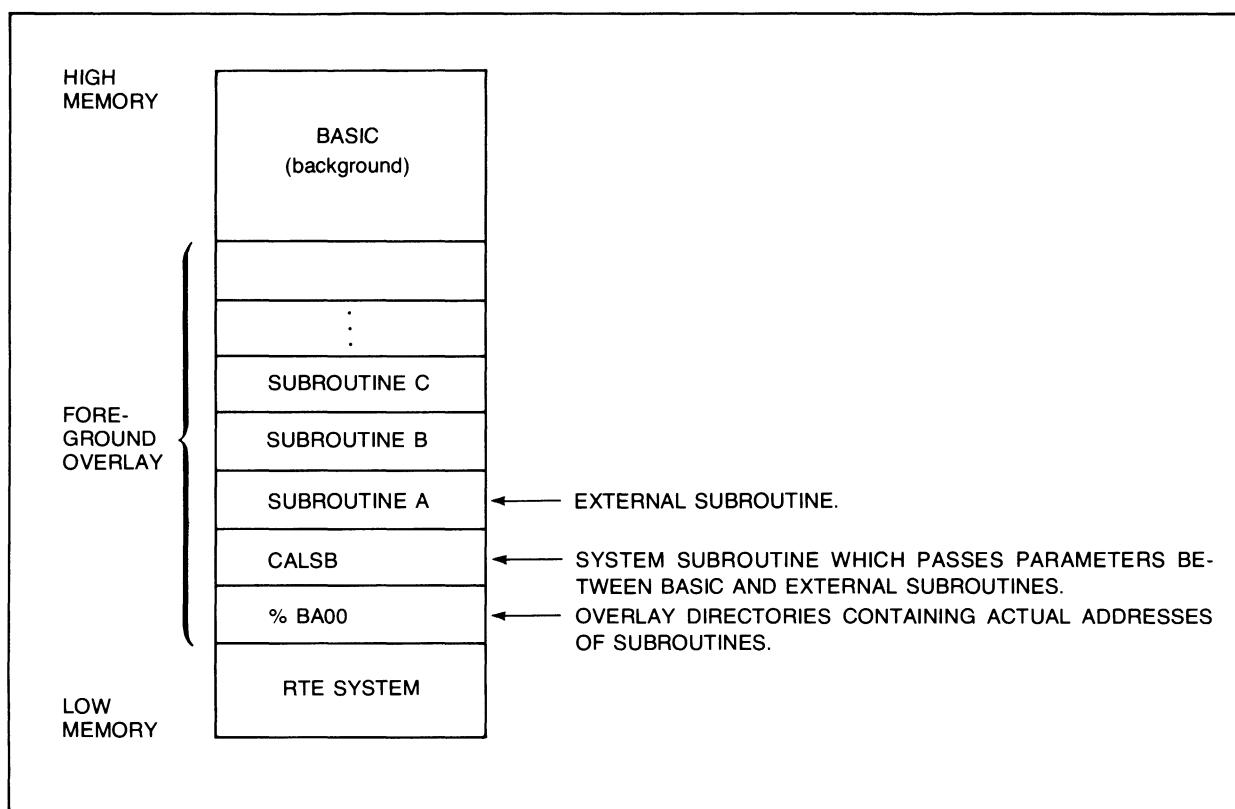
Figure 14-1. BASIC and an Overlay in Memory

14-1.   RTETG

The RTE Table Generator (RTETG) operates in batch mode. You can provide the description of a subroutine or function from a type 3 or 4 disc file or type 0 file only (for non-interactive devices only, such as cards, paper tape, or magnetic tape).

You specify:

* the name of the subroutine or function as it is called in your BASIC program,

* the type of each parameter (integer, real, or array) used by the routine,

* the entry point name of the routine if different from the name as used in your BASIC program,

* the name of the file containing the relocatable subroutine,

* overlay groupings.

RTETG processes this information and produces the Branch Table, the Mnemonic Table, overlay directories, and a FMGR transfer file which is used to load the overlays. Each overlay directory is a relocatable program which allows execution to be transferred to the subroutine or function when it is called. When the overlay is loaded, it is linked to all routines defined as part of the overlay.

A single overlay may be used for up to 64 subroutines. For efficiency you should arrange to have all subroutines and functions which are likely to be used consecutively in the same overlay. The maximum number of overlay directories (and therefore, overlays) is 32.

As many as 26 pairs of tables and sets of overlays may be defined for each version of BASIC on a system. Each time you run BASIC, you specify which tables you are using. In this way, you can optimize efficiency by setting up tables which are appropriate for your program. Overlays can not be shared by another copy of BASIC.

RTETG is a segmented program and requires a minimum of 5K of system background area.

14-2.   SCHEDULING RTETG

When you schedule RTETG you must specify the name of a file where the commands defining your subroutines are to found. The file may be a type 3 or 4 File Manager file or a type 0 file corresponding to a non-interactive input device such as the card reader or a terminal. You may optionally specify the logical unit number of the list device where commands will be printed.

The command to start RTETG is:

> :RUN,RTETG,*filename*[*:security*[*:cartridge*]][,*list*]

or

> *RUN,RTETG,*filename*[*:security*[*:cartridge*]][,*list*]

> *filename*   name of a file. (Cannot specify a logical unit number.)

> *security*   optional security code. Must be specified if file was created with a security code.

> *cartridge*   optional cartridge reference (label or negative LU).

> *list*   is the logical unit number of the list device. Default is the user's terminal.

The RTETG commands are described below.

14-3.   THE FIRST RTETG COMMAND

The first command you use must specify the files to be used for the Branch Table, the Mnemonic Table, the transfer file, and an identification letter to identify the group of overlay directories to be produced. The command is:

brt[:*sc*[:*cr*]] ,*mnt*[:*sc*[:*cr*]] ,*trf*[:*sc*[:*cr*]] ,ID=*i*[,*prior*[,*sec*[,*cref*]]]

brt is the name to be given to the Branch Table file.

sc   is the security code to be assigned to the file it follows. The value may be an integer between −32767 and +32767 or two ASCII characters. Default equals 0, no security code. The first ASCII character may not be a number.

cr   is the cartridge reference number. A positive integer between 1 and 32767 is the label of the cartridge on which you want the file to reside. A negative number is the logical unit number of the cartridge. Default equals 0, use any cartridge.

mnt is the name to be given to the Mnemonic Table file.

trf   is the name to be given to the transfer file.

ID=*i* indicates the upper case alphabetic character (A-Z) to identify your set of Overlay Directories since there may be more than one set of overlays defined in your RTE system. The ID= must always precede the *i* parameter.

prior is the priority assigned to the overlay directory when it is scheduled. Default is 80.

sec   is the security code to be used with the overlay directories. The permissible values are the same as for the *sc* parameter.

cref is the cartridge reference number to be used with the overlay directories. Default is 0, no cartridge reference. The permissible values are the same as for the *cr* parameter.

The three file names may be any legal File Manager file names.

14-4.   OTHER RTETG COMMANDS

For each subroutine or function you want to call from a BASIC program, you must use the following command:

name[(*p1,p2, . . . ,pn*)],OV=*nn*[,SZ=*mm*][,SS] $\begin{bmatrix} \text{INTG} \\ \text{,REAL} \end{bmatrix}$ $\begin{bmatrix} \text{FP} \\ \text{FT} \\ \text{,BP} \\ \text{,BT} \end{bmatrix}$ [,VL][,ENT=*epoint*][,FIL=*fname*[:*sc*[:*cr*]]]

name   is the subroutine or function name which may be from 1 to 5 characters.

(*p1,p2, . . . , pn*) is a list of 1 to 15 parameter types to be passed to the subroutine if it requires them. The list must be enclosed in parenthesis. If there are no parameters, put parenthesis without any parameters. *pn* equals:

$\begin{bmatrix} \text{I} \\ \text{R} \end{bmatrix}$[V][A]

(I=integer, R=real, V=value is returned from the subroutine, A=array variable) (1 <= *n* <= 15). For example, SUBX(I) or SUBZ(IV,RVA). To pass a string variable, RA or RVA should be specified. Note that the simple variable A is different from Array (A). Default is R for real. Refer to Section II for a description of variables.

From this point, the following parameters may be specified in any order.

OV=*nn*   indicates the overlay in which the subroutine will reside. The OV = must always precede *nn*. (0 <= *nn* <= 31) overlays must be specified in ascending order.

SZ=*mm*   indicates the number of pages required for the overlay. This parameter is used for all systems except RTE-II and need appear only once. If it appears more than once, the largest value supplied is used and a warning is printed. (1 < = *mm* < = 32) Specify at least one page more than the actual program size to allow enough room for parameter passing. The message ERROR NO MEMORY-1 IN LINE XXX will occur when there is a program call to an external subroutine and insufficient space is available to store the passed parameters. The SZ command referring to the flagged subroutine should be modified to provide for enough space in memory to store the parameters passed from the calling program.

SS   indicates the overlay is to reference SSGA (RTE-III and RTE-IV only)

INTG   is used if you are defining a function to indicate the function value is returned as an integer. If this parameter is used, the routine is automatically a function.

REAL   is used to indicate the function value is returned as a floating point number. If this parameter is used, the routine is automatically a function.

FP, FT, BP, or BT   specifies that the overlay is real-time Permanent, real-time Temporary, Background Permanent, or Background Temporary. The default is FP.

VL   is used to indicate that the length of the subroutine call parameter list is variable. BASIC supports the calling of subroutines using a variable length parameter list only if the subroutine being called has been written to handle a variable length list. Remember that FORTRAN programs can not handle a variable length list. BASIC always passes 15 parameter addresses to the subroutine. Unused parameters are zeroed out. Each parameter and its type must be specified in the RTETG command entry. BASIC then will allow execution of subroutine calling sequences with less than the maximum number of parameters.

For example, the RTETG command:
SUBX (RA,RA,I,I,IV) ,VL,OV=2

will allow BASIC to accept the call:
10 CALL SUBX (A,B,C)

but not the call:
10 CALL SUBX (A,B,C,D,E,F)

ENT=*epoint*   specifies the 1 to 5 character subroutine entry point name. If this name is identical to the first five characters of name, it need not be specified. If it is different than the first 5 characters of name, it must be specified. ENT = must precede the entry point name. If greater than 5 characters, an error is issued.

FIL=*fname*   specifies the name of a File Manager relocatable file where the routine resides. The FIL=must always precede the file name. If you do not supply a name when you run the transfer file, the disc resident libraries are searched for the routine. If the routine is not found, the Loader prompts for the name at the user's terminal when you load a program which uses the routine.

sc   is the security code of the file named *fname.*

cr   is the cartridge reference (label or LU) of the cartridge on which the file named *fname* resides.

Constant numbers, string literals, and expressions cannot be used as parameter values when calling a subroutine if the parameter has been defined as type V (returned from subroutine). Functions cannot be used in a parameter list.

14-5. RTETG OUTPUT FILES

RTETG creates two separate binary files for the Branch and Mnemonic Tables and stores them in type 7 File Manager files.

The Overlay Directories are produced as separate standard relocatable programs named % BX*nn* and are stored in type 5 files. The third letter (x) of the program name and the entry point name vary according to the ID letter supplied with the command which defines the subroutine. The file name corresponds to the entry point name. The last two characters of the name *(nn)*, are digits which indicate the overlay number specified in the OV = command for each subroutine.

14-6. RTETG COMMANDS REQUIRED FOR LIBRARY SUBROUTINES

In order to use the subroutines and functions described in various sections of this manual, you must enter the subroutine and function names in the Branch and Mnemonic Tables. You may select from figure 14-2 the commands for the particular subsystems and subroutines you want to use. Always refer

		Type of Memory Residence	Overlay Number	Function Value is Integer	Entry Point Name	Name of File containing Routine
First Command	BTBL,MTBL,TRFL,ID=A					
Data Conversion	DCODE(RVA,RVA,RA),	BP,	OV=0,		ENT=DCODE	
	NUM(RA),	BP,	OV=0,	INTG,	ENT=NUM	
	CHRS(I,RVA)	BP,	OV=0,		ENT=CHRS	
Bit Manipulation	IBSET(I,I),		OV=1,	INTG,	ENT=BBSET	
	IEOR(I,I),		OV=1,	INTG,	ENT=BEOR	
	OR(I,I),		OV=1,	INTG,	ENT=BIOR	
	AND(I,I),		OV=1,	INTG,	ENT=BAMD	
	NOT(I),		OV=1,	INTG,	ENT=BNOT	
	ISHFT(I,I),		OV=1,	INTG,	ENT=BSHFT	
	IBTST(I,I),		OV=1,	INTG,	ENT=BBTST	
	IBCLR(I,I),		OV=1,	INTG,	ENT=BBCLR	
	ISFIC(RA),		OV=1,	INTG,	ENT=ISETC	
HP 6940 Subsystem	DAC(I,R),		DV=2,	SS	ENT=DAC,	FIL=%694BS
	MPNRM,		DV=2,	SS	ENT=MPNRM,	FIL=%694BS
	RDWRD(I,IV),		DV=2,	SS	ENT=RDWRD,	FIL=%694BS
	WRWRD(I,I),		DV=2,	SS	ENT=WRWRD,	FIL=%694BS
	RDBIT(I,I,IV),		DV=2,	SS	ENT=RDBIT,	FIL=%694BS
	WRBIT(I,I,I),		DV=2,	SS	ENT=WRBIT,	FIL=%694BS
	SENSE(I,I,I,I),		DV=2,	SS	ENT=SENSE,	FIL=%694BS
HP 2313/91000 Subsystems	AISQV(I,I,RVA,IV),		DV=3,	SS	ENT=AISQV,	FIL=%A2313
	AIRDV(I,RA,RVA,IV),		DV=3,	SS	ENT=AIRDV,	FIL=%A2313
	PACER(I,I,I),		DV=3,	SS	ENT=PACER,	FIL=%A2313
	NORM(I),		DV=3,	SS	ENT=NORM,	FIL=%A2313
	SGAIN(I,R),		DV=3,	SS	ENT=SGAIN,	FIL=%A2313
	RGAIN(I,RV),		DV=3,	SS	ENT=RGAIN,	FIL=%A2313
	AOV(I,RA,RA,IV),		DV=3,	SS	ENT=AOV,	FIL=%A2313
Magnetic Tape Subsystem	MTTRD(I,RVA,I,IV,IV),		OV=4,		ENT=MTTRD	
	MTTRT(I,RA,I,IV,IV),		OV=4,		ENT=MTTRT	
	MTTPT(I,I,I),		OV=4,		ENT=MTTPT	
	MTTFS(I,I),		OV=4,		ENT=MTTFS	

Subroutine or Function Name   P $_n$ Parameters

Figure 14-2. RTETG Commands for Library Subroutines

		Type of Memory Residence	Overlay Number	Function Value is Integer	Entry Point Name	Name of File containing Routine
Plotter Subsystem	SFACT(R,R),		OV=5,		ENT=SFACT,	FIL=PLOTR
	FACT(R,R),		OV=5,		ENT=FACT,	FIL=PLOTR
	WHERE(RV,RV),		OV=5,		ENT=WHERE,	FIL=PLOTR
	PLOT(R,R,I),		OV=5,		ENT=PLOT,	FIL=PLOTR
	LLEFT,		OV=5,		ENT=LLEFT,	FIL=PLOTR
	URITE,		OV=5,		ENT=URITE,	FIL=PLOTR
	PLTLU(I),		OV=5,		ENT=PLTLU,	FIL=PLOTR
	AXIS(R,R,RA,R,R,R,R),		OV=5,		ENT=AXIS,	FIL=PLOTR
	NUMB(R,R,R,R,R,I),		OV=5,		ENT=NUMB,	FIL=PLOTR
	SYMB(R,R,RA,R,I),		OV=5,		ENT=SYMB,	FIL=PLOTR
	LINES(RA,RA,I,I,I,R),		OV=5,		ENT=LINES,	FIL=PLOTR
	SCALE(RVA,R,I,I),		OV=5,		ENT=SCALE,	FIL=PLOTR
HPIB Subsystem	HPIB(I,I,I),		OV=6,SS,		ENT=HPIB	
	SRQ(I,I,RA),		OV=6,SS,		ENT=SRQ	
	SRQSN(I,I)		OV=6,SS,			
	CMDR(I,RA,RVA),		OV=6,SS,		ENT=CMDR	
	CMDW(I,RA,RA),		OV=6,SS,		ENT=CMDW	
	IBERR(I),		OV=6, INTG,SS,		ENT=IBERR	
	IBSTS(I),		OV=6, INTG,SS,		ENT=IBSTS	
Task Scheduling	TIME(RV),		OV=7,SS,		ENT=TIME,	FIL=%TSKSC
	SETP(I,I),		OV=7,SS,		ENT=SSETP,	FIL=%TSKSC
	START(I,R),		OV=7,SS,		ENT=SSTRT,	FIL=%TSKSC
	DSABL(I),		OV=7,SS,		ENT=DSABL,	FIL=%TSKSC
	ENABL(I),		OV=7,SS,		ENT=ENABL,	FIL=%TSKSC
	TRNON(I,R),		OV=7,SS,		ENT=TRNON,	FIL=%TSKSC
	TTYS(I,I),		OV=7,SS,		ENT=TTYS,	FIL=%TSKSC
HP IMAGE 1000 (92063A)	DBOPN(IVA,RA,RA,I,I),	BP,	OV=8,SS,	SZ=13,	ENT=DMOPN,	FIL=%BAIMG
	DBINF(I,RA,RVA),	BP,	OV=8,SS,	SZ=13,	ENT=DMINF,	FIL=%BAIMG
	DBFND(IVA,RA,RA,RA),	BP,	OV=8,SS,	SZ=13,	ENT=DMFND,	FIL=%BAIMG
	DBGET(IVA,RA,I,RA,RA,RVA,RVA,RVA,RVA,RVA,RVA,RVA,RVA,RVA,RVA), VL,	BP,	OV=8,SS,	SZ=13,	ENT=DMGET,	FIL=%BAIMG
	DBUPD(IV,RA,RA,RA,RA,RA,RA,RA,RA,RA,RA,RA,RA,RA,RA), VL,	BP,	OV=8,SS,	SZ=13,	ENT=DMUPD,	FIL=%BAIMG
	DBPUT(IV,RA,RA,RA,RA,RA,RA,RA,RA,RA,RA,RA,RA,RA,RA), VL,	BP,	OV=8,SS,	SZ=13,	ENT=DMPUT,	FIL=%BAIMG
	DBDEL(IV,RA),	BP,	OV=8,SS,	SZ=13,	ENT=DMDEL,	FIL=%BAIMG
	DBCLS(IV,I),	BP,	OV=8,SS,	SZ=13,	ENT=DMCLS,	FIL=%BAIMG
	DBLCK(IV,I),	BP,	OV=8,SS,	SZ=13,	ENT=DMLCK,	FIL=%BAIMG
	DBUNL(IV),	BP,	OV=8,SS,	SZ=13,	ENT=DMUNL,	FIL=%BAIMG

92069A IMAGE/1000 is documented in manual part number 92069-90001 and its subroutines are in file %BAIMX)

Decimal String Arithmetic	SADD(RA,RVA,IV),	BP,	OV=9,		ENT=D.ADD
	SSUB(RA,RVA,IV),	BP,	OV=9,		ENT=D.SUB
	SMPY(RA,RVA,IV),	BP,	OV=9,		ENT=D.MPY
	SDIV(RA,RVA,IV,IV),	BP,	OV=9,		ENT=D.DIV
	SFDIT(RA,RVA,IV),	BP,	OV=9,		ENT=D.EDT
String Subroutines	DEB$(RVA),		OV=10,		ENT=DEB$
	BLK$(I,RVA),		OV=10,		ENT=BLK$

Variable Length Parameter List

Minimum Number of Pages required for Overlay

Where SS is required as a parameter in the Overlay definition for RTE-III and RTE-IV systems only.

Figure 14-2. RTETG Commands for Library Subroutines (Continued)

to the appropriate subsystem manual for complete information regarding the subroutines. The commands contain extra spaces for readability. All commands must be entered as one record, terminated with a carriage return.

The first command specifies a Branch Table named BTBL, a Mnemonic Table named MTBL, a Transfer File named TRFL, and an A to identify the set of Overlay Directories.

The BASIC Interpreter does not check the Branch and Mnemonic Tables before it parses a line to isolate reserved words. A user function (not a subroutine) included in the Branch and Mnemonic Table that has the letter combination AND or OR in its function name will not be recognized, e.g., SANDR(I), CORE(I,I). These examples would be interpreted as S AND R(I), C OR E(I,I). Therefore, any functions which have these letter combinations in their name should be changed.

14-7. RUNNING THE TRANSFER FILE

The transfer file is produced as a type 3 source file which is named in the first RTETG command. After you have run RTETG, you use the File Manager TR command to execute the transfer file and load the overlays. There must be one ID segment available for each overlay. The loader list device is the same list device specified in the :RUN, RTETG command, Section 14-2. To run the transfer file:

*RUN,FMGR	*Schedule FMGR.*
:TR,*trf*	*Type the TRANSFER command and the name of*
:EXIT	*the transfer file. After FMGR prompts for*
$END FMGR	*another command, give the EXIT command.*

When you have loaded the BASIC Interpreter as described in Appendix D, BASIC will be ready for execution.

14-8. ERROR MESSAGES

Table 14-1 contains a summary of the error messages you may encounter while running RTETG. Each error is prefixed by * ERROR *.

14-9. REPLACING A SUBROUTINE

If a subroutine you have entered in the Branch and Mnemonic Tables does not work properly, you may need to replace the subroutine or regenerate the tables. This can occur if a specification in the Branch and Mnemonic Tables is erroneous or if there are programming errors in the subroutine.

If there is an error in the tables you must generate them again. To do so, you must first purge all files created by the Table Generator. For example:

:PU,*mnemonic table filename*
:PU,*branch table filename*
:PU,*transfer filename*
:PU,%Bx00
    .   *Purge all overlay relocatable file names. % Bxnn,*
    .   *where x is the ID letter, nn is the overlay number.*
:PU,%Bx*nn*

Table 14-1. RTETG Error Messages

MESSAGE	EXPLANATION AND ACTION
BAD ENTRY POINT NAME	A name contains illegal characters. See the RTE-IV Assembler Reference Manual for naming rules.
BAD FILE NAME	A name contains illegal characters. See Section 7 for naming rules.
BAD SECURITY CODE	RTETG cannot access the command file. Create it without a security code or use the correct one.
CANNOT READ COMMAND FILE	If the command file is from a type 0 file, make sure that it is set up for reading.
CARTRIDGE LOCKED	Make sure the disc is not being packed before running RTETG.
CARTRIDGE NOT FOUND	Cartridge specified in first command line is not mounted in system.
COMMAND FILE NOT FOUND	Check to make sure the command file name is spelled correctly.
DIRECTORY FULL	Directory track is full. Purge some files and pack the disc or re-initialize the disc specifying one more directory track.
DISC DOWN	Possible disc malfunction. Try to turn disc back on or check out disc problem.
DUPLICATE FILE NAME	One of the files to be created is named the same as an existing file. Use a different name for the file or specify another cartridge.
ENTRY POINT NAME TOO LARGE	The entry point name exceeds 5 characters.
EOF OR SOF ERROR	Trying to read past the end-of-file mark. If using a logical unit for input make sure that you are at the beginning of the data.
FILE OPEN	The command file is open and RTETG cannot access it. Use the File Manager DL command to determine which program has it open. Decide whether to wait until later or to abort the program which is using it if the operator has abandoned the run without terminating the program.
ILLEGAL FORMAT	Syntax error in RTETG command. Check for missing commas, bad characters, misspelled keywords.
ILLEGAL PARAMETER SPECIFICATION	There is an error in one of the parameter specifications supplied within parenthesis following the subroutine name.
MUST BE NUMERIC LIST DEVICE	The list device specified in the run string must be a logical unit and not a file.
MUST SPECIFY A COMMAND FILE OR LU	RTETG is not interactive.
NAME TOO LARGE	The file name exceeds 6 characters or the subroutine name exceeds 6 characters.

Table 14-1. RTETG Error Messages (Continued)

MEANING	EXPLANATION AND ACTION
NOT ENOUGH PARAMETERS	You have not supplied all of the required parameters with the RTETG command.
NOT ENOUGH ROOM ON CARTRIDGE	Cartridge specified is full. Purge and pack the cartridge or specify another one.
OVERLAY OUT OF NUMERIC ORDER	Overlay numbers must be specified in ascending order.
TABLE OVERFLOW	You have exeeded the limit of 300 subroutine specifications.
TOO MANY PARAMETERS	You have supplied too many parameters with the RTETG command.

Next you must remove the overlay programs by using the Loader.

RTE-II/III

:RU,LOADR , , , 4
PNAME?%Bx00
·
·   or
·
:RU,LOADR , , , 4
PNAME?%Bx*nn*

RTE-IV

:RU,LOADR , , , PU
·
·   *Remove all absolute overlay programs*
·
:RU,LOADR , , , PU

For RTE-IVB in the session environment, instructions for purging a loaded program are given in the RTE-IVB Terminal User's Reference Manual (92068-90002) under the heading of RELOCATING LOADER. The load default of the Table Generator is to place the overlay as a real-time permanent program.

After purging all files and programs simply correct the RTETG input file and run the Table Generator again.

If the problem is in the subroutine itself, the operation is considerably simpler. First, correct the programming error in the subroutine and purge the old overlay absolute programs from the RTE system with the Loader as shown above. Then run the transfer file to reload all the overlays.

:TR,*transfer filename*

Now you can again run BASIC and use your new subroutines.

This section contains all the Real-Time BASIC calls used to communicate with the HP 2313/91000 Data Acquisition Subsystem. Information about generating a system containing the instrument (subsystem) subroutines is provided at the end of the section.

15-1.   MEASUREMENT OF ANALOG INPUT

The HP 2313 Subsystem is capable of measuring single-ended high-level inputs or differential high- or low-level inputs, with 12-bit resolution. The system can be equipped with a plug-in Programmable Pacer for timing measurements with 50 nanosecond precision. High-level input is ±10.24 volts full scale and low-level input is programmable in eight ranges from ±10 millivolts to ±800 millivolts full scale. Measurement ranges on the various low-level channels are preassigned in a table in memory so no distinction has to be made between high- and low-level channels when programming specific measurements.

The HP 91000 Plug-In 20 KHz Analog-to-Digital Interface Subsystem is capable of measuring 16 high-level single-ended or 8 high-level differential inputs. Each plug-in card constitutes a subsystem and can be programmed with the appropriate commands described in this section.

15-2.   ANALOG OUTPUT

The HP 2313 Subsystem can also be used for analog outputs. A dual 12-bit digital-analog converter card, providing two analog outputs, can be installed in any of the working card spaces provided in the basic or expanded measurement subsystem. Analog output speeds can be timed precisely by the pacer option.

15-3.   HP 2313 SUBSYSTEM SUBROUTINES

A series of subroutines are used to perform various 2313 operations. These subroutines are called in the same way as other Real-Time BASIC subroutines:

CALL *name(parameters as required)*

Each call is listed in alphabetical order by name.

15-4.   AIRDV (Random Scan)

The AIRDV subroutine reads analog input in a random manner.

Format

CALL AIRDV (*numb,achan,volt,error*)

Parameters

numb   number of channels to be read. If *numb* is negative, the readings are paced by the system pacer (see PACER call).

achan   array whose contents are channel numbers and whose positional relationship is the same as the voltage in *volt*. If you designate a channel number in *achan,* the corresponding voltage appears in the corresponding position in the *volt* array.

volt   first location of an array where the voltages to be read are placed.

error   variable set to one of the following values upon return from the routine:

                  0 = No error.

                  1 = Overload has occurred. The voltage available at the sensor multiplied by the gain you specified exceeds the voltage range of the analog-to-digital converter.

                  2 = Pace error. The subsystem was not ready when a pace pulse occurred. Normally this is caused by too rapid a pace rate or failing to turn off the pacer after a paced operation.

Example

Assume the channel number array A has been initialized to the channel numbers. The values in the V array correspond on a one-to-one basis to the channel numbers in the A array.

```
10 DIM V(75),A(75)              Reads 75 voltages into the V array.
20 LET N=75
30 CALL AIRDV(N,A(1),V(1),E)
```

15-5. AISQV (Sequential Scan)

The AISQV routine reads analog input sequentially.

Format

CALL AISQV (*numb,schan,volt,error*)

Parameters

numb number of channels to be read. This, in conjunction with the starting address in *schan* defines the starting place and number of channels to be addressed. If *numb* is negative, *numb* readings are taken on the channel and are paced by the system pacer (see PACER call).

schan starting channel number of the sequential scan (the first reading is taken on this channel). If *schan* is negative, *numb* readings are taken on channel *schan*.

volt first location of an array where the voltages to be read are placed.

error variable set to one of the following values upon return from the routine:

0 = No error.

1 = Overload has occurred. The voltage available at the sensor multiplied by the gain you specify exceeds the voltage range of the analog-to-digital converter.

2 = Pace Error. The subsystem was not ready when a pace pulse occurred. Normally this is caused by too rapid a pace rate or failing to turn off the pacer after a paced operation.

Example

```
100 LET N = 100
110 LET C = 50
120 DIM V(10,10)
130 CALL AISQV(N,C,V(1,1),E)
```

Read 100 voltages into the V array beginning with the 50th channel.

15-6. AOV (Digital to Analog Conversion)

The AOV routine converts digital information to analog voltage output.

Format

CALL AOV(*numb,achan,volt,error*)

Parameters

numb   the number of channels to be output. If *numb* is negative, the analog output is paced by the system pacer.

achan   array containing channel numbers corresponding to voltages in *volt* array.

volt   array of voltages to be output to the channels defined by *achan*.

error   variable set to one of the following values upon return from the routine:

      0 = No error.

      1 = Overload has occurred. The gain you specified is outside the range of the digital-to-analog converter.

      2 = Pace error. The subsystem was not ready when a pace pulse occurred. Normally this is caused by too rapid a pace rate or failing to turn off the pacer after a paced operation.

Example

```
300  CALL  AOV(25,C(1),V(1),E)     Convert data from channels in the C array to
                                    analog voltages and store in V array.
```

15-7.   NORM

The NORM routine normalizes the subsystem, resets it to a home or known state.

Format

CALL NORM [(*unit*)]

Parameter

unit   HP 2313/91000 unit number. This number is defined as part of the Instru-
                          ment Table. See the Section XVIII. If the unit number is going to be
                          supplied, the NORM subroutine must be modified in the Branch and
                          Mnemonic Tables to accommodate the parameter. If not, it is assumed to be
                          one.

Unless specifically excluded, all numbers are floating point numbers. Either the actual number or a reference to the number may be given.

Example

```
200 LET Z = 2          Set subsystem unit equal to 2.
210 CALL NORM(Z)       Normalize the HP 2313 Subsystem.
```

All DACs (Digital to Analog Converters) are set to 0. The HP 2313 pacer and the LAD (Last Address Detector) is turned off. The gain settings on the LLMPX channels are not changed.

15-8.   PACER

The PACER routine sets the pace rate of the HP 2313 system. The HP 91000 Subsystem pacer is not controlled by this call.

Format

CALL PACER(*rate,mult,start*[,*unit*])

Parameters

rate   basic pacer rate in microseconds. Basic rate must be between 0 and 255 inclusive.

mult   rate multiplier. This parameter specifies one of eight ranges (0 - 7) which is a power of ten multiplier applied to the basic pace rate.

For example:

rate   = 25 (25 microseconds)

mult   =   3 results in a pace period of 25 milliseconds.

start   external and internal start/stop control for the pacer.

Value	Change Period or Start/Stop	External Start/Stop
0	Immediately	Disable
1	Next pace pulse	Disable
2	Immediately	Enable
3	Next pace pulse	Enable

unit   HP 2313 unit number. This number is defined as part of the instrument table. See Paragraph 16-10. If *unit* is to be supplied, the PACER subroutine must be modified in the Branch and Mnemonic Tables.

If *start* is zero, a 10-millisecond delay is inserted by the PACER routine. This allows time for the analog input or output statement to be executed so that no pace error occurs.

Example

```
100 REM NORMALIZE SYSTEM        Perform a paced reading on a single channel
110 CALL NORM                   with a period of 1 millisecond.
200 REM RATE = 1*10↑3 MICROSECONDS
210 CALL PACER(1,3,0)
300 REM TAKE 20 READINGS FROM CHANNEL 5
310 CALL AISQV(20,-5,V(1),E)
400 REM TURN OFF PACER
410 CALL PACER(0,0,0)
```

15-9.   RGAIN

The RGAIN routine tells you what the gain setting is on a particular channel.

Format

<div>

CALL RGAIN(*chan,gain*)

Parameters

chan   the channel number which must be greater than or equal to 1.

gain   location which upon return contains the discrete number 1, 12.5, 25, 50, 100, 125, 250, 500, or 1000 representing the setting of the gain amplifier on that particular channel.

</div>

The gain for any particular channel is initially set during system configuration. This call allows you to determine the present gain setting of any channel.

Example

```
400 CALL RGAIN(55,X)            Determine present gain on channel 55 and store
                                it in X.
```

15-10.   SGAIN

The SGAIN routine sets the gain for all channels in a group.

Format

CALL SGAIN(*chan,gain*)

Parameters

chan   LLMPX channel number which must be greater than or equal to 1.

gain   value of the gain to which you wish a specified channel to be set. It should
                be one of the discrete values: 12.5, 25, 50, 100, 125, 250, 500, or 1000. These
                are the only values available on the multiplexers. If the gain specified is
                not one of these, it will be set to the next lowest gain in relation to its
                absolute value. If it is < 12.5, 12.5 is used. For example, 5 or − 5 will be set
                to 12.5, − 999 will be set to 500, and 1001 will be set to 1000.

In multiple channel groups all of the channels in the group have the same gain, so changing the gain on a single channel sets all of the channels in the group to the new gain. This does not apply if the groups have been specified as one channel per group during configuration.

Example

```
500  CALL  SGAIN(CH,12.5)
```
                              The channel CH is set to 12.5.

15-11. SUBSYSTEM ERRORS

All error messages generated by the previous subroutines take the following form:

        ERROR *name-numb* IN LINE *xx*

where *name* is the module name, *numb* is the error type, and *xx* is the line in which the error occurs.

The following is a list of HP 2313/91000 Subsystem errors (*name-numb* values):

HLMPX/LLMPX

ADC-1 Driver timeout
ADC-2 Parameter value outside defined range
ADC-3 Attempt to set gain on HLMPX channel or HP 91000 Subsystem.

DAC

AOV-1 Driver timeout
AOV-2 Addressed channel undefined.

15-8

15-12. TABLE PREPARATION

In order to use the HP 2313/91000 Subsystem subroutines you must provide information about the subsystem which is incorporated in the Instrument Table. The method for doing this is described in Section XVIII.

You must also add the names of the subroutines you want to use to the Branch and Mnemonic Tables. Generation of these tables is described in Section XIV. A list of the RTETG commands required to add these subroutines is given there.

15-13. SUBSYSTEM CONCEPT

Figure 15-1 illustrates the concept of the HP 2313 Subsystem configuration. A subsystem is defined as the equipment occupying a computer I/O slot. Note that in the figure, the HP 2313 is one subsystem (including all three boxes) having one I/O slot number, and the HP 91000 DAS Card is another subsystem occupying an I/O slot number. The Real-Time BASIC software treats the HP 91000 exactly like an HP 2313 Subsystem. The only difference is that there are no low-level multiplexers or Digital-to-Analog cards in the HP 91000.

1. The HP 2313 HLMPX and LLMPX input cards are divided as follows:

   High-Level Multiplexer   16-Differential inputs
                                 32-Single ended inputs

   Low-Level Multiplexer   16-Differential

2. All HP 2313 HLMPX and LLMPX cards are numbered sequentially for programming purposes.

3. The HP 91000 DAS has either 8 HLMPX differential inputs or 16 HLMPX single-ended inputs.
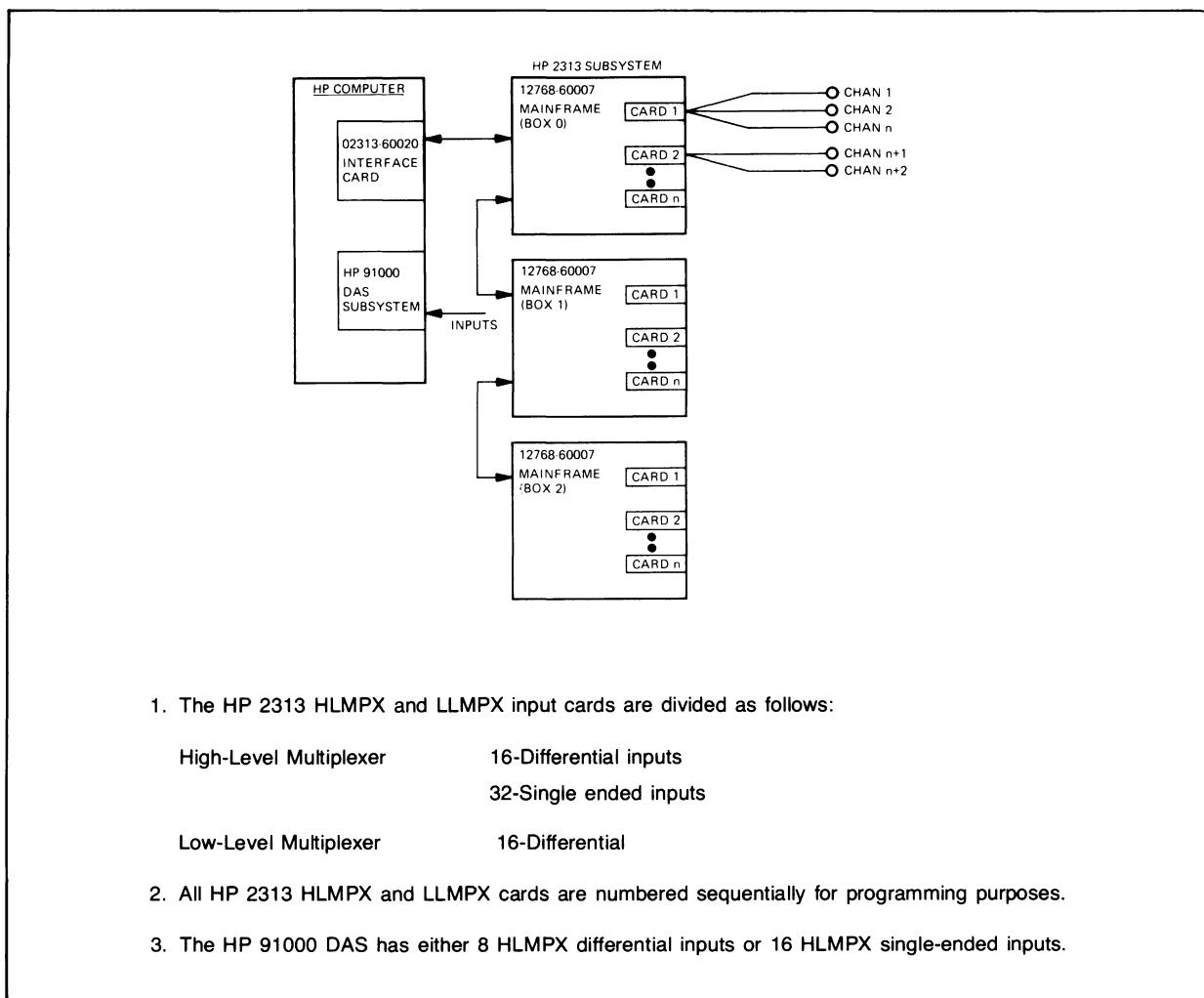
Figure 15-1. HP 2313 Subsystem Configuration

15-14. CARD CONFIGURATION

The configuration of the various types of cards in an HP 2313 Subsystem must follow the conventions given below. If expansion is anticipated, configuration can include blank slots to accommodate future cards. The blank slots will then have channel numbers assigned to them, but cannot be accessed until the hardware is added.

Box 0 Configuration

slot 0 - System Pacer (optional)

slot 1 - ADC

slot 2 - Sample and Hold Amplifier

slot 3 - ⎫
            ⎬ Working Cards
slot 11 - ⎭

The working cards must be in the following order:

a. All high-level multiplexers with single-ended inputs.

b. All high-level multiplexers with differential inputs.

c. All low-level multiplexers.

d. All dual DACs.

If the subsystem contains a Last Address Detector (LAD) card, it must follow the last multiplexer or DAC card.

15-15. CHANNEL NUMBERING

As a result of physically placing all high-level multiplexer (HLMPX) and low-level multiplexer (LLMPX) cards in the required order, and then specifying how many of each type there are, all of the multiplexer channels will be assigned a number starting with 1 for the first specified channel and ending with n for the last. It is your responsibility to record the division line between channel numbers as shown in the following example.

CHANNEL NUMBERS	TYPE
1	
2	HLMPX
:	single-ended
32	
33	HLMPX
34	differential
:	
48	
49	
50	
:	Blank
64	
65	LLMPX
66	Gain = 25
:	
72	
73	LLMPX
74	Gain = 100
75	1 channel
76	groups
77	
78	
79	
80	

15-16. SETTING GAIN

Gain must be specified for low-level multiplexer channels. At system generation time a gain is assigned to a group of channels which is used until changed by the SGAIN command. Low-level channels can be configured into groups of 1 to N channels where N is the total number of low-level channels in a subsystem.

All of the channels in a group have the same gain, so if the gain of one channel in a group is changed by the SGAIN command, all channels in that group are changed to the specified gain. If programming the gain of each channel independently is required, then the groups would consist of only 1 channel. It might be noted that the execution of the SGAIN command does not have any direct effect on the hardware, but simply changes the group-gain entry in the configuration table. When a reading is taken, this table is read to determine the gain with which to set the corresponding multiplexer card, and the appropriate conversion factor to compute the actual terminal voltage. Therefore, the SGAIN command only has to be issued to change the table and does not have to be issued following a system normalize command or before each reading.

The HP 6940 Multiprogrammer is the master control unit for bidirectional data transfers (i.e., output data distribution/input data multiplexing). The HP 6940 can be used in a single-unit system employing from one to fifteen plug-in input/output cards, or in a multi-unit system consisting of up to 8 HP 6940 master units each with up to fifteen HP 6941A extender units. Each extender unit can also accommodate up to fifteen input/output cards. The digital I/O capabilities include direct or isolated digital input and interrupting event sense input, solid-state digital output, relay register output, stepping motor control, voltage and current DAC, programmable timers, and pulse counters.

16-1.   HP 6940 SUBSYSTEM SUBROUTINES

A series of subroutines are used to operate the subsystem. They vary depending on the characteristics of the particular device. These subroutines are called in the same way as other Real-Time BASIC subroutines:

> CALL *name(parameters as required)*

Each call is listed in alphabetical order by name.

16-2.   DAC

The DAC subroutine converts digital information to analog.

Format

CALL DAC(*chan,value*)

Parameters

chan   number of the analog output channel.

value   either the voltage or current (depending on the hardware) to be output by the DAC. (Current is in milliamperes.)

DAC causes the desired analog voltage or current value (based on *value*) to be output on the channel defined by *chan*.

Example

100 CALL DAC(1,10)   *Outputs 10 volts on channel 1.*

16-3.   MPNRM

The MPNRM routine clears the event sense mode and erases the channel/bit to trap number correspondence. This call should be issued before any SENSE calls to insure there are no residual definitions from previous programs. This command also negates any previous SENSE calls.

Format

```
CALL MPNRM
```

16-4.   RDBIT

The RDBIT subroutine checks the state of a specified bit on a channel.

Format

CALL RDBIT(*chan,nbit,bit*)

Parameters

chan   number of the channel being read.

nbit   the bit position starting from the right with bit 0.

bit   state of *nbit*, either 0 or 1.

RDBIT reads from the specified HP 6940 channel the state of a certain bit. Bit positions having a 1 or 0 may represent relay contact opening or closing. The precise state is a function of the particular hardware and its interface with the computer.

Digital input can be performed in two modes, either with or without wait for the input card flag. The flag is set by making the channel number negative in the RDBIT call.

Examples

100 CALL RDBIT(-1,3,0)   *Performs digital input with wait from channel 1 bit 3.*

200 CALL RDBIT(1,2,B)   *Reads a bit from channel 1, bit position 2 into location B without wait.*

16-5.   RDWRD (Read Channel)

The RDWRD subroutine reads the contents of a channel into a word.

Format

CALL RDWRD(*chan,word*)

Parameters

chan   number of the channel being read.

word   location into which the results of the information from the channel will be placed.

The RDWRD routine reads the HP 6940 interface board on the specified channel and places the twelve input bits into the twelve low-order positions of the location defined by *word*. The four high-order positions are zeroed. It is possible to read any digital signal up to 12 bits wide. The signals may represent any type of device in any combinations.

Digital input can be performed in two modes, either with or without wait for the input card flag. The flag is set by making the channel number negative in the RDWRD call.

Examples

```
100  CALL  RDWRD(1,W)
```
        Reads a complete word of information from channel 1 and places the information in location W.

```
200  CALL  RDWRD(-1,W)
```
        Digital input is performed with the wait flag set.

16-6.   SENSE

The SENSE (Event Sense) routine senses a change in the bit pattern.

Format

CALL SENSE(*chan,nbit,bit,trapn*)

Parameters

chan   number of the channel being sensed.

nbit   bit position starting from the right with bit 0.

bit   state of *nbit*, either 0 or 1.

trapn   trap number to which the sensing is directed.

The SENSE subroutine allows you to ask the hardware to constantly monitor for the presence of a specified condition. Specifically, the two conditions that can be sensed are:

- a given bit position becoming 1,
- a given bit position becoming 0.

As soon as one of the conditions is met, an interrupt is set and the associated TRAP is initiated. The condition being met in one direction, and thereby causing the interrupt, does not imply any interrupt in the opposite direction. If the SENSE is upon a given bit becoming 0 and that bit changes from the 0 condition, no action results in BASIC.

The SENSE command is designed for the HP 69434 Event Sense Card only. If the SENSE command is addressed to any other card, the following message is given:

ERROR A6940-2 IN LINE *nnnn*

where *nnnn* is the line number.

The HP 69434A card is operated in the "not equal" mode (jumper W3 in position D on the card). In this mode, an event is detected when the external data is "not equal" to the reference bit (parameter *bit* in the SENSE call).

Example

Time the duration of the "on" (1) condition of bit 4 on channel 2.

```
300 TRAP 5 GOSUB 500        Define TRAP 5.
310 TRAP 6 GOSUB 600        Define TRAP 6.
320 SENSE(2, 4, 1, 5)       Sense when bit 4 of channel 2 changes from 0 to
     •                      1.
     •                      Execute TRAP 5.
500 CALL TIME(X)            Record time when bit changes. Then sense when
510 SENSE(2, 4, 0, 6)       turns "off" (1 to 0) and execute TRAP 6.
520 RETURN
600 CALL TIME(Y)            Record time when turns off.
610 LET Z = Y - X           Compute interval between on and off.
620 PRINT "TIME INTERVAL = ";Z   Print interval.
630 SENSE(2,4,1,5)          Sense "on" again.
640 RETURN
```

16-7.   WRBIT

The WRBIT routine writes a bit onto a channel.

Format

CALL WRBIT(*chan,nbit,bit*)

Parameters

chan   number of the channel.

nbit   bit position starting from the right with bit 0.

bit   state of *nbit*, either a 0 or 1.

WRBIT writes a single bit on the specified channel of the HP 6940. The bit can turn a light on or off, close or open a relay, or perform any other discrete function.

Digital output can be performed in two modes, either with or without wait for the output card flag. The flag is set by making the channel number negative in the WRBIT call.

Examples

```
100 CALL WRBIT(1,3,0)       Write a 0 bit onto channel 1 position 3 without
                            wait.
200 CALL WRBIT(-1,3,0)      Perform digital output with wait.
```

16-5

16-8. WRWRD (Write Channel)

The WRWRD routine writes the contents of a word onto a channel.

Format

CALL WRWRD(*chan,word*)

Parameters

chan   number of the channel.

word   location from which the information is written.

WRWRD writes on the channel specified, the contents of the word stated by the parameter *word*. This provides a 12-bit parallel digital output which corresponds to the 12 least significant bits (right-most bits) of *word*. Depending upon the associated hardware, lights will be operated, relays closed, etc.

Digital output can be performed with or without wait for the output card flag. The flag is set by making the channel number negative in the WRWRD call.

Examples

```
100 CALL WRWRD(1,W)
```
Writes the lower 12 bits of information from the location W onto channel 1 without wait.

```
200 CALL WRWRD(-1,W)
```
Performs the same output with wait.

16-9. SUBSYSTEM ERRORS

All error messages generated by the previous subroutines take the following form:

ERROR *name-numb* IN LINE *xx*

where *name* is the module name, *numb* is the error type, and *xx* is the line in which the error occurs.

The following is a list of HP 6940 errors.

INPUT/OUTPUT CALLS

A6940-1   Driver timeout.

A6940-2   Parameter value outside defined range.

DAC CALL

DAC-1   Driver timeout.

DAC-2   Parameter value outside defined range.

16-10. TABLE PREPARATION

In order to use the HP 6940 subroutines you must provide information about the subsystem which is incorporated in the Instrument Table. The method for doing this is described in Section XVIII.

You must also add the names of the subroutines you want to use to the Branch and Mnemonic Tables. Generation of these tables is described in Section XIV. A list of the RTETG commands required to add the HP 6940 subroutines is given there.

16-11. CARD CONFIGURATION

The configuration of the various types of cards in an HP 6940 Subsystem must follow the conventions given below. If expansion is anticipated, configuration can include blank slots to accommodate future cards. The blank slots then have channel numbers assigned to them, but cannot be accessed until the hardware is added.

Figure 16-1 illustrates the concept of the HP 6940 Subsystem configuration. The cards must be installed in the following order:

a. All HP 69434 Event Sense cards operating in the "not equal" mode. A maximum of 15 cards is allowed for each 6940. Event Sense cards are not allowed in the HP 6941.

b. All digital output or input/output cards.

HP 69435 - Pulse counter
HP 69330 - Relay output
HP 69331 - TTL output
HP 69333 - Relay output with readback
HP 69335 - Stepping motor
HP 69600 - Timer

c. All digital input only cards.

HP 69430 - Isolated digital input
HP 69431 - Digital input

d. All analog output cards.

HP 69321 - Voltage DAC
HP 69370 - Current DAC
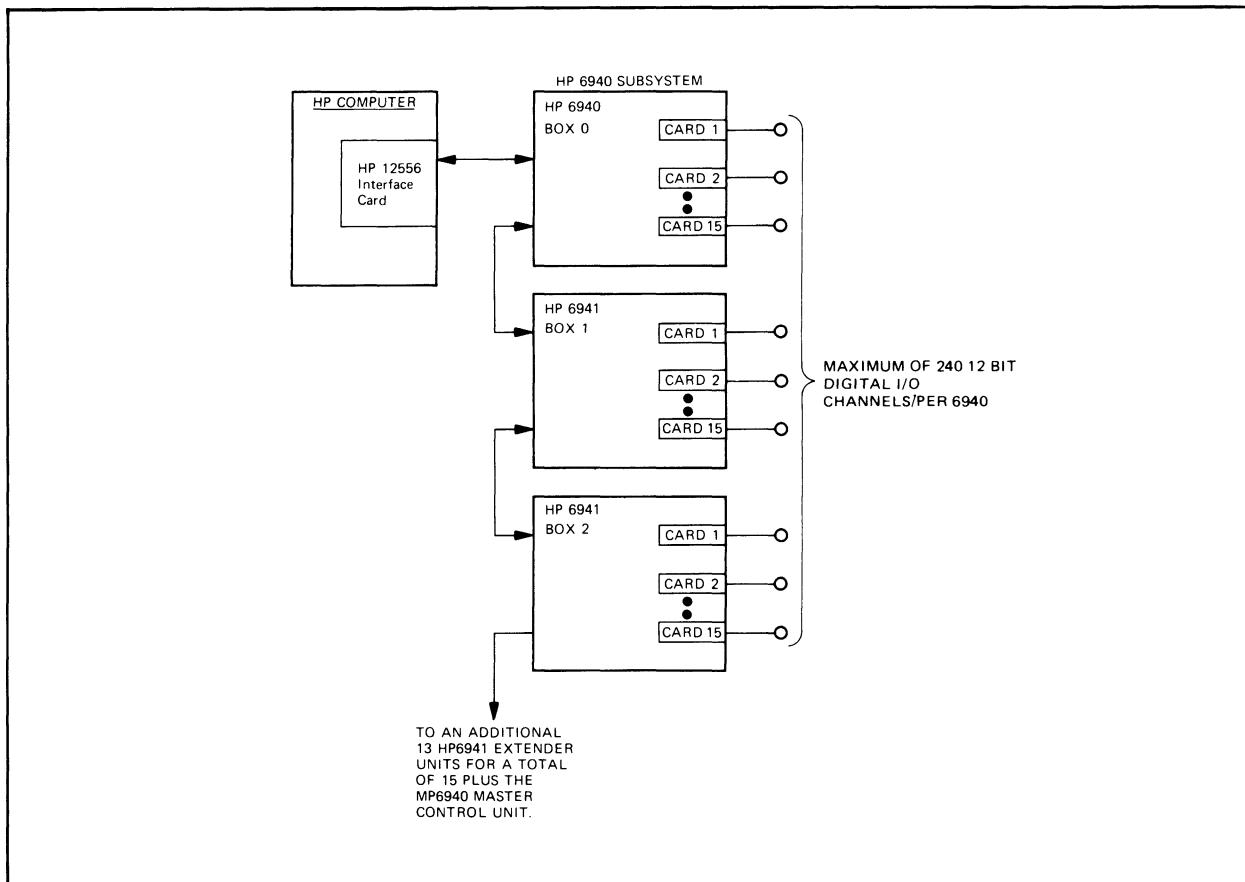
Figure 16-1. HP 6940 Subsystem Configuration

16-12. EXPANSION

The system may be expanded by adding additional HP 6940 Subsystems or by adding HP 6941 Extenders to existing subsystems. The order of cards described in paragraph 16-11 applies to each HP 6940 added. The card order also applies to the slots beginning with slot 401 in the 6940 through slot 414 in a HP 6941, if one is added. Remember that Event Sense cards are not allowed in the HP 6941.

16-13. CHANNEL NUMBERING

The channels are numbered sequentially starting after the last HP 2313 channel number. If there is no HP 2313, the first HP 6940 channel is 1.

Figures 16-2 and 16-3 illustrate two channel numbering schemes and the effect of expanding the system in one of the two ways described above. Assume that the system shown in the examples has 96 HP 2313 channels.

CHANNEL NUMBERS	6940/6941 SLOT NUMBERS	CARD TYPE
97	400	Event Sense
98	401	Event Sense
99	402	Digital I/O
.	.	.
.	.	.
108	411	Digital I/O
109	412	Voltage DAC
110	413	Current DAC
111	414	Current DAC
112	400	Event Sense
113	401	Digital I/O
.	.	.
.	.	.
126	414	Digital I/O

1st 6940 (rows 97–111)
2nd 6940 (rows 112–126)

Figure 16-2. Channel Numbers for Additional 6940

CHANNEL NUMBERS	6940/6941 SLOT NUMBERS	CARD TYPE
97	400	Event Sense
98	401	Event Sense
99	402	Event Sense
100	403	Blank cards for
101	404	future Event Sense
102	405	expansion
103	406	Digital I/O
.	.	.
.	.	.
111	413	Digital I/O
112	414	Blank (future Digital I/O)
113	400	Voltage DAC
114	401	Current DAC

1st 6940 (rows 97–112)
6941 (rows 113–114)

Figure 16-3. Channel Numbers for Addition of a 6941 Extender

The plotter draws precise lines on a paper surface that is a maximum size of 15 by 10 inches. The lines are drawn by a pen on the plotter which moves from a desired point to some other desired point in either a pen-up (no line) or pen-down position. To facilitate the drawing of these lines, several routines have been written for your use with BASIC programs. The routines can be separated into two categories:

- those that draw characters (alphabetic characters and special symbols), and
- those that draw lines.

Figure 17-1 at the end of this section contains a program which uses most of these routines, and figure 17-2 contains a program which uses some of the special routines (e.g., scale, axis, lines).

The HP 7210 Plotter routines interface with the RTE Driver, DVR10. There are two versions of this driver. The Complete Plotter Driver DVR10 supports both character and line drawing routines. The Minimum Plotter Driver DVR10 supports only line drawing routines. Refer to the HP 92060 Software Numbering Catalog for the module names and part numbers of these drivers.

17-1. AXIS

The AXIS routine plots one axis (horizontal or vertical) of a graph with a specified axis label, a specified length, and a specified value at each inch marker.

Format

CALL AXIS(x,y,"label",length,angle,min val, inc val)

Parameters

x	horizontal coordinate.
y	vertical coordinate.
label	axis label (Enclose in quotes as a literal string.)
length	length of axis in inches.
angle	angle of axis in degrees. 0° is horizontal, angle increases in counter-clockwise direction.
min val	minimum value of axis (may be calculated with SCALE routine).
inc val	incremental value (may be calculated with SCALE routine).

Parameter *length* can be positive to place label counterclockwise to axis (as in y axis) and negative to place clockwise (as in x axis).

SCALE must be called before AXIS if points on the graph are scaled by SCALE. AXIS calls the SYMB routine to plot the labels 0.14 inches high.

Numbers are printed with .07 inch character height. Since X and Y always refer to the origin of the axis, leave at least .5 inch for axis label and numbers.

Example

```
100 CALL AXIS(0,0,"PSI",10,90,1,1)
```

*Plots the Y axis with the label "PSI"
on the counterclockwise side, 10 inches
long at 90 degrees.*

17-2.   FACT

The FACT (factor) routine sets the ratio between the horizontal and vertical axis.

Format

CALL FACT(*x,y*)

Parameters

x   horizontal axis

y   vertical axis

FACT and SFACT are interrelated routines. Once SFACT is used to establish the graph limits (paper size), FACT does not need to be called unless you want to change the scale relationship.

Initially the horizontal and vertical axis in FACT are automatically set to 1 (e.g. the magnitude of 2 equals two inches in both directions). If you wish to have something plotted at half size, and SFACT has been set to 15 X 10, set FACT (.3333,.5) in the program. *x* and *y* are multiplied by 1000 plotter increments per inch when the new scaling factor is established.

The following equations give the actual translation from the X,Y in the plot call to the value actually put out by the plotter.

$$X_{out} = 1000.0 * X_{in} * X_{fact}$$

$$Y_{out} = 1000.0 * Y_{in} * Y_{fact}$$

X_{in} is X in PLOT call. X_{fact} is X in FACT call (set to 1 originally). Y_{in} is Y in PLOT call. Y_{fact} is Y in FACT call (set to 1 originally).

17-3.   LINES

The LINES routine plots a line and/or symbols through successive data points in arrays previously scaled by the SCALE routine.

Format

CALL LINES(*x (1),y (1),numb,scale,cntrl,symb*)

Parameters

x	array scaled for the abscissa.
y	array scaled for the ordinate.
numb	number of points to be plotted.
scale	integer that specifies the point to be scaled.
	1 = every point.
	2 = every second point.
	n = every nth point.
cntrl	control value:
	0 = line plot only.
	1 = symbol at every point with line.
	−1 = symbol at every point.
	2 = line and symbol at every second point.
	−2 = symbol at every second point.
	n = line and symbol at every nth point.
	−n = symbol at every nth point.
symb	number of centered symbol to be plotted (see SYMB routine).

Since the LINES routine requires the adjusted minimum and delta values produced by the SCALE routine, SCALE must be called before LINE for each graph.

Before calling this routine, the pen should be moved to the point 0,0 on the graph, and the origin should be set at that point with LLEFT.

17-4.   LLEFT

The LLEFT routine lifts the pen and moves it to the lower-left corner.

Format

CALL LLEFT

When LLEFT is called, the internal x and y references are set to zero. This provides a defined reference for starting a plot. The pen is left in the up position at the completion of this call. This means that a call to PLOT must be made to put the pen down.

17-5. NUMB

The NUMB routine plots a number, with or without the decimal point, at a specified height, location, and angle.

Format

CALL NUMB(*x,y,height,numb,angle,digits*)

Parameters

x   horizontal coordinate of lower left corner of number.

y   vertical coordinate.

height   height of number in inches.

numb   number to be plotted.

angle   angle at which the number is to be plotted.

digits   number of digits.

   0 = print the decimal point of an integer.

   −1 = suppress the decimal point.

   n = number of digits to print to the right of the decimal point (maximum of seven).

Example

```
100  CALL  NUMB(5.32,8.79,0.1,0.0,-1)
110  CALL  NUMB(6.3,8.79,0.1,J,0.0,-1)
120  CALL  NUMB(7.16,8.79,0.1,K,0.0,-1)
```

Plot three numbers .1 inches high, with decimal point suppressed, at 8.79 inches above 0,0 and 5.32,6.3, and 7.16 inches to the right of 0,0.

17-6. PLOT

The PLOT routine moves the pen from an origin to a destination.

Format

CALL PLOT (*x,y,pram*)

Parameters

x   horizontal coordinate in inches.

y   vertical coordinate in inches.

pram   constant or variable name set equal to one of the following:

   −2 = move with pen down; consider the point where the pen stops (x,y) as the new origin.

   −3 = move with the pen up; consider the point where pen stops as new origin.

   +2 = move with the pen down; origin unchanged.

   +3 = move with the pen up; origin unchanged.

If the LLEFT routine is called first to establish the lower-left corner, the pen is left up. The PLOT routine must be called as follows to put the pen down:

CALL PLOT(0,0,2)

The parameters of the routine determine whether the origin is the original reference of the lower-left corner of the paper or the last known position from the previous plot.

Example

Plot a rectangle 8.5" by 11" starting at the origin (assuming the pen starts at the origin and that scaling has been set at 15 by 10 inches by calling LLEFT and SFACT(15,10)).

```
100 CALL PLOT(0,0,2)        Sets the pen down.
110 CALL PLOT(11.,0.,2)     Moves the pen from X,Y =0,0 to X,Y = 11,0.
120 CALL PLOT(11.,8.5,2)    Moves the pen from X,Y = 11,0 to X,Y = 11,8.5.
130 CALL PLOT(0.,8.5,2)     Moves the pen from X,Y = 11,8.5 to X,Y = 0,8.5.
140 CALL PLOT(0.,0.,2)      Moves the pen from X,Y = 0,8.5 to the origin.
```

17-7.   PLTLU

The PLTLU routine defines the logical unit number of the plotter for all the plotter calls.

Format

CALL PLTLU(*lu*)

Parameter

lu   logical unit number of the plotter.

The PLTLU routine must be the first routine called to establish the logical unit number.

17-8.   SCALE

The SCALE routine scales an array of numbers to fit a specified size graph; the values generated are used by the LINES and AXIS routines.

Format

CALL SCALE(*array(1),length,num pts,scale*)

Parameters

array   an array of values.
length   length of axis in inches.
num pts   number of points to be plotted.
scale   integer specifying the points to be scaled:

1 = every point.

2 = every second point.

n = every *n*th point.

Separate calls are required for X and Y axis.

The adjusted minimum value is a number less than or equal to the minimum data value. The adjusted delta value is the result of subtracting the minimum data value from the maximum data value, divided by the length of the axis and adjusted to provide one-inch increments that will cover the data. The adjusted scale values are used by the LINES and AXIS routines.

The adjusted values are stored following the array. The minimum value of Y is stored in Y($np*k+1$) where np is the number of points to be plotted; the delta value is stored in Y($np*k+2$). Therefore, the array must be dimensioned ($k+2$) locations larger than ($np*k$), which is the number of locations necessary for data points. Normally, k equals 1, so an array Z of ten data points would be dimensioned as Z(12).

Example

```
10  DIM  X(52),Y(52)              Dimension X and Y arrays.
      •
      •
100  CALL  SCALE(X(1),6.5,50,1)    Scale every point in a 50 point array, fitting X
110  CALL  SCALE(Y(1),10.0,50,1)   values on a 6.5 inch X axis and Y values on a 10
                                   inch Y axis.
```

17-9. SFACT

The SFACT routine sets or adjusts the plotter for the particular size paper being used.

Format

CALL SFACT(*width,height*)

Parameters

width   width scale for horizontal movements.

height   height scale for vertical movements.

SFACT and FACT are interrelated routines. Once SFACT is used to establish the graph limits (paper size) FACT does not need to be called unless you wish to change the scale size. Initially the width and height parameters in SFACT are automatically set to 10 inches by 10 inches.

SFACT should be used as follows:

a. Place the paper on the plotter bed.

b. Manually adjust the pen and carriage travel to the lower left and then upper right corners of the paper (image area if desired).

c. Call SFACT with the parameters set to the paper size that was set up in step b.

After the plotter is adjusted and SFACT has been called with the proper dimensions, the horizontal and vertical dimensions in other calls are set for a one-to-one relationship.

Example

```
10 CALL PLTLU(13)
20 CALL LLEFT
25 CALL SFACT(15,10)
30 CALL PLOT(0,0,2)
40 CALL PLOT(5,5,2)
```

The pen moves five inches horizontally and five inches vertically for a paper size of 15 by 10 inches.

17-10. SYMB

The SYMB routine is used when characters are to be plotted.

Format

CALL SYMB(*x,y,size,*"*char*" *string variable* *,theta,pram*)
integer

Parameters

x	horizontal starting coordinate.
y	vertical starting coordinate.
size	character height in inches.
"*char*"	actual characters to be plotted enclosed in quotes (string literal).
string variable	string variable containing characters to be printed.
integer	any single integer 0 through 25 (without quotes) representing a special character.
theta	number of degrees counterclockwise that the line of plotting is to be rotated.
pram	constant or variable name set equal to one of the following:
	1 = plot the characters between the quotes or in the string variable.
	−1 = draw the special character and leave the pen up.
	−2 = draw the special character and leave the pen down.

The SYMB routine is used to write characters on the paper if *pram* = +1. If *pram* = −1 or −2, the special character represented by the ASCII number is drawn. If the pen is left down, moving the pen to the place where the next character is to be drawn causes the special characters to be connected like points along a graph. If the pen is left up, the points are drawn without intervening connecting lines. When drawing graphs, it is recommended that symbols that are centered be used. The coordinates specified by you refer to the lower left corner of an imaginary box enclosing the character to be drawn. If you wish to continue to call PLOT and have the set of ASCII characters be contiguous, place the value 9999 as the *x* and *y* coordinates. Be aware that the *x* and *y* coordinates from one subroutine are not transmitted to the other subroutines. You can use WHERE to supply missing information when you transfer from the PLOT to the SYMB routine, etc.

The following are the ASCII characters which may be enclosed within quotes:

A through Z and 0 through 9.

@ [ ] ＼ ↑ ← ! " # $ % & ' ( ) * , - . / 0 : ; < = > ? "space"

These symbols are centered and have their ASCII reference number immediately above the symbol:

0	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

17-11. URITE

The URITE routine lifts the pen and moves it to the upper right corner of the paper to facilitate unloading the paper at the end of a plot. This routine does not affect any of the coordinates.

Format

```
CALL URITE
```

17-12. WHERE

The WHERE routine indicates the current position of the plotter pen.

Format
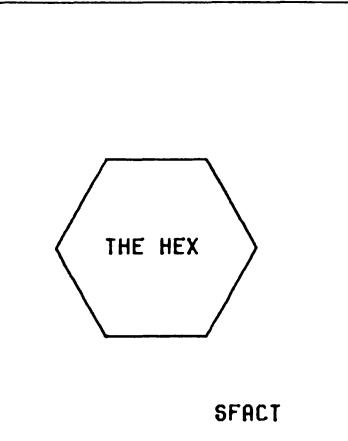
```
CALL WHERE(x,y)

Parameters

x       variable in which the horizontal coordinate is returned.

y       variable in which the vertical coordinate is returned.
```

17-13. TABLE PREPARATION

In order to use the plotter subroutines, you must add the name of the routines to the Branch and Mnemonic Tables. The procedure for doing this is described in Section XIV. The required RTETG commands are provided there.

```
10   DIM A$(30)
20   LET Z=1
30   LET A=D=X=0
40   PRINT "PLOT DEMO"

50   CALL SFACT(15,10)
60   CALL LLEFT
70     FOR A=1 TO 2
80     CALL PLOT(1,1,3)
90     CALL PLOT(7,5,1,2)
100    CALL PLOT(7,5,9,2)
110    CALL PLOT(1,9,2)
120    CALL PLOT(1,1,2)
130    CALL SYMB(5,1,5,.25,"SFACT",0,1)
140    CALL SFACT(30,20)
150    CALL PLOT(3,3,-3)
160    NEXT A

170  CALL SFACT(0,40)
180  LET X=3
190  LET Y=3
200    FOR B=1 TO 6
210    FOR A=1 TO 15
220      CALL SYMB(X,Y,.25,15,0,-2)
230      LET X=Y=9999
240      NEXT A
250    LET D=D+60
260    NEXT B
270  CALL SYMB(3,3,.5,"THE HEX",0,1)

280  CALL LLEFT
290  CALL SFACT(15,10)
300  LET X=1.75
310  LET Y=8.5
320    FOR C=0 TO 14
330    CALL SYMB(X,Y,.2,C,0,-1)
340    LET X=Y=9999
350    NEXT C

360  LET X=3.1
370  LET Y=7.75
380    FOR C=15 TO 25
390    CALL SYMB(X,Y,.2,C,0,-1)
400    LET X=Y=9999
410    NEXT C

420  LET A$="CENTERED SYMBOLS"
430  CALL SYMB(3.5,8.125,.1,A$,0,1)
440  LET A$="UNCENTERED SYMBOLS"
450  CALL SYMB(3,4,7.5,.1,A$,0,1)

460  CALL SYMB(1.75,7.1,.2,"ABC",0,1)
470  CALL SYMB(9999,9999,.2,"DEF",0,1)
480  CALL SYMB(9999,9999,.2,"GHIJKLMNOPQRSTUVWXYZ",0,1)
490  CALL SYMB(3.25,6.7,.2,"0123456789",0,1)
500  CALL SYMB(1.85,6.25,.2,"@[]\↑←1",0,1)
510  CALL SYMB(9999,9999,.2,"#$X&'()*,./!<>=?",0,1)

520  CALL URITE
530  PRINT "END OF PLOT"
540  END
```

To demonstrate the effect of "SFACT" on plot size, this section draws a box twice. The first time "SFACT" is set to (15,10). The second time it is set to (30,20) causing the second box drawn to be scaled one-half of the original larger box. Note that the different scale factor applies also to the size of the lettering.

The scale factor is again reset and the "underline" character is used to draw a hexagon. This demonstrates the ability to draw characters in any orientation. Note that the initial setting of the pen is given to place the pen at the proper starting point and that all subsequent characters are drawn from the last-used position by employing coordinates of (999,999) thereby producing a solid line.

As the position of the pen is unknown, the pen is brought to the known position of the lower-left-hand corner. The magnitude is made equal to inches by (15,10) and then the pen is sent to the starting position in the first "SYMB" execution. Thereafter, the last pen position is accepted for the next character. The special characters 0 through 14 are first plotted.

The same logic is used to plot characters 15 through 25.

The two lines are labeled.

All remaining characters are plotted by first positioning the pen at the beginning of each new line desired.

The pen is moved out of the way to the upper-right corner.

Figure 17-1. Plotter Control Sample Program #1

```
10    REM
20    REM        7210 PLOTTER EXAMPLE
30    REM
40    LET N=36
50    CALL PLTLU(18)
60    CALL LLEFT
70    CALL SFACT(15,10)
80    CALL PLOT(.5,0,-3)
90    REM
100   REM        PLOT AXES FOR X AND Y
110   REM
120   CALL AXIS(0,5,"RADIANS",-14,0,0,1)
130   CALL AXIS(0,1,"VOLTAGE",8,90,-.4,.1)
140   LET F=0
150   LET F=F+1
160   REM
170   REM        PLOT CURVES
180   REM
190   DIM X[142],Y[142]
200   IF F=2 CALL PLOT(0,1,-3)
210   FOR I=0 TO 14 STEP .1
220   LET X[I*10+1]=I
230   IF F=1 LET Y[I*10+1]=SIN(I)*4
240   IF F=2 LET Y[I*10+1]=COS(I)*4
250   NEXT I
260   REM
270   REM        SCALE ARRAY
280   REM
290   CALL SCALE(X[1],14,140,1)
300   CALL SCALE(Y[1],10,140,1)
310   REM
320   REM        PLOT THE ARRAY
330   REM
340   CALL LINES(X[1],Y[1],140,1,1,F)
350   IF F<2 GOTO 150
400   REM
410   REM        PRINT PLOT NUMBER
420   REM
430   CALL SYMB(12.2,8.5,.15,"PLOT #",0,1)
440   CALL NUMB(9999,9999,.15,N,0,-1)
500   REM
510   REM        PRINT LEGEND
520   REM
530   CALL SYMB(12.2,1.5,.2,"LEGEND",0,1)
540   CALL PLOT(12.2,1.4,3)
550   CALL PLOT(13.3,1.4,2)
560   CALL SYMB(12.2,1.1,.15,1,0,-1)
570   CALL SYMB(9999,9999,.15," SINE WAVE",0,1)
580   CALL SYMB(12.2,.6,.15,2,0,-1)
590   CALL SYMB(9999,9999,.15," COSINE WAVE",0,1)
600   CALL WRITE
610   STOP
9999  END
```

Figure 17-2. Plotter Control Sample Program #2
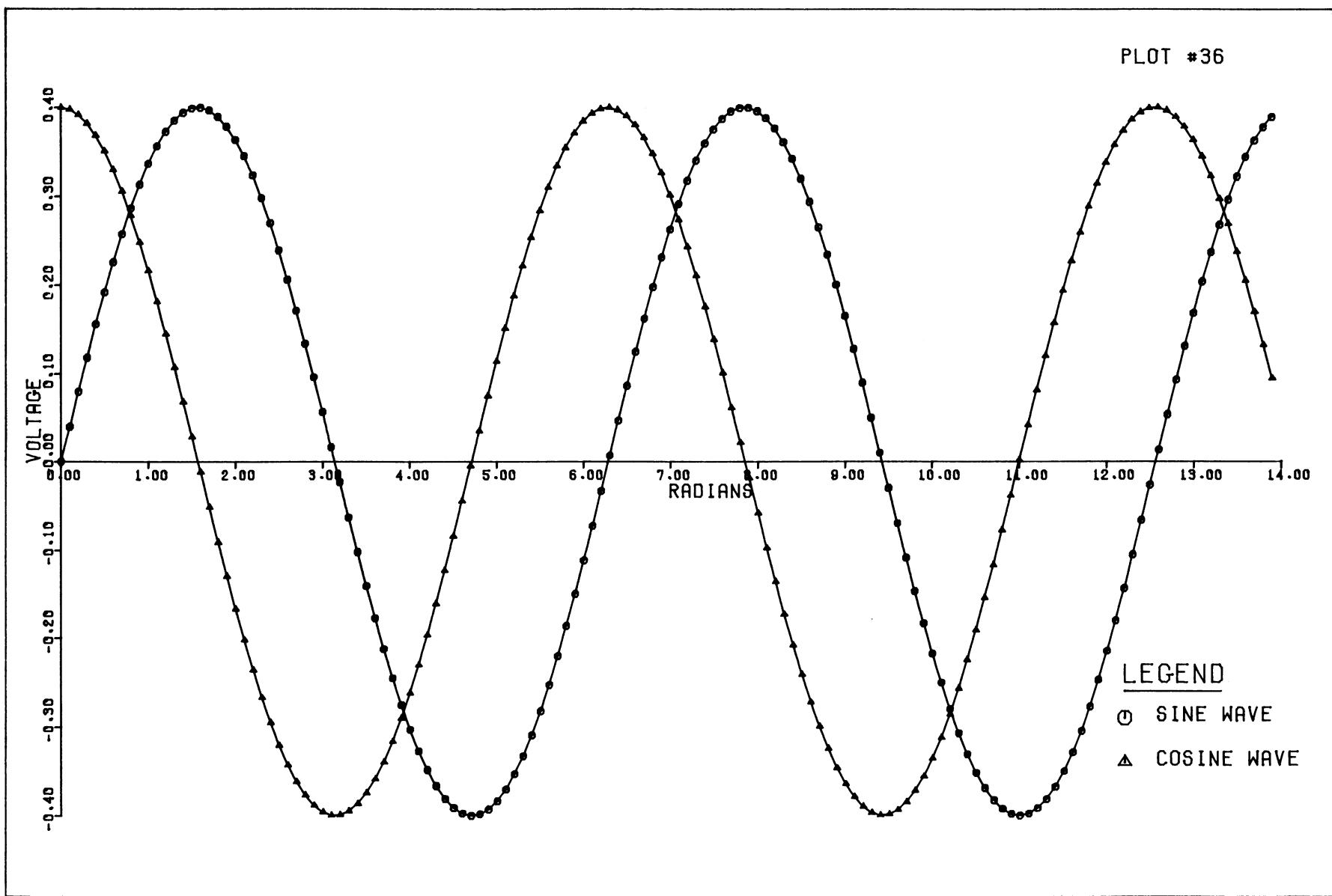
Figure 17-3. Plotter Control Sample Program #2 (Plot)

In order for the HP 2313 and HP 6940 to be configured into the RTE system, it is necessary to generate an Instrument Table tape which is loaded during RTE system generation. This tape contains all of the information concerning the 2313 and 6940 subsystems which is required by the RTE Operating System. You generate the Instrument Table tape with the Instrument Table Generator program. (See Preface for part number.)

Before generating the Instrument Table tape, be sure that the HP 2313 and HP 6940 cards are in the proper order as described in Sections XV and XVI.

18-1. OPERATING INSTRUCTIONS

The Table Generator is an absolute program that uses the HP System Input/Output (SIO) drivers and is loaded into memory with the Basic Binary Loader (BBL). To execute the program do the following:

a. Apply power to all equipment. Set the system teleprinter to LINE.

b. Load the Table Generator tape and configured SIO Drivers into memory using the BBL.

c. Go to the starting address of the program, location 2, and set the switch register as follows:

   • If the teleprinter punch is to be used, set bit 15 to 1. The following information halts will occur:

      HLT07 (turn on punch)
      HLT70 (turn punch off).

   • If the teleprinter punch is not used, set bit 15 to 0 (clear the switch register).

d. Push RUN. The Table Generator begins with the HP 2313 Configuration Phase. See below.

After each question is printed by the system, respond with the required answer followed by carriage return and linefeed. The Table Generator prompts with a hyphen. For clarity, it has been omitted from the following text. Responses are shaded and are only examples; actual responses should be given as appropriate to the particular system you are generating.

If an error is made and discovered before RETURN is pressed, type the RUBOUT key then RETURN. Otherwise restart at step c above.

18-2. HP 2313/91000 CONFIGURATION PHASE

This part of the generation produces information about the HP 2313/91000 Subsystems. For specific information on the configuration of those subsystems see Section XV.

After being loaded into memory and execution started, prints:

   2313 AND 6940 TABLE GENERATOR
   TURN ON PUNCH

If the punch is already on, or after it is turned on, some leader will be punched.

# OF 2313'S? `1`	The Table Generator requests the number of HP 2313 Subsystems and 91000 Subsystems. The number is equal to the total of both types of subsystems. If there are no subsystems of either type, type a 0 (zero) and the Table Generator skips to the HP 6940 Configuration Phase. If there is more than one subsystem, subroutines NORM and PACER must have the unit number parameter added in the Branch and Mnemonic Tables. (See Section XIV.)
SUBSYSTEM #01 CONFIGURATION	One is the unit number used in the subsystem calls referencing this subsystem.
LU? `7`	The Table Generator requests the logical unit number of the subsystem.
#HL - SE? `32`	Specify the number of high-level single-ended input channels. Each HP 2313 card has 32 channels, so multiply the number of HLMPX single-ended cards times 32 for the total number of channels. If the subsystem is a HP 91000, answer either 16 or 0. If 16, the next question must be answered 0 for the HP 91000.
#HL - DIF? `16`	Specify the number of high-level differential input channels. Each 2313 card has 16 channels, so multiply the number of HLMPX differential cards times 16 for the total number of channels. If the subsystem is a HP 91000, answer either 8 or 0.

#LL,GAIN?
`8,12.5`
`8,100`
`0,0`
	Specify the number of low-level channels in a group and the gain of that group. For the HP 91000 Subsystem, answer 0,0. The allowable gains are:
	
	12.5, 25, 50, 100, 125, 250, 500, 1000.
	
	The sample responses specify that there are 8 channels in one group, all with a gain of 12.5, and 8 channels in a second group, all with a gain of 100. A negative gain specifies that each single channel group has the specified gain (e.g., 8, −12.5 indicates that 8 single channel groups each have a 12.5 gain.) The prompt character (hyphen) is repeated until a 0,0 response is given.

#DAC'S? `2`	Specify the number of Digital-to-Analog Converters. For the HP 91000, answer 0. Each HP 2313 card has 2 DAC'S. If the system has at least one DAC, the AOV subroutine must be entered in the Branch and Mnemonic Tables.

If more than one 2313 Subsystem is being used, the prompts above (beginning with SUBSYSTEM #*n* CONFIGURATION) will be repeated for each subsystem until you respond with a zero to the first prompt.

18-3. HP 6940 CONFIGURATION PHASE

Refer to Section XVI for HP 6940 configuration information. If the answer to the number of 2313's question was zero or the last HP 2313 subsystem has been described, the Table Generator begins this part of the generation.

# OF 6940'S? 1	The Table Generator requests the number of subsystems. If there are none, enter a zero and the Table Generator will skip to the Constant Configuration Phase. Otherwise it prints the next prompt.
# OF CHANNELS IN 2313'S? 64	Enter the total number of multiplexer channels in all the HP 2313 Subsystems you configured in the preceding part of the table generation.
SUBSYSTEM #01 CONFIGURATION	
I/O SLOT? 12	Enter the computer I/O slot for this HP 6940 Subsystem.
LU? 8	Enter the logical unit number of this subsystem.
#EVENT? 2	Enter the number of event sense cards.
# DIG INPUT FOR PRESET CNTR W/INT? 0	Enter zero. This question does not apply to Multi-User Real-Time BASIC.
# DIGITAL I/O? 9	Enter the number of digital input/output cards.
# DIGITAL INPUT? 0	Enter the number of digital input-only cards.
# VOLTAGE DACS? 2	Enter the number of voltage digital-to-analog converter cards.
# CURRENT DACS? 2	Enter the number of current digital-to-analog converter cards.
# TIMERS? 0	Enter zero. This question does not apply to Multi-User Real-Time BASIC.
# COUNTERS? 0	Enter zero. Does not apply to Real-Time BASIC.

If more than one 6940 Subsystem is being used, the prompts above (beginning with the message SUBSYSTEM #n CONFIGURATION) will be repeated for each subsystem. After the last subsystem has been described, the Table Generator prints:

ENTER INSTRUMENT CONFIGURATION CONSTANTS /E	Enter symbols and constants representing any additional instrument subsystems. Terminate the list with "/E".

At this point, the Instrument Table tape will be punched out. Then the Table Generator prints:

TABLE GENERATION COMPLETE

Tear off the tape, label it, and save it for loading during RTE system generation.

18-4. LOADING THE TAPE

You must load the Instrument Table tape into the RTE system during system generation along with the BASIC Subroutine Libraries that contain the BASIC software. This method of loading allows the Loader to automatically search and load the table. Under RTE-III and RTE-IV, the Instrument Table should be made into a type 30 module (put into SSGA).

18-5. ERROR MESSAGES

Table 18-1 contains a summary of the error messages you may encounter while running the Table Generator.

Table 18-1. Error Messages

MESSAGE	MEANING	CORRECTIVE ACTION
ILLEGAL CONTROL DIRECTIVE	Incorrect format.	Type again in correct format.
ILLEGAL RESPONSE	Incorrect input.	Type again with correct response.

Formatted output gives you explicit and exact control over the format of your program output:

- Numbers can be printed in three different representations: integer, fixed point, and floating point.

- The exact position of plus and minus signs can be specified.

- String values can be printed in specified fields and literal strings and blanks can be inserted wherever needed.

- Full control over carriage returns and line feeds is possible.

- Arbitrarily long lines can be printed without the carriage returns and line feeds normally provided by PRINT statements.

19-1. SPECIFYING FORMATTED OUTPUT

There are two key elements in formatted output; the list of items to be printed (the using list) and the format in which these items are to be printed (the format string).

Example:

```
10 PRINT USING format string; using list
10 PRINT #20 USING "4D";A
```

19-2. USING LIST

The using list specifies what items are to be printed. A using list is made up of the same types of items that can be included in the PRINT statements. The items are expressions, numeric constants, string variables, and print functions.

Examples of using lists:

```
10 PRINT USING 100;    A,SIN(X),F$
40 PRINT USING 150;    (L3+N)/R(I),TAB(N4).A4
```

Print using list

19-3. FORMAT STRING

As the using list specifies which items are to be printed, the format string specifies how the items are to be printed. A format string can be made up of a carriage control character to control carriage returns and/or line feeds; format specifications to indicate the format for each item in the using list; and delimiters to separate the carriage control character from the format specifications and the format specifications from each other.

A format string included in the PRINT statement can be in a literal string, in a string variable, or in a special statement called the IMAGE statement. However, if a literal string is used, for the format

specification, it cannot contain a string literal as one of the format specifications. This must be done either through a string variable or an IMAGE statement.

Examples of format strings:

```
10 PRINT USING        "-,DD.D,3X,2(20A)";P4,A$,B$
30 PRINT USING        R$                ;A(I),F2,TAB(3)
50 IMAGE              #,2X,3(4DE,2A),""
60 IMAGE              2X,6A/2X,6A//
```

<p align="center">format string</p>

A statement number referencing an IMAGE statement may be used instead of the format string.

Example:

```
            statement
            reference        using list
10 PRINT USING 20;    A,SIN(X),F$
20 IMAGE #,3DX/,D.DDD,3AX
```

19-4. USING FORMATTED OUTPUT

The first step in producing formatted output is to decide what each item in the using list should look like when it is printed. There are two basic types of data that can be formatted, numbers and strings.

19-5.   NUMBER REPRESENTATION

A number can be printed as any of the following:

> Integer (with no decimal point)

> Fixed Point Number (with decimal point in fixed position)

> Floating Point Number (either an integer or fixed point number followed by an exponent)

The format characters used to specify numeric formatting are D, E, S, period (.), and X, where:

> D   specifies that one digit should be printed

> E   specifies that an exponent should follow the number

> S   specifies printing the sign of the number

> (.)   specifies printing a decimal point and defines its position

> X   specifies printing a blank.

An integer repeat specification may be included immediately preceeding the X or D format characters to be repeated.

Integer format examples:

```
DDD
4D
2DDD      equivalent
2D2D

2DX3D
SDDD
S4D
DX3DS
```

Integer output examples:

Format Specification	Value	Format of Output
4D	1234	1234
S4D	1234	+1234
4DS	1234	1234+
5D	1234	1234
4D	1234.8	1235
DXDDD	1234	1 234
S10D	1234	+1234
DSDDD	1234	1+234
5D	-1234	-1234
4D	1234.2	1234

Fixed-point format examples:

```
DDD.DDD
DDD.3D
3D.3D      equivalent
3D.DDD

S3D.3D
DXDXDX.DDXD
XD6X4D.8D
DDSDD.3D
```

Fixed-point output examples:

Format Specification	Value	Format of Output
3D.4D	465.465	465.4650
4D.2D	465.465	465.47
4D.3D	-465.465	-465.465
SDD2D.D	465.465	+465.5
S2D.4D	.465	+0.4650
S.4D	.465	+.4650
D.4D	-.465	-.4650
2D.4D	-.465	-0.4650

Floating point format examples:

```
SD.5DE
DDD.DDDXEX
SD.8DXE
S6DE
S6D.E
S6D.XE
S6D.DDDE
```

Floating point output examples:

Format Specification	Value	Format of Output
SDXE	4.82716×10^{21}	+5 E+21
DDDD.DDE	SAME	4827.16E+18
S5DX.X5DEX	SAME	+48 . 27159E +20
SD.5DE	SAME	+4.82716E+21
S.10DE3X	SAME	+.4827159382E +22

19-6.   CARRIAGE CONTROL

One of the following optional carriage control characters may appear as the first non-blank character of a format string:

+   suppressing the line feed

−   suppressing the carriage return

suppressing both the carriage return and linefeed

If supplied, the carriage control character must be followed by a comma and at least one slash, format specification, or group. The specified action occurs at the completion of the PRINT USING statement. If the carriage control character is not supplied, the default action is a carriage return and a linefeed at the completion of the PRINT USING statement.

Examples:

```
10 PRINT USING "#,DDD,2X,AA";A,A$
20 IMAGE -,2A,3X,4D
```

19-7.   LITERAL STRING

A literal string (any combination of characters not including a quote mark and enclosed by quote marks) can be included as a format specification in an IMAGE statement and is printed as it appears.

Example:

```
400 IMAGE "TOTAL = ",X,S3D.2D
```

19-8.   DELIMITERS

A delimiter serves to set off the format specifications and can be either a comma or a slash. A comma serves no purpose other than to delimit format specifications. A slash can delimit format specifications and, in addition, generates a carriage return, line feed sequence.

Example:

```
500 IMAGE -,3A/3D,3D
```

19-9. TAB FUNCTION

The print function TAB can be used in the PRINT USING statement (and can also be used in all other PRINT statements).

TAB(X)   tabs out to column X, where X may be either a literal or a variable, before printing the next item.

The TAB function is implemented by inserting blanks to move the current print position. This means that devices that overwrite on spaces will replace characters with blanks if those characters are tabbed over.

19-10. STRING REPRESENTATION

A string can only be printed as a sequence of characters. The format character which is used to specify string output is an "A". In addition, the format character "X" can be used to denote blanks in a format specification. An "X" or "A" format character may be preceded by an unsigned integer to indicate repetition.

String format examples:
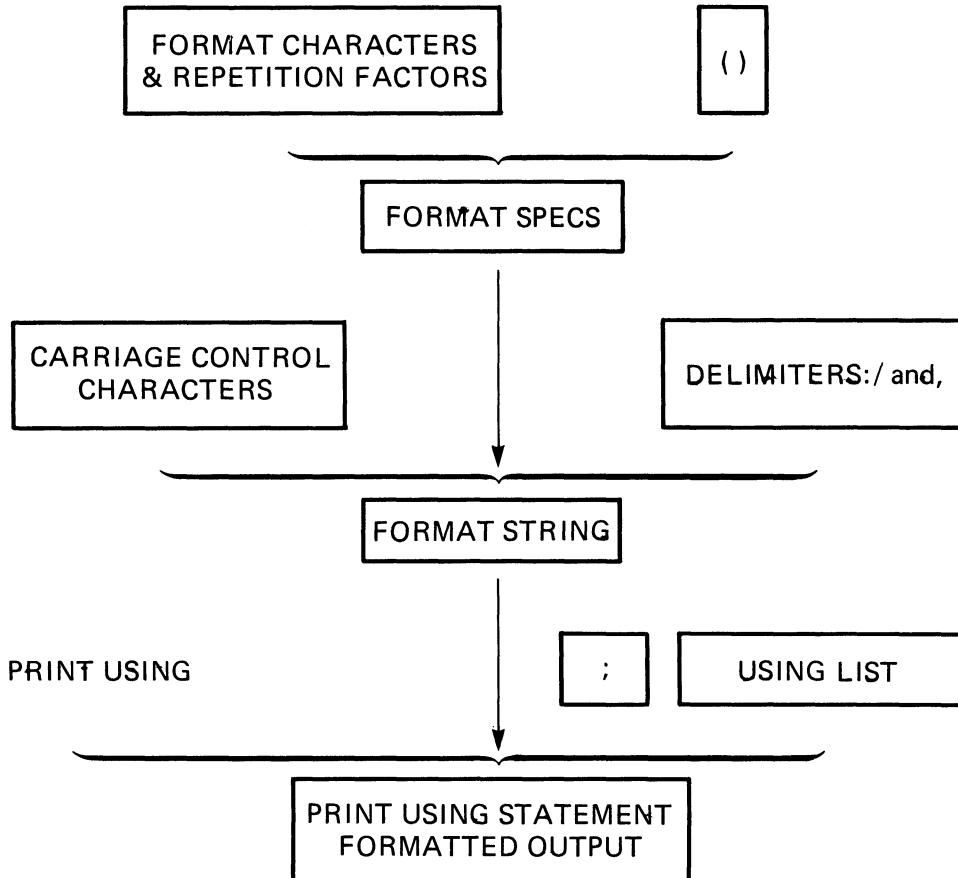
```
AAAA
4A          Equivalent
2A2A

4X          special case (all blanks, so no variable required)
AXAXAXA     alternate characters and blanks
2X20A
```

String output examples:

Format Specification	Contents of String Variable	Format of Output
6A	ABCDEF	ABCDEF
5A	ABCDEF	ABCDE
8A	ABCDEF	ABCDEF
2X6A	ABCDEF	ABCDEF
AXAXAXAXAXA	ABCDEF	A B C D E F

Figure 19-1 illustrates how the various format and control characters are used to make up a PRINT USING statement.

Figure 19-1. PRINT USING Statement Structure

USING LIST   USING LIST   USING LIST

Examples:

```
PRINT USING     "SDDD.DDDE"          ;Z1
PRINT USING     "2X,3(3D.3D,2X),14A" ;Z1,Z2,Z3,F0$
                Format String        Using List
```

The following program example prints out a numeric conversion chart. It prints a value in inches (from 1 to 25) and then prints the equivalent value in meters, centimeters, and micrometers. Integer, fixed-point, and floating point formats are used.

Program example:

```
100 REM PRINT THE HEADING
110 PRINT USING 120
120 IMAGE "INCHES",10X,"METERS",4X,"CENTIMETERS",4X,"MICROMETERS",//
130 FOR I=1 TO 25
140 REM PRINT THE VALUE OF INCHES
150 REM USING CARRIAGE CONTROL TO SUPPRESS CARRIAGE RETURN AND LINE FEED
160 PRINT USING 170;I
170 IMAGE #,2X,2D,13X
180 REM CONVERT TO OTHER UNITS
190 M=I*.0254
200 C=M*100
210 M1=M*1.E+06
220 REM PRINT VALUES IN FIXED POINT AND FLOATING POINT
230 PRINT USING ".DDD,8X,DD.2D,8X,D.3DE";M,C,M1
240 NEXT I
250 END
```

```
>RUN
```

INCHES	METERS	CENTIMETERS	MICROMETERS
1	.025	2.54	2.540E+04
2	.051	5.08	5.080E+04
3	.076	7.62	7.620E+04
4	.102	10.16	1.016E+05
5	.127	12.70	1.270E+05
6	.152	15.24	1.524E+05
7	.178	17.78	1.778E+05
8	.203	20.32	2.032E+05
9	.229	22.86	2.286E+05
10	.254	25.40	2.540E+05
11	.279	27.94	2.794E+05
12	.305	30.48	3.048E+05
13	.330	33.02	3.302E+05
14	.356	35.56	3.556E+05
15	.381	38.10	3.810E+05
16	.406	40.64	4.064E+05
17	.432	43.18	4.318E+05
18	.457	45.72	4.572E+05
19	.483	48.26	4.826E+05
20	.508	50.80	5.080E+05
21	.533	53.34	5.334E+05
22	.559	55.88	5.588E+05
23	.584	58.42	5.842E+05
24	.610	60.96	6.096E+05
25	.635	63.50	6.350E+05

```
>BASIC READY
```

19-11.   REPORT GENERATION

This program is a sample report generator. It first requests a school number from the terminal, then reads and prints out information about the school's teachers from a file. Note that a carriage control character is used to advantage (statement 100), slashes (/) are used (statement 200) string and fixed-point fields are used (statement 210), and an error occurs in the output for the eighth teacher (number too large for field; therefore, it is printed in E format on a separate line).

Program:

```
10 REM THIS PROGRAM GENERATES A REPORT ON TEACHERS
50 DIM A$(25),B$(19),C$(19)
60 FILES SCH1,SCH2,SCH3,SCH4,SCH5
100 IMAGE #,"ENTER SCHOOL NUMBER:"
150 IMAGE "TEACHER..",13X,"SUBJECT",13X,"SALARY",4X,"ATTND."
175 IMAGE "-------",13X,"-------",13X,"------",4X,"------"/
200 IMAGE "CENTRAL CITY SCHOOL DISTRICT"/"DAILY REPORT OF ",25A//
210 IMAGE 20A,20A,"$",DDD.DD,DD.DDDD
230 PRINT USING 100
250 INPUT Z
260 READ #Z;A$,N
270 PRINT
500 PRINT USING 200;A$
550 PRINT USING 150
555 PRINT USING 175
557 FOR A1 =1 TO N
560 READ Z;B$,C$,A,B
600 PRINT USING 210;B$,C$,A,TAB(50),B
620 NEXT A1
1000 END
```

```
ENTER SCHOOL NUMBER:?1

CENTRAL CITY SCHOOL DISTRICT
DAILY REPORT OF B. BAKER HIGH SCHOOL
```

TEACHER	SUBJECT	SALARY	ATTND.
-------	-------	------	------
MISS BROOKS	ENGLISH	$450.34	12.5000
MISS CRABTREE	REM. READING	$400.00	64.3200
MISS GRUNDIE	HISTORY	$350.00	1.0010
MRS. HUMPREY	SPELLING	$700.00	99.9900
COLONEL MUSTARD	CRIMINOLOGY	$700.00	21.4500
MISS PEACH	LIFE PREPARATION	$232.00	23.2320
PROF. PLUM	AGRICULTURE	$777.77	65.0050
MISS H. PRYNNE	SOCIAL STUDIES	$100.25	
+5.00500E+02			
MISS SCARLETT	P.E.	$205.10	25.0000
MR. SIR	HOME ROOM	$890.00	99.9000
MR. T. TIM	MUSIC	$ 10.99	0.0500
MR. WEATHERBY	ECONOMICS	$767.99	10.0400

19-12. PRINT USING FORMAT ERRORS

These errors cause termination of the program.

1. Replicator is outside the range $1 \leq n \leq 255$.

2. D, S, E, or . occurs in a string specification.

3. A occurs in a numeric specification.

4. Characters other than A, X, D, S, E, /, or . occur in any specification other than a literal.

5. Adjacent commas.

6. More than two levels of parentheses.

7. No D in a numeric specification.

8. An S in a blank specification.

9. String expression encountered for a numeric format specification.

10. Two or more E's, S's, or .'s in a specification.

11. Replicator not followed by (, D, X, or A.

12. Literal string not set off by delimiters and enclosed in quotes.

13. Parentheses used without a replicator.

14. Referenced statement is not IMAGE.

15. Numeric data encountered by a string format.

16. Leading or trailing commas in group or format string (i.e., null group or missing format specification).

These errors do not terminate the program.

1. String specification too small for the string data causes the string to be truncated from the right.

2. Field too small for number. Causes the number to be printed on the next line using SD.5dE format. Printing resumes on the next line.

STATEMENT SUMMARY

This summary of Real-Time BASIC statements provides the statement names in alphabetical order with a brief description and a reference to the paragraph or paragraphs containing a complete statement description.

STATEMENT	DESCRIPTION	REFERENCE
ASSIGN	Dynamically assigns a file name to a file number and opens the file.	7-4
CALL	Calls for execution of an external subroutine, optionally passing parameters to the subroutine.	6-3
CHAIN	Terminates the current program and calls for execution of the Real-Time BASIC program named in the CHAIN statement. Variables are shared between programs if named in COM statement.	6-2
COM	Declares a common block to contain specified variables used in common by more than one program. Effective when one program calls another with CHAIN.	3-13
DATA	Provides data to be read by READ statements.	3-10 4-12
DEF	Defines a function.	5-2
DIM	Reserves storage for arrays and sets upper bounds on the number of elements.	3-12
	DIM also reserves storage for strings and sets their maximum character length.	4-5
END	Terminates execution of the current program. Last statement in the program must be END.	3-4
FILES	Allocates file numbers to file names or reserves file numbers for later assignment with ASSIGN. FILES is declarative and unlike ASSIGN, is not executed.	7-3
FOR. . .NEXT	Allows repetition of a group of statements between FOR and NEXT. The number of repetitions is determined by the initial and final values of a FOR variable, and by an optional step specification.	3-5
GOTO	Transfers control to a specified statement label.	3-3

STATEMENT	DESCRIPTION	REFERENCE
GOTO. . .OF	Multibranch GOTO transfers control to one of a list of statement labels depending on the value of an integer expression.	3-3
GOSUB	Causes execution of a subroutine beginning at a specified statement label. Following a RETURN statement in the subroutine, control returns to the statement following GOSUB.	6-1
GOSUB. . .OF	Multibranch GOSUB executes one of a list of subroutines depending on the value of an integer expression.	6-1
IF END #. . .THEN	Specifies action to be taken when an end-of-file condition occurs.	7-5
IF. . .THEN	Evaluates a conditional expression and specifies action to be taken if condition is true. The condition is a numeric or string expression. The action may transfer to a statement label or may be a single executable statement.	3-6 4-10
INPUT	Requests user input to one or more variables by printing a ? and accepts string or numeric data from the terminal.	3-11 4-7
LET	Introduces assignment statement that assigns one or more values to a variable or array element. The word LET may be omitted.	3-1 4-6
NEXT	Terminates a loop introduced by a FOR statement. Specifies a variable that must match the FOR variable.	3-5
PAUSE	Stops program execution without terminating the program.	3-14
PRINT	Prints the contents of a list of numeric or string expressions on the list device.	3-7 4-8
PRINT #	Outputs the contents of a list of numeric or string expressions to the specified file.	4-13 7-9, 10
PRINT #*lu*	Outputs contents of a list of numeric or string variables or expressions to the specified *lu* (logical unit number). May be used without FILES statement.	4-13 7-9, 10
READ	Assigns constants and string literals from one or more DATA statements to the variables specified in READ. Treats contents of all DATA statements as a single data list.	3-10 4-9
READ #	Reads one or more items from a file into specified variables.	4-14 7-6, 7, 8
READ #*lu*	Reads one or more items from a *lu* (logical unit) into specified variables.	4-14 7-7, 8

STATEMENT	DESCRIPTION	REFERENCE
REM	Introduces remarks and comments in the program listing.	3-2
RESTORE	Resets the data pointer to the beginning of the program or to the first DATA statement following a specified label.	3-10
RETURN	Returns control from a GOSUB subroutine to the statement following the last GOSUB.	6-1
STOP	Terminates execution of the run.	3-4
TRAP	Associates a trap number with a task.	11-11
WAIT	Causes an executing program to stop for a specified number of milliseconds.	3-15

COMMAND SUMMARY

Each command is listed by name in alphabetical order followed by a brief description and a reference to the paragraph or paragraphs containing a complete description of the command.

COMMAND	DESCRIPTION	REFERENCE
ABORT	Legal only in break period; terminates the suspended program and returns control to the BASIC Interpreter where all commands are legal.	10-4
BACKF	Backspaces past previous file mark.	13-1
*BR,BASIC	Breaks execution of a program.	9-14
BREAK	Specifies breakpoints where execution of program will be interrupted to enter debugging commands.	10-2
BYE	Terminates execution of BASIC Interpreter.	9-12
CALLS	Lists all of the mnemonics in the current Mnemonics Table.	10-8
CREATE	Creates a Real-Time BASIC formatted file with a specified length; optionally on a specified cartridge with a security code.	9-6
CSAVE	Stores a program in semi-compiled format as a disc file.	9-2
DELETE or DEL	Deletes one or a range of more than one statement from current program.	9-5
LIST	Lists the contents of the current program at the terminal or on a specified type 0 file.	9-13
LOAD	Loads all or a portion of a source or semi-compiled program.	9-1

STATEMENT	DESCRIPTION	REFERENCE
LOCK	Locks a peripheral device to your program.	9-11
MERGE	Merges a source program with a program in memory.	9-3
PURGE	Deletes the specified data or program file from the system.	9-7
RENAME	Renames a program or data file.	9-8
REPLACE	Replaces a source program on a disc file.	9-4
RESEQ	Renumbers any group of statements in the current program, optionally from a new first line number with a specified increment. By default, renumbering starts at 10 with increments of 10.	9-9
RESUME or RES	Resumes normal BASIC operation following breakpoint or a call simulation.	10-3
REWIND	Rewinds magnetic tape on specified device.	13-1
RUN	Executes the current program or gets and executes a specified program file.	9-10
SAVE	Saves the current program as a program disc file.	9-2
SET	Legal only in break period, sets any program variable to a constant value.	10-7
SHOW	Legal only in break period; lists the values of the specified variables.	10-6
SIM	Simulates a subroutine call.	10-5
SKIPF	Skips to end of current file.	13-1
TABLES	Specifies names of Branch and Mnemonic Tables.	8-2
TRACE	Traces variables and array values, and the execution of statements and segmented programs.	10-1
UNBREAK	Turns off any or all breakpoints specified with the BREAK command.	10-2
UNLOCK	Unlocks a locked device.	9-11
UNSIM	Turns off simulation facility.	10-5
UNTRACE	Turns off tracing facility.	10-1
WEOF	Writes an EOF mark on magnetic tape.	13-1

SUBROUTINE SUMMARY

Each subroutine is listed in alphabetical order followed by a brief description and a reference to the paragraph containing a complete description of the routine.

SUBROUTINE	DESCRIPTION	REFERENCE
AIRDV	Reads HP 2313 analog input in a random manner.	15-4
AISQV	Reads 2313 analog input sequentially.	15-5
AOV	Converts digital values to analog output. (2313)	15-6
AXIS	Plots an axis of a graph.	17-1
BLK$	Initializes a string to a specified number of blanks and resets the logical length of the string.	E-4
CHRS	Places the ASCII character of a specified decimal value into a string.	6-8 & E-6
DAC	Converts digital value to analog output. (6940)	16-2
DBCLS	Closes IMAGE data base files.	*
DBDEL	Deletes existing data records from IMAGE data sets.	*
DBFND	Locates the beginning of an IMAGE data chain in preparation for access to entries in the chain.	*
DBGET	Reads data items from IMAGE data sets.	*
DBINF	Provides information about the organization and components of the IMAGE data base being accessed.	*
DBLCK	Locks an IMAGE data base temporarily to provide exclusive access.	*
DBOPN	Initiates access to IMAGE data bases and defines the user's mode of access.	*
DBPUT	Adds new data records to IMAGE data sets.	*
DBUNL	Unlocks a data base previously locked by a call to DBLCK.	*
DBUPD	Modifies the values of data items in existing IMAGE data records.	*
DEB$	Deletes leading and trailing blanks from a string.	E-5
DSABL	Disables a specified task.	11-6

*Refer to HP IMAGE/1000 Data Base Management System Reference Manual, part no. 92063-90001.

SUBROUTINE	DESCRIPTION	REFERENCE
ENABL	Enables a specified task, permits scheduling of a previously disabled task.	11-7
FACT	Sets the ratio between the horizontal and vertical axis of a graph.	17-2
LINES	Plots a line and/or symbols through successive data points in arrays.	17-3
LLEFT	Lifts the plotter pen and moves it to the lower-left corner.	17-4
MPNRM	Clears the event sense mode and erases the channel/bit to trap number correspondence. (6940)	16-3
MTTFS	Writes an end-of-file and rewinds the magnetic tape.	13-6
MTTPT	Positions a magnetic tape forward or backward a certain number of files and/or records.	13-5
MTTRD	Reads a data record from magnetic tape into an array.	13-4
MTTRT	Writes a record onto a magnetic tape.	13-3
NORM	Normalizes the 2313 Subsystem, resets it to a home or known state.	15-7
NUM	Converts the first character of an ASCII string to its decimal value.	6-7 & E-6
NUMB	Plots a number, with or without decimal point, at a specified height, location, and angle.	17-5
PACER	Sets the pace rate of the HP 2313 Subsystem.	15-8
PLOT	Moves the plotter pen from an origin to a destination.	17-6
PLTLU	Defines the logical unit number of the plotter for all plotter calls.	17-7
RDBIT	Reads (or checks the state) of a specified bit on a channel. (6940)	16-4
RDWRD	Reads the contents of a channel into a word. (6940)	16-5
RGAIN	Reads the gain on a particular channel. (2313)	15-9
SADD	Adds one decimal substring to a second decimal substring.	*
SCALE	Scales an array of numbers to fit a specified graph size.	17-8
SDIV	Divides one decimal substring into a second decimal substring.	*

---

*Refer to HP Decimal String Arithmetic Routines Manual, part no. 02100-90140.

SUBROUTINE	DESCRIPTION	REFERENCE
SEDIT	Edits data in one decimal substring using an edit mask within a second decimal substring.	*
SENSE	Sets up link between event sense and a specified trap. Senses a change in the bit pattern. (6940)	16-6
SETP	Sets the priority of a task.	11-8
SFACT	Sets or adjusts the plotter for the particular size paper being used.	17-9
SGAIN	Sets the gain for all channels in a group. (2313)	15-10
SMPY	Multiplies one decimal substring by a second decimal substring.	*
SSUB	Subtracts one decimal substring from a second decimal substring.	*
START	Schedules a task for processing after a specified delay.	11-9
SYMB	Writes characters on a plot.	17-10
TIME	Returns the time according to the system real-time clock.	11-10
TRNON	Executes a task at a specified time.	11-12
TTYS	Sets up a link between a trap number and a teleprinter logical unit number.	11-13
URITE	Moves the plotter pen to the upper right so the paper can be removed.	17-11
WHERE	Indicates the current position of the plotter pen.	17-12
WRBIT	Writes a bit onto a channel. (6940)	16-7
WRWRD	Writes the contents of a word onto a channel. (6940)	16-8

*Refer to HP Decimal String Arithmetic Routines Manual, part no. 02100-90140.

Four types of errors may cause error messages: command errors, statement syntax errors, compiling errors, and execution errors resulting from program execution. The error messages are self-explanatory.

COMMAND ERRORS

Command error messages are printed following the command that caused the error.

SYNTAX ERRORS

When a syntax error in a statement is detected, an error message is printed. You may type a carriage return and enter the statement correctly, or type P to have the erroneous statement reprinted for character editing.

COMPILING ERRORS

These errors are detected following a RUN command but before execution of the program. If no errors are detected, the program will be executed. Otherwise, compilation terminates with no attempt to run the program.

Whenever possible, the line number in which the error occurred will be appended to the message in the form: IN LINE n.

EXECUTION ERRORS

These errors are detected during program execution and printed as they occur; the run terminates.

The line number where the error occurred is appended to all run error messages in the form: IN LINE n, where n is the line number of the statement that caused the error.

OTHER ERRORS

Errors may also occur when you are accessing files. Appropriate error messages are printed and the line number is printed if the error occurs during program execution.

If you are operating under RTE-II (92001B), RTE-III (92060B), or RTE-IV (92067A), the error message "COMMAND FILE NOT FOUND" may also mean that the disc cartridge specified was not found or that there was not enough room on the cartridge.
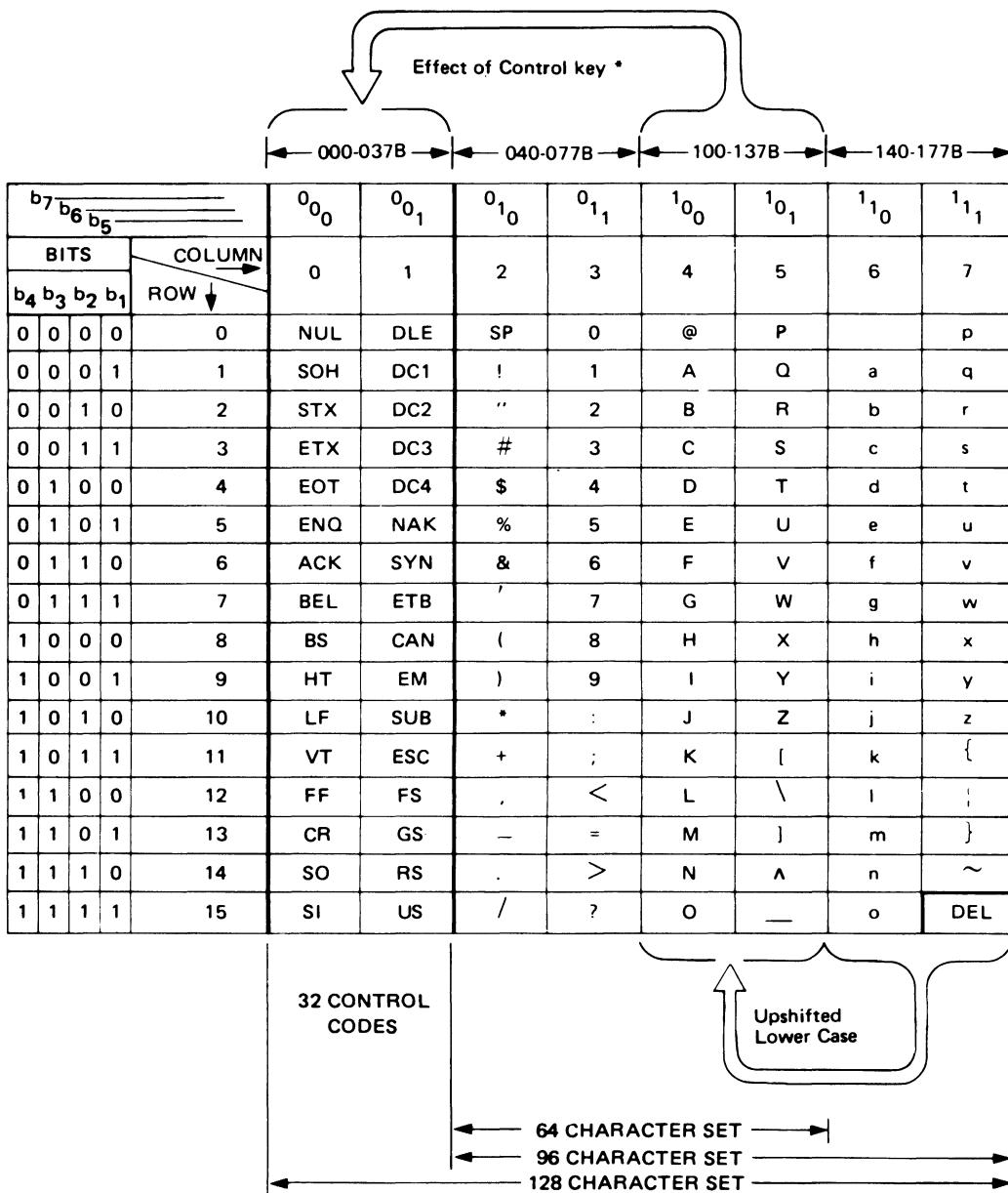
When an IONR (I/O Not Ready) message is issued while using default logical units, you must satisfy the I/O request and make the device ready. For example, if a LOAD command is done from LU 5 (usually the right terminal CTU) and you forgot to insert your mini-cartridge, you will get an IONR error message. Simply insert the mini-cartridge and "UP,eqt" to continue.

In RTE-IVB session environment, if the message FILE MANAGER ERROR is displayed, consult your System Manager. This is a result of any FMGR error with an error code value greater than 41. If an I/O request is made to a logical unit not defined under your session, BASIC issues an error message and allows you to continue.

Effect of Control key *

← 000-037B →		← 040-077B →		← 100-137B →		← 140-177B →	

b_7 b_6 b_5			$^0 0_0$	$^0 0_1$	$^0 1_0$	$^0 1_1$	$^1 0_0$	$^1 0_1$	$^1 1_0$	$^1 1_1$
BITS		COLUMN →	0	1	2	3	4	5	6	7
b_4 b_3 b_2 b_1		ROW ↓								
0 0 0 0		0	NUL	DLE	SP	0	@	P		p
0 0 0 1		1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0		2	STX	DC2	"	2	B	R	b	r
0 0 1 1		3	ETX	DC3	#	3	C	S	c	s
0 1 0 0		4	EOT	DC4	$	4	D	T	d	t
0 1 0 1		5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0		6	ACK	SYN	&	6	F	V	f	v
0 1 1 1		7	BEL	ETB	'	7	G	W	g	w
1 0 0 0		8	BS	CAN	(	8	H	X	h	x
1 0 0 1		9	HT	EM	)	9	I	Y	i	y
1 0 1 0		10	LF	SUB	*	:	J	Z	j	z
1 0 1 1		11	VT	ESC	+	;	K	[	k	{
1 1 0 0		12	FF	FS	,	<	L	\	l	¦
1 1 0 1		13	CR	GS	—	=	M	]	m	}
1 1 1 0		14	SO	RS	.	>	N	∧	n	~
1 1 1 1		15	SI	US	/	?	O	_	o	DEL

32 CONTROL CODES

Upshifted Lower Case

| ← 64 CHARACTER SET → |
| ← 96 CHARACTER SET → |
| ← 128 CHARACTER SET → |

EXAMPLE: The representation for the character "K" (column 4, row 11) is.

```
           b7 b6 b5 b4 b3 b2 b1
BINARY     1  0  0  1  0  1  1
OCTAL      1     1        3
```

* Depressing the Control key while typing an upper case letter produces
the corresponding control code on most terminals. For example,
Control-H is a backspace.

9206-1A

HEWLETT-PACKARD CHARACTER SET FOR COMPUTER SYSTEMS

This table shows HP's implementation of ANS X3 4-1968 (USASCII) and ANS X3 32-1973 Some devices may substitute alternate characters from those shown in this chart (for example. Line Drawing Set or Scandanavian font) Consult the manual for your device

The left and right byte columns show the octal patterns in a 16 bit word when the character occupies bits 8 to 14 (left byte) or 0 to 6 (right byte) and the rest of the bits are zero To find the pattern of two characters in the same word. add the two values For example. "AB" produces the octal pattern 040502 (The parity bits are zero in this chart)

The octal values 0 through 37 and 177 are control codes The octal values 40 through 176 are character codes

Decimal Value	Octal Values		Mnemonic	Graphic[1]	Meaning
	Left Byte	Right Byte			
0	000000	000000	NUL	N_U	Null
1	000400	000001	SOH	S_H	Start of Heading
2	001000	000002	STX	S_X	Start of Text
3	001400	000003	ETX	E_X	End of Text
4	002000	000004	EOT	E_T	End of Transmission
5	002400	000005	ENQ	E_Q	Enquiry
6	003000	000006	ACK	A_K	Acknowledge
7	003400	000007	BEL	Q	Bell, Attention Signal
8	004000	000010	BS	B_S	Backspace
9	004400	000011	HT	H_T	Horizontal Tabulation
10	005000	000012	LF	L_F	Line Feed
11	005400	000013	VT	V_T	Vertical Tabulation
12	006000	000014	FF	F_F	Form Feed
13	006400	000015	CR	C_R	Carriage Return
14	007000	000016	SO	S_O	Shift Out ⎱ Alternate
15	007400	000017	SI	S_I	Shift In ⎰ Character Set
16	010000	000020	DLE	D_L	Data Link Escape
17	010400	000021	DC1	D_1	Device Control 1 (X-ON)
18	011000	000022	DC2	D_2	Device Control 2 (TAPE)
19	011400	000023	DC3	D_3	Device Control 3 (X-OFF)
20	012000	000024	DC4	D_4	Device Control 4 ($\overline{\text{TAPE}}$)
21	012400	000025	NAK	N_K	Negative Acknowledge
22	013000	000026	SYN	S_Y	Synchronous Idle
23	013400	000027	ETB	E_B	End of Transmission Block
24	014000	000030	CAN	C_N	Cancel
25	014400	000031	EM	E_M	End of Medium
26	015000	000032	SUB	S_B	Substitute
27	015400	000033	ESC	E_C	Escape[2]
28	016000	000034	FS	F_S	File Separator
29	016400	000035	GS	G_S	Group Separator
30	017000	000036	RS	R_S	Record Separator
31	017400	000037	US	U_S	Unit Separator
127	077400	000177	DEL	▓	Delete, Rubout[3]

Decimal Value	Octal Values		Character	Meaning
	Left Byte	Right Byte		
32	020000	000040		Space, Blank
33	020400	000041	!	Exclamation Point
34	021000	000042	"	Quotation Mark
35	021400	000043	#	Number Sign, Pound Sign
36	022000	000044	$	Dollar Sign
37	022400	000045	%	Percent
38	023000	000046	&	Ampersand, And Sign
39	023400	000047	'	Apostrophe, Acute Accent
40	024000	000050	(	Left (opening) Parenthesis
41	024400	000051	)	Right (closing) Parenthesis
42	025000	000052	*	Asterisk, Star
43	025400	000053	+	Plus
44	026000	000054	,	Comma, Cedilla
45	026400	000055	−	Hyphen, Minus, Dash
46	027000	000056	.	Period, Decimal Point
47	027400	000057	/	Slash, Slant
48	030000	000060	0	
49	030400	000061	1	
50	031000	000062	2	
51	031400	000063	3	
52	032000	000064	4	
53	032400	000065	5	Digits, Numbers
54	033000	000066	6	
55	033400	000067	7	
56	034000	000070	8	
57	034400	000071	9	
58	035000	000072	:	Colon
59	035400	000073	;	Semicolon
60	036000	000074	<	Less Than
61	036400	000075	=	Equals
62	037000	000076	>	Greater Than
63	037400	000077	?	Question Mark

Decimal Value	Octal Values		Character	Meaning
	Left Byte	Right Byte		
64	040000	000100	@	Commercial At
65	040400	000101	A	
66	041000	000102	B	
67	041400	000103	C	
68	042000	000104	D	
69	042400	000105	E	
70	043000	000106	F	
71	043400	000107	G	
72	044000	000110	H	
73	044400	000111	I	
74	045000	000112	J	
75	045400	000113	K	
76	046000	000114	L	
77	046400	000115	M	Upper Case Alphabet,
78	047000	000116	N	Capital Letters
79	047400	000117	O	
80	050000	000120	P	
81	050400	000121	Q	
82	051000	000122	R	
83	051400	000123	S	
84	052000	000124	T	
85	052400	000125	U	
86	053000	000126	V	
87	053400	000127	W	
88	054000	000130	X	
89	054400	000131	Y	
90	055000	000132	Z	
91	055400	000133	[	Left (opening) Bracket
92	056000	000134	\	Backslash, Reverse Slant
93	056400	000135	]	Right (closing) Bracket
94	057000	000136	^ ↑	Caret, Circumflex; Up Arrow[4]
95	057400	000137	_ ←	Underline, Back Arrow[4]

Decimal Value	Octal Values		Character	Meaning
	Left Byte	Right Byte		
96	060000	000140	`	Grave Accent[5]
97	060400	000141	a	
98	061000	000142	b	
99	061400	000143	c	
100	062000	000144	d	
101	062400	000145	e	
102	063000	000146	f	
103	063400	000147	g	
104	064000	000150	h	
105	064400	000151	i	
106	065000	000152	j	
107	065400	000153	k	
108	066000	000154	l	
109	066400	000155	m	
110	067000	000156	n	Lower Case Letters[5]
111	067400	000157	o	
112	070000	000160	p	
113	070400	000161	q	
114	071000	000162	r	
115	071400	000163	s	
116	072000	000164	t	
117	072400	000165	u	
118	073000	000166	v	
119	073400	000167	w	
120	074000	000170	x	
121	074400	000171	y	
122	075000	000172	z	
123	075400	000173	{	Left (opening) Brace[5]
124	076000	000174	¦	Vertical Line[5]
125	076400	000175	}	Right (closing) Brace[5]
126	077000	000176	~	Tilde, Overline[5]

9206-1C

Notes [1]This is the standard display representation. The software and hardware in your system determine if the control code is displayed, executed, or ignored. Some devices display all control codes as " ", "@", or space

[2]Escape is the first character of a special control sequence. For example, ESC followed by "J" clears the display on a 2640 terminal

[3]Delete may be displayed as "_", "@", or space

[4]Normally, the caret and underline are displayed. Some devices substitute the up arrow and back arrow

[5]Some devices upshift lower case letters and symbols (` through ~) to the corresponding upper case character (@ through ^) For example, the left brace would be converted to a left bracket

RTE SPECIAL CHARACTERS

Mnemonic	Octal Value	Use
SOH (Control A)	1	Backspace (TTY)
EM (Control Y)	31	Backspace (2600)
BS (Control H)	10	Backspace (TTY, 2615, 2640, 2644, 2645)
EOT (Control D)	4	End-of-file (TTY 2615, 2640, 2644, 2645)

9206-1D

SYSTEM GENERATION

Here is a summary of the items that should be loaded at system generation time:

Resident Library	#92101-12002	Always rquired (unless replaced by a dummy module at load time).
Subroutine Library	#92101-12003	Optional.
Instrument Tables	No part number.	Required for specific instruments.
ALARM Program	#92413-16007	Required for Event Sense.

The BASIC Interpreter and Table Generator must not be loaded during RTE system generation. Refer to the Summary of Steps section in this appendix.

If you intend to utilize TRAP and task scheduling statements (see Section XI), the BASIC Resident Library Module (#92101-12002) must be loaded at the time of system generation. This must be done to satisfy references to external subroutines when the BASIC Interpreter is loaded. During the Parameter Input Phase of RTE-III or RTE-IV system generation, the following program types must be changed:

TRAP,30   *Put into SSGA
TTYEV,17   *Memory Resident Program Using SSGA

If you do not require TRAP and task scheduling statement execution, you do not need to include the BASIC Resident Library in your system generation process. However, to satisy the external subroutine references in the BASIC Interpreter, you do need to provide a dummy TRAP module when you load the BASIC software. If you choose to do this, BASIC will operate properly, but TRAP statements and all time scheduling statements will not function. Further, if you use the dummy TRAP module, the execution speed of BASIC is improved because the task scheduling table is not scanned after execution of each scheduling statement. The listing in Figure D-1 shows the dummy TRAP module.

```
0001                      ASMB,R,L,Q
 002   00000                     NAM TRAP
 0003*
 0004* DUMMY TRAP MODULE
 0005* FOR BASIC INTERPRETER
 0006* MUST BE LOADED IF TRAP AND TASK SCHEDULING IS NOT DESIRED
 0007* AND IF XBAMLB IS NOT GENERATED INTO THE SYSTEM
 0008*
 0009                             ENT TRAP
 0010   00000 000000     TRAP    NOP
 0011   00001 000000R             ISZ TRAP
 0012   00002 000000R             JMP TRAP,I
 0013                             END TRAP
**   NO ERRORS *TOTAL **RTE ASMB 92067-16011**
```

Figure D-1. Dummy TRAP Module.

If the HP 6940 or HP 2313 Subsystem is used, the Instrument Tables produced by the Instrument Table Generator (see Section XVIII) must be input during system generation and changed to a type 30 (put into SSGA) module (RTE-III and RTE-IV). If the HP 6940 Event Sense interrupts are required then the ALARM program (#92413-16007) must also be input at that time. If the ALARM program is used, a special entry in the Interrupt Table is required as follows:

    sc,PRG,ALARM   where *sc* is the subchannel of the 6940.

If you plan to schedule tasks from an auxiliary terminal, you must also include an entry in the Interrupt Table as follows:

    sc,PRG,TTYEV   where *sc* is the subchannel of the auxiliary terminal.

<div align="center">NOTE</div>

If your system has a 2313 and a 6940 *and* you are using both the ISA FORTRAN Extension Package and the BASIC 2313 and 6940 subroutines, the following generation error will be reported:

| ERR05 | # GET! | DUPLICATE ENTRY POINT |
| ERR08 | # GET! | DUPLICATE PROGRAM NAME |

Ignore this error message: both # GET! routines are identical.

The BASIC Subroutine Library (#92101-12003) should be loaded during system generation but may be input and stored as a File Manager file. The advantage of loading it at system generation time is that the Loader automatically searches this library for undefined references. The Subroutine Library contains HP supplied software for using the HP 2313, HP 6940, HP 7970, BASIC callable task scheduling routines, and bit manipulation routines. The Resident Library contains all pertinent routines for handling traps, time scheduled tasks, and searching the trap and task tables. The Resident Library must be loaded during system generation (see above) to be able to use TRAP and time scheduling tasks.

LOADING THE INTERPRETER

When the RTE System is operational with all modules loaded as required, you may load the BASIC Interpreter (#92101-12001) by using the File Manager and On-line Loader. BASIC must not be generated during RTE system generation. BASIC can only be added to the system by the On-line Loader. Under RTE-IV, the modules of the BASIC Interpreter must be merged together into one file before loading BASIC.

Here are the required File Manager commands for RTE-II/III systems:

:LG,10	*Assign 10 LG tracks. (Load and GO)*
:MR,*lu*	*Input BASIC relocatable tape part 1 of 3.*
:MR,*lu*	*lu is the logical unit number of the input device (paper tape reader).*
:MR,*lu*	*Repeat for parts 2 and 3 of the tape.*
[:MR,%TRAP]	*Dummy Trap module must be specified here if %BAMLB is not loaded at system generation.*
:SA,LG,%BASIC	*Modules must be merged together for RTE-IV system (see below) also.*
:RU,LOADR,99,1,28,1,2	*Loader command for RTE-II. If you are using RTE-III substitute this command:* *:RU,LOADR,99,1,28,NN001,2 where NN is the number of pages required for BASIC (at least 10).*
:RU,BASIC	*Start the BASIC Interpreter.*

Before loading BASIC on an RTE-IV system, the relocatable modules first must be merged into one file. For RTE-IV, merge %BAIN1, %BAIN2, and %BAIN3 as shown above in the RTE-II/III example. Also refer to the Batch Spool Monitor Reference Manual (part number 92060-90013) regarding the FMGR commands. For RTE-IVB, merge the relocatable files with the utility program, MERGE, which is described in the RTE-IVB Terminal User's Manual (part number 92068-90002).

Here is the command string for loading the BASIC Interpreter for RTE-IV systems. The BASIC Resident Library (part number 92101-12002) has already been loaded at system generation.

:RU,LOADR	
/LOADR: SZ,xx	*xx is the partition size required. xx must be >11*
/LOADR: OP,LB	*Large background program*
/LOADR: OP,SS	*Uses SSGA for TRAP*
/LOADR: REL,%BASIC	*%BASIC is made up of 92101-12001 modules merged together to form one file*
/LOADR:END	*Terminate Loader*
•	
•	
•	
:RU,BASIC	*Start the BASIC Interpreter*

If you do not have %BAMLB generated into your system, you must load a dummy TRAP module to stasify undefined externals:

:RU,LOADR	
/LOADR: LIB,%TRAP	*Load the dummy TRAP module (see Figure D-1).*
/LOADR: SZ,xx	*xx is the partition size required. xx must be >11.*
/LOADR: OP,LB	*Large background program.*
/LOADR: REL,%BASIC	*%BASIC is made of 92101-12001 modules merged together to form one file.*
/LOADR:END	*Terminate Loader*
:RU,BASIC	

The Branch and Mnemonic Table Generator, RTETG, can be loaded by using the same commands as described above, except the tape to be loaded is part number 92101-16008 and 92101-16024 (RTE-II/III), or 92101-16023 (RTE-IV).

To load under RTE-II/III systems:

:LG,10	
:MR,*lu*	*Logical unit of input device containing %BATGN and*
:MR,lu	*%BATG3 (#92101-16024)*
:SA,LG,%RTETG	
:RU,LOADR,99,1,28,1,2	

To load under RTE-IV systems:

:RU,LOADR	
/LOADR: SZ,xx	*override partition size. xx >11.*
/LOADR: OP,LB	
/LOADR: REL,%BATGN	*#92101-16008*
/LOADR: REL,%BATG4	*#92101-16023*

SET UP FILES FOR LOADING OVERLAYS

Next, use the File Manager to load the files used for task scheduling and communication between the 2313 and 6940. The names and part numbers of these files are:

Task Scheduler (#92101-16013)
A2313 (#29102-60016)
A6940 (#29102-16003)

To load the files, use the following File Manager commands:

:ST,*lu*,SCHEDR,BR (to load the task scheduler file)
:ST,*lu*,A2313,BR (to load the A2313 file)
:ST,*lu*,A6940,BR (to load the A6940 file)

Note that *lu* represents the logical unit number of the input unit, the paper tape reader.

SYSTEM CONSIDERATIONS

You must properly prepare your RTE and BASIC system to obtain multiple terminal operation of BASIC using the background swapping capabilities of RTE. The steps required are itemized below:

• The RTE Operating System must be generated to allow background swapping.

• Provide at least 11K background (or partition) area and at least 2K foreground (or partition) area.

• Include at least as many keyboard/printer devices during system generation as are required for BASIC terminals.

• Provide at least 6 ID segments and 11 background ID segments for the BASIC Interpreter, RTETG, and a maximum of 4 overlays.

• Include any instrument drivers at system generation time that might be called by an instrument device subroutine. Also include the Instrument Table generated with the procedure described in Section XVIII.

• Prepare the system for use with the Multi-Terminal Monitor. All requirements and instructions are included in the RTE Programming and Operating Manual.

- Communication between BASIC and overlays (refer to Section XIV) is done by means of class I/O. One class number per overlay is recommended.

- System Available Memory (SAM) is needed for parameter passing for external subroutines in overlays. The area required can be calculated by analyzing the subroutine with the largest amount of data:

    SAM = data +1+ (3*n) words

where data is the total number of words for variables and arrays; n is the number of parameters.

For example, CALL SUB (A(100),I,B) requires:

    200 words for real array A
      1 word for integer I
      2 words for real B
     10 words for parameters (1+3*3)
    ——
    213 words total for SAM

If you cannot determine the exact requirements for SAM, a good rule of thumb is 2K words for SAM.

MULTIPLE COPIES OF BASIC

To prepare multiple copies of BASIC for use on the Multi-User Real-Time BASIC terminals you must use the following File Manager commands:

:SP,BASIC	*Save a copy of the BASIC program in a file.*
:RN,BASIC,BASIX	*Rename the file BASIX.*
:SP,BASIC	*Save another copy of the BASIC program.*
:RN,BASIC,BASIK	*Name the file BASIK.*
:SP,BASIC	*Save another copy of the BASIC program.*
:RN,BASIC,BASIQ	*Name the file BASIQ.*

There are now four versions of BASIC: one permanent program loaded previously as a permanent program, BASIX, BASIK, and BASIQ (copies of the program saved in files which can be loaded by typing the File Manager RUN command). The RUN command creates a temporary ID segment for the copy of BASIC that you are using.

There are other ways of creating multiple access to the BASIC program including making multiple ID segments pointing to the same BASIC program file, but the advantage of doing it this way is that you only have to use the above command sequence once. When you load the RTE system again, the files will still be available and you can run the various versions of BASIC by typing the File Manager RUN command.

Each terminal in the system must be enabled with the following File Manager command:

    :CN,*lu*,20B   *lu* is the logical unit number of the terminal.

When the terminals are enabled and multiple copies of the program have been made, you can press any key on the terminal keyboard and the system prompts with:

    lu >   *lu* is the logical unit number of the terminal you are using.

You now have access to the operating system and can execute the BASIC Interpreter by using the RTE RUN command:

lu >RUN,BASIC	*Run BASIC with RTE run command.*
>BASIC READY	*BASIC indicates it is ready.*
>TABLES *branch table,mnemonic table*	*If you are using subroutines, provide the table names.*

You can run whichever version of BASIC you want as long as no one else is using it.

SUMMARY OF STEPS REQUIRED TO GENERATE A BASIC SYSTEM

The remainder of this appendix contains a sample RTE-II system generation. These are the steps you must follow to prepare your system:

1. Use the Instrument Table Generator to produce the HP 2313 and/or HP 6940 Instrument Tables if you intend to use the subroutines related to these instruments. For RTE-III and RTE-IV systems the Instrument Tables should have their type changed to program type 30.

2. Prepare answers for RTE system generation (see the RTE Programming and Operating Manual for detailed instructions). Make sure the Interrupt Table includes ALARM (for 6940) and TTYEV (for auxiliary teleprinter) if you intend to use them.

3. Generate the RTE system. Be sure to include the BASIC Resident Library (92101-12002) and BASIC Subroutine Library (92101-12003) and the Instrument Table from step 1. For RTE-III and RTE-IV systems, the TRAP and TTYEV modules in 92101-12002 must have their type changed at system generation during the Parameter Input phase. This is done by using the following commands:

TRAP,30	*Put into SSGA*
TTYEV,17	*Memory Resident program with SSGA*

4. Load the new RTE system. Initialize File Manager (FMGR) with the :IN command. (See the Batch-Spool Monitor Reference Manual for detailed instructions.)

5. Load BASIC (92101-12001) and create as many copies as necessary.

6. Load the Branch and Mnemonic Table Generator, RTETG (92101-16008); and the RTE II/III Transfer File Builder, %BATG3 (92101-16024), or RTE-IV Transfer File Builder (92101-16023).

7. Load these files if needed: Task Scheduler, A2313, A6940 — use the File Manager.

8. Prepare the Branch and Mnemonic Table Generator input (see Section XIV). Run RTETG.

9. Run the transfer file which RTETG creates.

10. Run BASIC.

```
* 2313 AND 6940 TABLE GENERATOR *

* TURN ON PUNCH *

# OF 2313'S?
-
2

  SUBSYSTEM #01 CONFIGURATION

LU?
-
12

# HL - SE?
-
32

# HL - DIF?
-
16

# LL, GAIN?
-
8,12.5
-

8,100
-
0,0

# DACS?
-
2

  SUBSYSTEM #02 CONFIGURATION

LU?
-
11

# HL - SE?
-
32

# HL - DIF?
-
16

# LL, GAIN?
-
8,1000
-
8,50
-
0,0
```

2313 and 6940 Instrument Table Generation

Generate Instrument Table for two HP 2313 subsystems and two HP 6940 subsystems.

Required:

1. logical unit numbers,
2. type and number of cards,
3. gains of multiplexer cards,
4. I/O slots for HP 6940's.

Terminal interaction.

DACS?
-
2

OF 6940'S?
-
2

OF CHANNELS IN 2313'S?
-
128

 SUBSYSTEM #01 CONFIGURATION

I/O SLOT?
-
12

LU?
-
7

EVENT?
-
2

DIG INPUT FOR PRESET CNTR W/INT?
-
0

DIGITAL I/O?
-
2

DIGITAL INPUT?
-
0

VOLTAGE DACS?
-
2

CURRENT DACS?
-
2

TIMERS?
-
0

COUNTERS?
-
0

SUBSYSTEM #02 CONFIGURATION

I/O SLOT?
-
14

LU?
-
13

EVENT?
-
2

DIG INPUT FOR PRESET CNTR W/INT?
-

0

DIGITAL I/O?
-
2

DIGITAL INPUT?
-
0

VOLTAGE DACS?
-
2

CURRENT DACS?
-
2

TIMERS?
-
0

COUNTERS?
-
0

* CONFIGURATION CONSTANT PHASE

* ENTER INSTRUMENT CONFIGURATION CONSTANTS
-
/E

* TABLE GENERATION COMPLETE *

```
MH DISC CHNL?
21

# TRKS, FIRST TRK ON SUBCHNL!
   0?
203,0
   1?
203,0
   2?
203,0
   3?
203,0
   4?
203,0
   5?
203,0
   6?
/E

# 128 WORD SECTORS/TRACK?
48

SYSTEM SUBCHNL?
1

SCRATCH SUBCHNL?
1

AUX DISC (YES OR NO OR # TRKS)'
NO

START SCRATCH?
160

TBG CHNL?
13

PRIV. INT. CARD ADDR?
0

FG SWAPPING?
YE

BG SWAPPING?
YE

FG CORE LOCK?
YE

BG CORE LOCK?
YE

SWAP DELAY?
50

LWA MEM?
72272
```

```
PRGM INPT?
MT

LIBR INPT?
PT

PRAM INPT?
TY

INITIALIZE SUBCHNL:
   0?
NO
   2?
NO
   3?
NO
   4?
NO
   5?
NO

PUNCH BOOT?
YES

PUNCH BOOT?
NO


*EOT


  NO UNDEF EXTS


PARAMETERS

O,RTR,1,1
WHZA1,2,1
TATLG,2,98
MEM,1,9000
SYSON,82
ASMB,3,95
FTN4,3,95
XREF,3,96
LOADR,3,97
EDITR,3,50
CNTRL,2,10
DSCMD,3,90
CDUMP,3,90
DDUMP,3,90
AUTOR,3,1
PRMPT,2
RSPNS,2
/E
```

```
CHANGE ENTS?

.MPY,RP,100200
.DIV,RP,100400
.DLD,RP,134200
.DST,RP,104400
/E


# OF BLANK ID SEGMENTS?
10

# OF BLANK BG SEG. ID SEGMENTS?
22

FWA BP LINKAGE?
33


SYSTEM

$CSYS(0099)02000 01777   92001-16012 REV.C 750305

DISPA(0099)02000 03162   92001-16012 750326

RTIME(0099)03163 03734   92001-16012 750305

$ASCM(0099)03735 04014   92001-16012 741120

RTIOC(0099)04075 07716   92002-16022  741125

$ALC (0099)07740 10153   92001-16012 741120

EXEC (0099)10157 11713   92001-16012 750305

$TRRN(0099)11722 12065   92001-16012 750326

SCHED(0099)12131 15202   92002-16012 750225

DVP43(0099)15205 15612   92001-16004 REV.B   741028

$YSLB(0099)15613 15612   92001-16005 REV.B  741120

$BALB(0099)15613 15612   92002-16006 REV.C 750312

DVR00(0099)15626 16717

DVR12(0099)16743 17600

DVR10(0099)17620 20123   7210 PLOTTER DVR,72009-60001 A SCHULTZ

DVA72(0099)20133 22040    09611-16005 REV.A 750617

DVR62(0099)22050 23116   29009-60001 REV. C 75156 -TLD-

DVR23(0099)23117 23762      92202-16001  REV. A
```

BASIC requires a minimum of 2 ID segments, plus one for each overlay.

BASIC requires 11 background ID segments.

Include all required subsystem drivers.

```
DVR31(0099)23764 25154

F4D.C(0099)25157 25156

F2F.B(0099)25157 25156

BP LINKAGE 00171

** OF I/O CLASSES?
10

** OF LU MAPPINGS?
10

** OF RESOURCE NUMBERS?
10

BUFFER LIMITS (LOW, HIGH)?
100,400

* EQUIPMENT TABLE ENTRY

EQT 01?
21,DVR31,D

EQT 02?
15,DVR00,B

EQT 03?
23,DVR23,D,B,T=9999

EQT 04?
17,DVR02,B,T=50

EQT 05?
14,DVA72

EQT 06?
20,DVR12,B,T=100

EQT 07?
12,DVA72

EQT 08?
16,DVR01,T=50

EQT 09?
25,DVR00,B,T=5000

EQT 10?
26,DVR00,B,T=5000

EQT 11?
27,DVR00,B,T=5000

EQT 12?
30,DVR00,B,T=5000
```

```
EQT 13?
31,DVR10,B,T=9999

EQT 14?
04,DVP43

EQT 15?
10,DVR62

EQT 16?
11,DVR62

EQT 17?
/E


*  DEVICE REFERENCE TABLE

 1 = EQT #?
2

 2 = EQT #?
1,1

 3 = EQT #?
0

 4 = EQT #?
4,4

 5 = EQT #?
8

 6 = EQT #?
6

 7 = EQT #?
7,0

 8 = EQT #?
3

 9 = EQT #?
8

10 = EQT #?
1

11 = EQT #?
16,0

12 = EQT #?
15,1

13 = EQT #?
5,1
```

14 = EQT #?
9

15 = EQT #?
2

16 = EQT #?
11

17 = EQT #?
12

18 = EQT #?
13

19 = EQT #?
14

20 = EQT #?
/E

* INTERRUPT TABLE

4,ENT,$POWR

10,EQT,15

11,EQT,16

12,PRG,ALARM ◄────────┐
                       ├────► These entries allow interrupts to be serviced by
14,PRG,ALARM ◄─────────┘   the event sense facility of the 6940 in BASIC.

15,PRG,PRMPT

16,EQT,8

17,EQT,4

20,EQT,6

21,EQT,1

22,EQT,1

23,EQT,3

24,EQT,3

25,PRG,PRMPT

26,EQT,10

27,PRG,TTYEV ◄──────────────── This entry allows interrupts to be serviced by
                               the auxiliary TTY in I/O slot 27.

```
3@,PKG,PRMPT

31,EQT,13

/E


BP LINKAGE 00177

 LIB ADDRS 30633
 CHANGE LIB ADDRS?
0

LIBRARY

TRAP      )30633 31650
   PRTN    31651 31753  92001-16005  741120
   86940   31754 32121

BP LINKAGE 00201


 FG COMMON 00000
 CHANGE FG COMMON?
0

FG RES ADD 32122
 CHANGE FG RES ADD?
0

FG RESIDENTS

D.RTR(0001)32127 34052  92002-16007 750421
   P.PAS    34070 34124  92002-16006 740801

ALARM(0002)34125 34522   92413-16007 07MAY75

TTYEV(0002)34523 34532             29102-60013

MEM  (9000)34533 34627  VERSION 2 NOV, 1973 -TS

BP LINKAGE 00207


FG DSC ADD 34630
 CHANGE FG DSC ADD?
0

FG DISC RESIDENTS

PRMPT(0010)34630 34740  92001-16003 REV.B   741216
   EQLU    34741 35016  92001-16005 741120
```

This is the TRAP and scheduling module used by BASIC (it is required).

HP 6940 and auxiliary TTY modules referenced in the Interrupt Table.

```
RSPNS(0010)34630 34776    92001-16003 REV.B   741002
   EQLU      34777 35054   92001-16005 741120
   MESSS     35055 35147   92001-16005 741120
   .ENTR     35150 35237

TATLG(0098)34630 35143

CNTRL(0010)34630 34757
   .ENTR     34760 35047
   RMPAR     35050 35072
   GETAD     35073 35110

WHZAT(0001)34654 36305    RTE-II ONLY J.F.B.741021
   TMVAL     36323 36342      92001-16005 741120
   .ENTR     36343 36432

SYSON(0090)34630 34664       10 JUL 74 EJW

BP LINKAGE 00213


BP LINKAGE 00213
 CHANGE BP LINKAGE?
600


   SYS AVMEM 36433
   CHANGE  SYS AVMEM?
46000

BG BOUNDRY 46000
  CHANGE BG BOUNDRY?
50000

BG COMMON  00000
  CHANGE BG COMMON ?
0

BG RES ADD 50000
  CHANGE BG RES ADD?
0

BG RESIDENTS

(NONE)


BG DSC ADD 50000
  CHANGE BG DSC ADD?
0

BG DISC RESIDENTS

LOADR(0097)50000 60661    92001-16002 REV.B 741230
```

```
AUTOR(0001)53440 50405
    TRVAL    50430 50425        92001-16005 741120
    FMTIO    50426 51650
    FRMTR    51671 54430
    OOLE     54512 54546
    SNGL     54547 54614
    .XPAK    54615 55011
    .XCOM    55012 55062
    .XFER    55063 55126
    CLRIO    55127 55131
    FLIB     55132 55234
    .ENTR    55235 55324
    .FLUN    55325 55345
    .PACK    55346 55461
    IAND     55462 55471
    PAUSE    55472 55635
    .XFLB    55636 55676
    .OPSY    55677 55736


FMGR (0090)50000 50752      92002-16008 REV.C 750312
    FM.CM    50754 52602
    .DRCT    52700 52706      92001-16005 741120
    IFBRK    52707 52730      92001-16005 741120
    OPEN     52731 53116      92002-16006 741205
    CLOSE    53117 53225      92002-16006  740801
    OOPEN    53226 53434      92002-16006   740801
    RWNDS    53435 53545      92002-16006  740801
    R/WS     53546 53701      92002-16006  /40801
    .ENTR    53702 53771
    RMPAR    53772 54014
    GETAD    54016 54033


FMGRA(0099)54034 54041      92002-16008   740801
    PK..     54042 55465
    COR.A    55466 55501      92001-16005 741120
    CK..     55503 56606
    READF    56735 57462      92002-16006  740801
    REIO     57463 57565      92001-16005 741120
    RWNDF    57566 57647      92002-16006  740801
    NAM..    57650 57744      92002-16006 740801
    P.PAS    57745 57773      92002-16006 740801
    RWSUB    57774 60245      92002-16006 750422
    LOCK.    60247 60316
    FM.UT    60317 61441
    CREA.    61442 61513
    CREAT    61514 61771      92002-16006   741022
    .XFER    61772 62035


FMGR1(0099)54034 54147      92302-16008   741223
    .PARS    54150 55171
    C.TAB    55172 55322      92002-16008  740801
    CA..     55323 55515      92002-16008  741119
    REA.C    55516 55600
    EE..     55601 55642
    TR..     55644 56065
    MR..     56102 56236
    SE..     56237 56423
```

D-16

```
IF..        56424 56634
AD..        56635 57027
OP..        57030 57067
INPRS       57070 57164    92001-16005    741119
READF       57165 57712    92002-16006    740801
REIO        57716 60020    92001-16005    741120
POSNT       60022 60256    92002-16006    740801
P.PAS       60257 60305    92002-16006    740801
RWSUB       60306 60557    92002-16006    750422
WRLG        60560 60575    92002-16006    740801
CK.SM       60576 60711
.XFER       60712 60755
%WRIT       60756 61456
.OPSY       61457 61516


FMGR2(0099)54034 54045    92002-16008    740801
IN.IT       54046 54773
IN..        55001 56652
MC..        56757 57266
RC..        57267 57454
PU..        57455 57677
PURGE       57700 57776    92002-16006    740801
NAM..       57777 60073    92002-16006    740801
J.PUT       60076 60122    92002-16006    740801
IPUT        60123 60143    92002-16006    740801
FID.        60144 60263
MSC.        60264 60320
LOCK.       60321 60370
FM.UT       60371 61513
CNT.        61514 61734
FCONT       61740 62031    92002-16006    740801
.XFER       62037 62102


FMGR3(0099)54034 54041    92002-16008    740801
LI..        54042 55501
DL..        55503 56722
READF       57034 57561    92002-16006    740801
REIO        57562 57664    92001-16005    741120
LOCF        57674 60062    92002-16006    750416
P.PAS       60106 60134    92002-16006    740801
RWSUB       60135 60406    92002-16006    750422
MSC.        60407 60443
FM.UT       60444 61556
.XFER       61567 61632


FMGR4(0099)54034 54050    92002-16008    740801
ST.OU       54051 55305
CO..        55306 56010
F.UTM       56015 56163
OPMES       56164 56364
CREAT       56365 56642    92002-16006    741022
READF       56643 57370    92002-16006    740801
REIO        57371 57473    92001-16005    741120
RWNDF       57474 57555    92002-16006    740801
LOCF        57556 57744    92002-16006    750416
NAM..       57750 60044    92002-16006    740801
P.PAS       60047 60075    92002-16006    740801
```

```
RW$UB        60076 00347   92002-16006  750422
FM.UT        60350 61472
CREA.        61473 61544
CK.SM        61545 61660
.XFER        61661 61724


FMGR5(0099)54034 54043   92002-16008    740801
??..        54044 56157   92002-16008    740801
RU..        56160 56620
RP..        56621 57762
TL..        57763 60002
READF       60004 60531   92002-16006    740801
REIO        60532 60634   92001-16005 741120
LOCF        60635 61023   92002-16006 750416
P.PAS       61024 61052   92002-16006 740801
RW$UB       61053 61324   92002-16006 750422
J.PUT       61325 61351   92002-16006 740801
IPUT        61352 61372   92002-16006    740801
IO.A        61373 61462
BUMP.       61463 61521   92002-16006    741025
SET.T       61522 61550   92002-16006 740801
TL.         61551 61600   92002-16006    741025
ST.TM       61601 61635   92002-16006 741223
.XFER       61636 61701


FMGR6(0099)54034 54044   92002-16008    740801
CN..        54045 54105
JU..        54106 55203
RNRU        55204 55431   92001-16005 741120
SALRN       55432 55537   92001-16005    741106
KCVT        55540 55552   92001-16005 741120
EO..        55555 56400
MESSS       56440 56540   92001-16005 741120
OF..        56541 56634
LG..        56635 57001
PARSE       57002 57021   92001-16005 741120
INPRS       57022 57116   92001-16005    741119
NAMF        57117 57270   92002-16006    740801
READF       57274 60021   92002-16006    740801
REIO        60022 60124   92001-16005 741120
POST        60125 60153   92002-16006    740601
NAM..       60154 60250   92002-16006 740801
P.PAS       60251 60277   92002-16006 740801
RW$UB       60300 60551   92002-16006 750422
SPOPN       60552 60622   92002-16006    741025
SET.T       60623 60651   92002-16006 740801
ST.TM       60652 60706   92002-16006 741223
B.FLG       60707 60755   92002-16006 741118
LULU.       60756 61051   92002-16006 740801
RANGE       61052 61075   92002-16006 740801
ONOFF       61076 61441   92002-16006 750128
EX.TM       61442 61630   92002-16006 741008
IPUT        61631 61651   92002-16006    740801
FREE.       61652 61721   92002-16006 740801
LU.CL       61722 61763   92002-16006    740801
AVAIL       61765 62057   92002-16006 741231
.XFER       62066 62131
```

```
FMGR7(0099)54034  54043    92002-16008    740801
     CL..     54044  54325
     NX.JB    54326  55173
     RNRW     55174  55421    92001-16005  741120
     SALRN    55422  55527    92001-16005  741106
     LU..     55531  56604
     KCVT     56674  56706    92001-16005  741120
     CS..     56707  57104
     READF    57105  57632    92002-16006  740801
     REIO     57633  57735    92001-16005  741120
     FSTAT    57736  57762    92002-16006  740801
     POST     57764  60012    92002-16006  740801
     P.PAS    60017  60045    92002-16006  740801
     RWSUB    60046  60317    92002-16006  750422
     SPOPN    60320  60370    92002-16006  741025
     B.FLG    60371  60437    92002-16006  741118
     LULU.    60440  60533    92002-16006  740801
     RANGE    60534  60557    92002-16006  740801
     AVAIL    60560  60652    92002-16006  741231
     .XFER    60653  60716

FMGR8(0099)54034  54042    92002-16008    740801
     SA..     54043  55015
     SP..     55016  55635
     MS..     55636  56131
     READF    56151  56676    92002-16006  740801
     REIO     56677  57001    92001-16005  741120
     RWNDF    57002  57063    92002-16006  740801
     LOCF     57064  57252    92002-16006  750416
     P.PAS    57253  57301    92002-16006  740801
     RWSUB    57302  57553    92002-16006  750422
     IPUT     57554  57574    92002-16006    740801
     CREA.    57575  57646
     CREAT    57662  60137    92002-16006  741022
     NAM..    60152  60246    92002-16006  740801
     CK.SM    60247  60362
     IO.A     60363  60452
     WRISS    60453  60511    92002-16006  740801
     READ.    60512  60536    92002-16006    740801
     .XFER    60537  60602
     XWRIS    60603  61072
     SREAD    61073  61630
     .OPSY    61631  61670

EDITR(0050)50000  54454    92002-16010  REV.C    750505
     REIO     54455  54557    92001-16005  741120
     CREAT    54560  55035    92002-16006    741022
     OPEN     55036  55223    92002-16006  741205
     READF    55224  55751    92002-16006    740801
     CLOSE    55754  56062    92002-16006    740801
     NAM..    56072  56166    92002-16006  740801
     SOPEN    56167  56375    92002-16006    740801
     P.PAS    56376  56424    92002-16006  740801
     RWSUB    56425  56676    92002-16006  750422
     RWNDS    56677  57007    92002-16006    740801
     R/WS     57010  57143    92002-16006    740801
```

```
     .XFER      57144  57207
     .ENTR      57210  57277
     RMPAR      57300  57322
     GETAD      57323  57340

ASMB (0095)50000  55577      92060-16022 REV.A 750420

ASMB0(0099)55600  56406      92060-16023 REV.A 750420

ASMB1(0099)55600  57630      92060-16024 REV.A 750420

ASMB2(0099)55600  60076      92060-16025 REV.A 750420

ASMB3(0099)55600  56671      92060-16026 REV.A 750602

ASMB4(0099)55600  57331      92060-16027 REV.A 750420

FTN4 (0095)50000  63573

F4.0 (0099)63574  71444

F4.1 (0099)63574  67335

F4.2 (0099)63574  71072

F4.3 (0099)63574  71072

XREF (0096)50000  56166      92060-16028 REV.A 750420
     .OPSY      56167  56226

CDUMP(0090)50000  51046
     ICHAR      51047  51105
     IGET       51106  51120
     FMTIO      51123  52353
     FRMTR      52442  55201
     DBLE       55351  55405
     SNGL       55406  55453
     .XPAK      55454  55650
     .XCOM      55651  55721
     .XFER      55722  55765
     CLRIO      55766  55770
     FLIB       55771  56073
     .ENTR      56075  56165
     .FLUN      56166  56206
     .PACK      56207  56322
     IAND       56323  56332
     RMPAR      56333  56355
     GETAD      56356  56373
     .ZRLB      56374  56434
     .OPSY      56435  56474

DDUMP(0090)50000  50717
     ICHAR      50720  50756
     FMTIO      50762  52212
     FRMTR      52254  55013
     DBLE       55153  55207
     SNGL       55210  55255
```

D-20

```
      .XPAK      55256  55452
      .XCOM      55453  55523
      .XFER      55524  55567
      CLRIO      55570  55572
      FLIB       55573  55675
      .ENTR      55676  55765
      .FLUN      55767  56007
      .PACK      56011  56124
      IAND       56125  56134
      RMPAR      56135  56157
      GETAD      56160  56175
      .ZRLB      56176  56236
      .OPSY      56237  56276

DSCMD(0090)50000  51467
      IPUT       51470  51510      92002-16006      749801
      IGET       51511  51523
      FMTIO      51524  52754
      FRMTR      53101  55640
      DBLE       56000  56034
      SNGL       56035  56102
      .XPAK      56103  56277
      .XCOM      56300  56350
      .XFER      56351  56414
      CLRIO      56415  56417
      FLIB       56420  56522
      .ENTR      56523  56612
      .FLUN      56613  56633
      .PACK      56634  56747
      .GOTO      56750  56772
      IAND       56773  57002
      RMPAR      57003  57025
      GETAD      57026  57043
      .ZRLB      57044  57104
      .OPSY      57105  57144

DSAVE(0090)50001  64300
      CLRIO      64302  64304
      IAND       64305  64314
      RMPAR      64315  64337
      GETAD      64340  64355
      PAUSE      64356  64521
      .OPSY      64522  64561
      .TAPE      64562  64574

DRSTR(0090)50001  64163
      CLRIO      64105  64167
      IAND       64170  64177
      RMPAR      64200  64222
      GETAD      64223  64240
      PAUSE      64241  64404
      .OPSY      64405  64444
      .TAPE      64445  64457

BP LINKAGE  01275


SYSTEM STORED ON DISC
SYS SIZE: 26 TRKS, 011 SECS(10)      RTE-II System Generation is finished.
```

```
SET TIME                                        Load the RTE system.
FMGR 002
:IN,JS,-2,2,BASIC,100                           Initialize File Manager with security code, disc
FMGR-006                                        labels, and first tracks used for files.
:
                                                The FMGR-006 message is printed if there is not
                                                a WELCOME file on the system.


COM       50000 50362                            Load BASIC on-line after RTE system
BASIC     50363 52666    92101-16002 750724      is loaded.
DBUG      52667 52671
SPDUM     52672 52675    92101-16003 750724


REIO      52676 53000    92901-16005 741120
.XFER     53001 53044
.ENTR     53045 53134                            This is the BASIC load map.
FLIB      53135 53237
READF     53240 53765    92002-16006  740801
RMPAR     53766 54010
P.PAS     54011 54037    92002-16006 740801
R/WS      54040 54173    92002-16006  740801
RWSUB     54174 54445    92002-16006 750422
RWNDS     54446 54556    92002-16006  740801
..FCM     54557 54571
.FLUN     54572 54612
..DLC     54613 54627
.PACK     54630 54743
GETAD     54744 54761
.ZRLB     54762 55022
.OPSY     55023 55062




BASC1     55063 60460    92101-16004 750724
MNEM      60461 60551     92101-16005 750724




BASC2     55053 61412    92101-16006 750724

CLOSE     61413 61521    92002-16006  740801
OPEN      61522 61707    92002-16006 741205
$OPEN     61710 62116    92002-16006  740801
MNEM      62117 62207     92101-16005 750724




BASC3     55063 56343    92101-16006 750724

OPEN      56344 56531    92002-16006 741205
$OPEN     56532 56740    92002-16006  740801
CLOSE     56741 57047    92002-16006  740801
```

```
BASC4   55063 02500   92101-16006 750812
BRTBL   62501 62546   92101-16007 750724

AINT    62547 62637
ALOGT   62640 62650
ALOG    62651 62770


SQRT    62771 63117
ATAN    63120 63255
EXP     63256 63420
SICOS   63421 63564
TAN     63565 63730
ABS     63731 63740
PWR2    63741 63770
CHEBY   63771 64067
MANT    64070 64103




BASC5   55063 57201   92101-16008 750724

LURQ    57202 57526   92001-16005 741120
CLOSE   57527 57635   92002-16006  740801
OPEN    57635 60023   92002-16006 741205
NAMF    60024 60175   92002-16006  740801
LOCF    60176 00364   92002-16006 750416
PURGE   60365 60463   92002-16006  740801
CREAT   60464 60741   92002-16006  741022
$OPEN   60742 61150   92002-16006   740801
FCONT   61151 61242   92002-16006 740801
$ALRN   61243 61350   92001-16005  741106
NAM..   61351 61445   92002-16006 740801




BASC6   55063 57502   92101-16008 750724

LURQ    57503 60027   92001-16005 741120
CLOSE   60030 60136   92002-16006  740801
OPEN    60137 60324   92002-16006 741205
LOCF    60325 60513   92002-16006 750416
POSNT   60514 60750   92002-16006  740801
$OPEN   60751 61157   92002-16006   740801
$ALRN   61160 61265   92001-16005  741106




BASC7   55063 60156   92101-16008 750724

CLOSE   60157 60265   92002-16006  740801
OPEN    60266 60453   92002-16006 741205
$OPEN   60454 60662   92002-16006   740801
MNEM    60663 60753    92101-16005 750724
```

```
BASC8   55063  55770   92101-16008 750724

CLOSE   55771  56077   92002-16006  740801
OPEN    56100  56265   92002-16006 741205
$OPEN   56266  56474   92002-16006   740801
RTETG   55000  55142
```

```
T.LNK   55143  55170   92101-16009 750407    Load RTETG (BASIC Table Generator)
                                             on-line.

.XFER   55171  55234
.ENTR   55235  55324
READF   55325  56052   92002-16006   740801
CLOSE   56053  56161   92002-16006   740801
OPEN    56162  56347   92002-16006 741205
RWNDF   56350  56431   92002-16006   740801
PURGE   56432  56530   92002-16006   740801
RMPAR   56531  56553
CREAT   56554  57031   92002-16006   741022
POST    57032  57060   92002-16006   740801
R/W$    57061  57214   92002-16006   740801     This is load map of RTETG.
P.PAS   57215  57243   92002-16006 740801
$OPEN   57244  57452   92002-16006   740801
RW$UB   57453  57724   92002-16006 750422
RWND$   57725  60035   92002-16006   740801
GETAD   60036  60053
REIO    60054  60156   92001-16005 741120
NAM..   60157  60253   92002-16006 740801
```

```
TG00S   60254  60266   92101-16009 750404
T.BTE   60267  62034   92101-16009 750818

.DRCT   62035  62043   92001-16005 741120
```

```
TG01S   60254  60266   92101-16009 750407
T.MES   60267  60661   92101-16009 750407
T.OVL   60662  61541   92101-16009 750507

.DRCT   61542  61550   92001-16005 741120
KCVT    61551  61563   92001-16005 741120
```

```
TG02S   60254  60261   92101-16009 750407
T.TRF   60262  61534

.DRCT   61535  61543   92001-16005 741120
KCVT    61544  61556   92001-16005 741120
CNUMD   61557  61576   92001-16005 741120
IABS    61577  61610
```

```
:RU,RTETG,CO,MN,D
$END RTETG
:TR,TRFL
:LG,10
:MR,%BA00 :          0:          0
:MR,A6940 :          0:          0
:RU,LOADR,99,6,7,     0000
:LG,10
:MR,%BA01 :          0:          0
:MR,A2313 :          0:          0
:RU,LOADR,99,6,7,     0000
:LG,10
:MR,%BA02 :          0:          0
:RU,LOADR,99,6,7,     0000
:LG,10
:MR,%BA03 :          0:          0
:MR,PLOTR :          0:          0
:RU,LOADR,99,6,7,     0000
:LG,10
:MR,%BA04 :          0:          0
:MR,SCHEDR:          0:          0
:RU,LOADR,99,6,7,     0000
: :
:EX
 /LOADR:%BA00 READY
 $END FMGR
 /LOADR:$END
 /LOADR:%BA01 READY
 /LOADR:$END
 /LOADR:%BA02 READY
 /LOADR:$END
 /LOADR:%BA03 READY
 /LOADR:$END
 /LOADR:%BA04 READY
 /LOADR:$END
```

After creation of the RTETG input file, the Table Generator can be executed and the output includes:

1. Branch and Mnemonic Files,
2. Transfer File used to produce overlays,
3. Overlay relocatable files.

After running the Table Generator, RTETG, run the Transfer File using the File Manager. This produces overlays %BA00 through %BA04.

```
%BA00    34641  34662
A6940    34663  35411       29102-16003A    05JUN75
DAC      35412  35561

.ENTR    35562  35651
CALSB    35652  36374       92101-16012 750923
BITCR    36375  36563       29102-60007 REV. A
#GET1    36564  36656        92413-16005 REV A  24AF
ISETC    36657  36745       92101-16011 750724
RMPAR    36746  36770
GETAD    36771  37006
```

These are the load maps
of the overlays:

%BA00

ENTRY POINTS

```
*EXEC    10157
*SLIBR   10367
*SLIBX   11050
*PRTN    31660
*86940   31763
*%BA00   34641
*CALSB   35652
*BBSET   36462
*BEOR    36407
*BIOR    36417
*BAND    36377
*BNOT    36426
*BSHFT   36436
*BBTST   36501
*BBCLR   36523
*ISETC   36660
*DAC     35414
*MPNRM   35265
*RDWRD   34665
*WRWRD   34723
*RDBIT   34702
*WRBIT   34737
*SENSE   34737
*.ENTR   35571
*#GET1   36564
*ERROR   36171
*.ENTP   35562
*ERRCD   36317
*LUERR   36302
*DESPT   36306
*RMPAR   36740
*GETAD   36771
*ADRES   37005
```

```
XBA01    34641  34651                                    %BA01
A2313    34652  35624     29102-80016B  05JUN75
AOV      35625  36000

.ENTR    36001  36070
..ADC    36071  36114
CALSB    36115  36637     9C'?'-'6013  75??23
..FCM    36640  36652
..DLC    36653  36667
RMPAR    36670  36712
GETAD    36713  36730
```

ENTRY POINTS

```
 *EXEC    10157
 *SLIBR   10367
 *SLIBX   11050
 *PRTN    31660
 *.OST   104400
 *XBA01   34641
 *CALSB   36115
 *AISOV   35332
 *AIROV   35025
 *PACER   35514
 *NORM    34753
 *SGAIN   34677
 *RGAIN   34654
 *AOV     35631
 *..ADC   36071
 *..FCM   36640
 *.ENTR   36010
 *ERROR   36434
 *..OAC   36111
 *.ENTP   36001
 *ERRCO   36562
 *LUERR   36545
 *OESPT   36551
 *RMPAR   36670
 *..DLC   36653
 *GETAD   36713
 *AORES   36727
```

%BA02   34641 346·   %BA02

MTTDR   34647 35122   A-92101-16014-1 REV. A
CALSB   35123 35645   92101-16012 750923
.ENTR   35646 35735
PAUSE   35736 36101
RMPAR   36102 36124
.OPSY   36125 36164
GETAD   36165 36202

ENTRY POINTS

    *EXEC   10157
    *$LIBR   10367
    *$LIBX   11050
    *PRTN   31660
    *%BA02   34641
    *CALSB   35123
    *MTTRD   34662
    *MTTRT   34647
    *MTTPT   34755
    *MTTFS   35050
    *.ENTR   35055
    *ERROR   35442
    *.STOP   35774
    *ERRCD   35570
    *LUERR   35553
    *DESPT   35557
    *RMPAR   36102
    *.ENTP   35646
    *.PAUS   35730
    *.OPSY   36125
    *GETAD   36165
    *ADRES   36201

```
%BA03      34041  34056                                          %BA03
PLOT       34057  35207    92469-80001  REV. C
SYMB       35210  36464    92469-80002  REV. C
AXIS       36465  37765
LINES      37766  40503
SCALE      40504  41360
NUMB       41361  42211

.ENTR      42212  42301
FLIB       42302  42404
..FCM      42405  42417
CALSB      42420  43142    92101-16012  750923
ERRB       43143  43215
ALOG       43216  43335
..DLC      43336  43352
SICOS      43353  43516
ABS        43517  43526
IABS       43527  43540
PWR2       43541  43570
.RTOR      43571  43662
.IENT      43663  43723
CHEBY      43724  44022
MANT       44023  44036
.ZRLB      44037  44077
.RTOI      44100  44233
RMPAR      44234  44256
.OPSY      44257  44316
.FLUN      44317  44337
EXP        44340  44502
GETAD      44503  44520
```

ENTRY POINTS

```
*FXEC      10157
*$LIBR     10367
*$LIBX     11050
*PRTN      31660
*.MPY     100200
*.DIV     100400
*.DLD     104200
*.DST     104400
**#PTLU    31651
**#DFAC    31654
**#CFAC    31652
**#XPEN    31656
**#YPEN    31657
*%BA03     34641
*CALSB     42420
*SFACT     35110
*FACT      34703
*WHERE     34661
*PLOT      34731
*LLEFT     35051
*WRITE     35074
*PLTLU     35035
*AXIS      36474
```

```
*NUMB      41367
*SYMB      35216
*LINES     37774
*SCALE     40510
*.ENTR     42221
*FLOAT     42345
*IFIX      42336
*.FDV      42327
*.FMP      42320
*.FAD      42302
*SIN       43364
*COS       43353
*ERRO      43143
*.FSB      42311
*..FCM     42405
*.RTOR     43571
*ALOG      43216
*ABS       43517
*IABS      43527
*.RTOI     44100
*.ENTP     42212
*.ZRLB     44037
*..DLC     43336
*ERRCD     43065
*LUERR     43050
*DESPT     43054
*ERROR     42737
*RMPAR     44234
*.OPSY     44257
*LN        43216
*.FLUN     44317
*.MANT     44023
*.CHEB     43724
*.IENT     43663
*.PWR2     43541
*EXP       44340
*GETAD     44503
*ADRES     44517
```

```
XBA04    34041  34051                        %BA04
SCHD     34052  35427   92101-16013  750724

.ENTR    35430  35517
CALSB    35520  36242   92101-16012  750923
..FCM    36243  36255
..DLC    36256  36272
RMPAR    36273  36315
GETAD    36316  36333
```

ENTRY POINTS

```
*EXEC     10157
*$LIBR    10367
*$LIBX    11050
*PRTN     31060
*TRAP#    31646
*TIME     31145
*TSNXT    31345
*TSEND    31343
*TSTBL    31262
*TSTIM    31346
*TSCNT    31350
*TSPTR    31344
*FINDS    31374
*TRMAK    31471
*TRDEL    31444
*TRTBL    31573
*TRPTR    31635
*TRNXT    31636
*TRFLG    31637
*TRMSK    31644
*TRPNO    31643
*SEKND    31641
*PRIND    31642
*X8A04    34641
*CALSB    35520
*SSETP    34742
*SSTRT    35174
*DSABL    35073
*ENABL    35033
*TRNUN    35157
*TTYS     35301
*ERROR    36037
*.ENTR    35437
*..FCM    36243
*.ENTP    35430
*ERRCO    36165
*LUERR    36150
*DESPT    36154
*RMPAR    36273
*..DLC    36256
*GETAD    36316
*ADRES    36332
```

```
BASIC READY                                    BASIC is ready for use with a complete set
>TABLES BTBL,MTBL                              of subroutines.
>CALLS                                         Use the CALLS command to list them.
 <<NAME(PARAMETER DESCRIPTION) TYPE-OVERLAY#>>
 IBSET(I,I) F 0
 IEOR(I,I) F 0
 OR(I,I) F 0
 AND(I,I) F 0
 NOT(I) F 0
 ISHFT(I,I) F 0
 IBTST(I,I) F 0
 IBCLR(I,I) F 0
 ISETC(RA) F 0
 DAC(I,R) S 0
 MPNRM S 0
 RDWRD(I,IV) S 0
 WRWRD(I,I) S 0
 RDBIT(I,I,IV) S 0
 WRBIT(I,I,I) S 0
 SENSE(I,I,I,I) S 0
 AISQV(I,I,RVA,IV) S 1
 AIRDV(I,RA,RVA,IV) S 1
 PACER(I,I,I) S 1
 NORM S 1
 SGAIN(I,R) S 1
 RGAIN(I,RV) S 1
 AOV(I,RA,RA,IV) S 1
 MTTRD(I,RVA,I,IV,IV) S 2
 MTTRT(I,RA,I,IV,IV) S 2
 MTTPT(I,I,I) S 2
 MTTFS(I,I) S 2
 SFACT(R,R) S 3
 FACT(R,R) S 3
 WHERE(RV,RV) S 3
 PLOT(R,R,I) S 3
 LLEFT S 3
 URITE S 3
 PLTLU(I) S 3
 AXIS(R,R,RA,R,R,R,R) S 3
 NUMB(R,R,R,R,R,I) S 3
 SYMB(R,R,R,RA,R,I) S 3
 LINES(RA,RA,I,I,I,R) S 3
 SCALE(RVA,R,I,I) S 3
 TIME(RV) S 4
 SETP(I,I) S 4
 START(I,R) S 4
 DSABL(I) S 4
 ENABL(I) S 4
 TRNON(I,R) S 4                                Now you may use BASIC as it is described
>                                              in this manual.
```

BASIC/HP-IB DATA CONVERSION

In computer-based HP-IB systems, it is very often necessary to have complete control over manipulation of numeric data formats for a given HP-IB device. This is particularly true of a device that requires a stringent fixed data format in order to operate properly. It is equally desirable to have a free-field conversion capability that will automatically translate different representations of the same data value. Although normal I/O programming statements provide part of this capability, they are primarily for I/O and do not provide for simple memory-to-memory conversions. This appendix provides general data conversion techniques for use in BASIC programming. Included are examples of converting variables to strings, or strings to variables. These subroutines are included in the BASIC Subroutine Library, %BASLB.

DATA CONVERSION REQUESTS — DCODE

BASIC subroutine DCODE converts binary numbers into ASCII-coded strings or vice versa.

```
DCODE(V1,A$,F$)
DCODE(B$,V2,F$)
```

where

V1 = binary variable to be converted.
A$ = string to contain ASCII-coded result. (Note that the string must be predefined as to size and content, as discussed later.)
B$ = ASCII-coded string to be converted.
V2 = variable to contain binary result.
F$ = format statement by which conversion will occur.

The format specification must be contained within parentheses and may be either of the following:
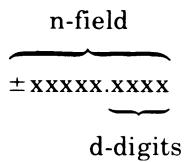
Fn.d = floating point form: n is the number of characters including sign and decimal point; d is number of digits following the decimal point.

En.d = E-format floating point form: n is the number of characters including sign, decimal point, E, and exponent sign; d is number of digits following the decimal point.

BINARY-TO-ASCII

When converting from a binary value to an ASCII string, certain conditions are assumed. The string variable where the converted results will be stored has been predefined. This means that the length of the string has been established (by a DIM statement) and that the contents of the string have been initialized. Thus, when conversion occurs the actual results are placed in the indicated string positions without regard to the string's attributes, such as length, current content, etc. This type of operation facilitates the piecemeal construction of strings as desired. However, this also means that the overall string requirements be anticipated by the user according to his application as demonstrated in the examples given later. (Also, refer to Section IV for an in-depth discussion of strings and Sections II, III, V, VI and VII for a discussion of normal language capability.) Two conversion format types are discussed in the following paragraphs.

Fn.d (F) FORMAT. When using binary-to-ASCII conversions, the Fn.d format performs certain special operations. This format specification generates the following:

n-field
$\overbrace{}$

$\pm$ xxxxx.xxxx
$\underbrace{}$

d-digits

The n-field positions include sign and decimal point as well as the digits. Plus signs are suppressed in the result but minus signs are always supplied. When the magnitude of the converted value is less than the n-field specification, the resulting string is always right-justified with the decimal point in the indicated position. The remainder of the n-field is filled with blank spaces to the left. When the magnitude of the value matches the exact n-field width and the d-digit sub-field is zero, the decimal point is suppressed and the result is an integer string. (Note that rounding off always occurs in the least significant digit of the resulting string.) When the magnitude of the value exceeds the n-field specification, dollar signs ($) appear in the result. This indicates an impossible conversion format was specified by the user and a correction should be made in his program. (See example 1 below.)

EXAMPLE 1. Predefined strings and F-format conversions; ($\wedge$ = blank).

```
10 DIM A$(7),B$(6)          <define string length>
20 LET A$ = "xxx.xxx"       <initialize string content>
30 LET B$ = "(F7.3)"        <define format specification>
40 DCODE (V,A$,B$)          <perform conversion>
```

variable (V)	string result (A$)
1.234	$\wedge\wedge$1.234
12.34	$\wedge$12.340
123.456	123.456
1234.567	1234.57
-1.3579	$\wedge$-1.358
12345600	$$$$$$$
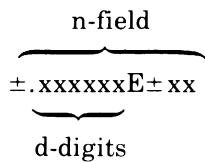
A special use of the F-format is the production of integer strings. The method consists of defining an F-format as Fn.0, where n is the exact number of integer digits to be produced. For example, 123.0 would result in an integer string when the F-format is specified as F3.0. (See example 2 below.)

EXAMPLE 2. F-format conversion to produce an integer string.

```
10 DIM A$(12)                   <define string length>
20 LET A$ = "INTEGER=xxxx"      <initialize string content>
30 DCODE (V,A$(9,12),"(F4.0)")  <perform conversion>
```

variable (V)	string result (A$)
1234.0	INTEGER=1234
-765.432	INTEGER=-765

En.d (E) FORMAT. Like the F-format, the En.d format conversion also provides special operations. This format specification generates the following:

```
          n-field
     ┌──────────────┐
     ±.xxxxxxE±xx
       └──┬──┘
        d-digits
```

The specified n-field positions include mantissa sign, decimal point, all digits, E, exponent sign, and exponent. A plus sign for the mantissa is always suppressed in the result but a minus sign is supplied. The decimal point is also supplied, followed by the d-digits. As in the F-format above, the least significant digit in the resulting string is rounded off. In the exponent part, the E and sign are always supplied, followed by the two-digit exponent. When the converted value requires fewer positions than indicated in the n-field, the result is right-justified and filled with blanks to the left. (See example 3.)

EXAMPLE 3. Use of substrings, literals, and E-format conversion.

```
10 DIM A$(25)                        <define string length>
20 LET A$="VALUE IS±.xxxxE±xx UNITS" <initialize string content>
30 DCODE (V,A$(10,19),"(E10.4)")     <perform conversion>
```

Note the use of substring character positions 10 through 19 to indicate the position in the A$ string where the results are to be placed. Also, note the use of a string literal for the format specification instead of a string variable as in example 1.

variable (V)	string result (A$)
1.234	VALUE IS .1234E+01 UNITS
12.3	VALUE IS .1230E+02 UNITS
123.456	VALUE IS .1235E+03 UNITS
-.00123	VALUE IS -.1230E-02 UNITS
0	VALUE IS .0000E+00 UNITS

ASCII-TO-BINARY

ASCII-to-binary data conversions provide operations somewhat similar to the reverse conversions discussed in the preceding paragraphs. In general, the format specifications indicate n-field positions (columns) of the ASCII string to be converted. Leading and trailing blanks within the n-field are ignored and may be used as data delimiters by the user. Data conversions occur as described in the following paragraphs.

Fn.d (F) FORMAT. The Fn.d format describes the floating point form of the string to be converted. The n-field position establishes the bounds of the string. Within this field, the data conversion takes place according to the actual decimal point position in the string. If a decimal point is not in the string then it is assumed to exist according to the d-digit specification as indicated. (See example 4 below.)

EXAMPLE 4. ASCII-to-binary conversion by F-format; (∧ = blank).

```
10 DIM A$(40),B$(6)          <define string length>
20 READ #12; A$              <input string via LU 12>
30 LET B$="(F7.3)"           <define format specifications>
40 DCODE (A$,V,B$)           <perform conversion>
```

ASCII string (A$)	result (V)
123.456	123.456
123.4∧∧	123.4
∧123.4∧	123.4
∧∧123.4	123.4
-00.123	-.123

En.d (E) FORMAT. The En.d format also uses the n-field to establish string bounds and the conversion occurs according to the decimal point position in the string. If a decimal point is not present, then it is assumed to be positioned as specified by the d-digit part of the format. (See example 5.)

EXAMPLE 5. ASCII-to-binary conversion by the E-format.

```
10 DIM A$(40)                        <define string length>
20 READ #12; A$                      <input string via LU 12>
30 DCODE (A$(5,16),"(E12.6)")        <perform conversion>
```

ASCII string (A$)	result (V)
DCV -.123456E+01	-1.23456
DCV +.123E+03	123.0
ABCD1.379E+00	1.379

BLK$

BASIC subroutine BLK$ initializes a string to a specified number of blanks and resets the logical length of the string to the number of blanks.

Format

CALL BLK$ *(number of blanks, string)*

Parameters

number of blanks   The string is initialized to the number of blanks specified. The range is from 1 to 255 and must not exceed the maximum dimensioned length of the string.

string   a valid dimensioned string array.

BLK$ assumes that the string is a valid string and does not verify the fact.

Examples

```
DIM D$(10)
   .
   .
   .
BLK$(10,D$)
```

Subroutine BLK$ sets the logical length of string D$ to 10 in the first word and will initialize the succeeding 10 characters in the string to blanks.

As a second example, suppose you make the following call to BLK$:

```
DIM F$(10)
   .
   .
   .
BLK$(5,F$)
```

Subroutine BLK$ sets the logical length of string F$ to 5 in the first word and will initialize the succeeding five characters to blanks.

If you specify a first parameter for BLK$ less than 1 or greater than 255, the subroutine produces the following error message:

```
ILLEGAL FIRST PARAMETER-BLK$
```

DEB$

BASIC subroutine DEB$ deletes leading and trailing blanks from a string.

Format

CALL DEB$ *(string)*

Parameters

string   a valid dimensioned string array.

DEB$ assumes that the string is a valid string and does not verify. A valid string must contain at least one non-blank ASCII character.

Examples

```
10 DIM B$(10)
20 LET B$="   ABCD "
30 DEB$(B$)
40 PRINT B$
50 END
```

Statement 40 will print the string without leading and trailing blanks as follows:

    ABCD

DEB$ has changed the logical length from 10 to 4 and the string has been shifted to the left to delete leading blanks.

If there are blanks embedded in the string, DEB$ does not delete them:

```
10 DIM B$(10)
20 LET B$=" AB C D "
30 DEB$(B$)
40 PRINT B$
50 END
```

After the call to DEB$, string B$ prints as follows:

    AB C D

Subroutine DEB$ changes the logical length of the string from 10 to 7 and shifts the string to the left, deleting leading and trailing blanks, but the embedded blanks remain in the string.

There are two error messages that DEB$ can print on the user terminal. The first occurs if the string passed to the subroutine has not been initialized and therefore has a length of zero:

    STRING NOT INITIALIZED-DEB$

The second occurs if the string contains all blanks:

    STRING ALL BLANKS-DEB$

Both messages are for information only. The subroutine takes no action.

NUM and CHRS

Many devices communicate with the RTE Operating System using ASCII characters. Sometimes, these characters are non-printing or pure binary numbers. BASIC needs a means to convert these ASCII characters to decimal values or visa versa. This is handled by function NUM and subroutine CHRS.

The NUM function takes the first character of an ASCII string and converts the character to its decimal value (See Appendix C for the decimal to ASCII equivalence.)

Format

    NUM *(string)*

    Parameters

    string   a string array, substring array, or string literal.

The string array must be defined before the function NUM is used.

Examples

The conversion of the first character of a string array to its decimal value would be:

```
 5 DIM A$[10]
10 LET A$="ABCDE"
20 PRINT NUM(A$)
30 END
```

>RUN

65   *the decimal value of the ASCII character A.*

A substring example would be:

```
 5 DIM A$[10]
10 LET A$="ABCDE"
20 PRINT NUM(A$(2,2))
30 END
```

>RUN

66   *the decimal value of the second element in A$, an ASCII B.*

The conversion of a literal string is:

```
10 PRINT NUM("F")
20 END
```

>RUN

70   *the decimal value of the literal string F.*

The subroutine CHRS takes the decimal value of an ASCII character and places the ASCII character in the specified location of a string variable. The string must be defined prior to the call to the subroutine CHRS.

Format

CALL CHRS *(decimal value, string)*

Parameters

decimal value   a valid specified decimal value of an ASCII character.

string   a string array or substring array in which the ASCII character is to be placed. If a string array is specified, the ASCII character is placed as the first element of the array. Array subscripts can be specified for any element except for the first element (i.e., do not use A$[1,1]; use A$).

The string must be defined prior to calling CHRS.

Examples

The control information for setting the margin on the 9871A line printer is 6 blanks followed by an escape character (decimal value 27) and an M. The 9871A line printer is LU9 in this case and when the print is issued to the line printer (file LP), the control sequence is executed as demonstrated by the listing of the program.

```
10 DIM A$[10]
20 LET A$[1,6]="      "
30 CALL CHRS(27,A$[6,6])
40 LET A$[7,7]="M"
50 PRINT #9;A$
60 END
```

```
>RUN
```

```
>LIST LP
```

```
10 DIM A$[10]
20 LET A$[1,6]="      "
30 CALL CHRS(27,A$[6,6])
40 LET A$[7,7]="M"
50 PRINT #9;A$
60 END
```

READER COMMENT SHEET

Multi-User Real-Time BASIC
Reference Manual

92060-90016   **April 1980**

Update No. _____

(If Applicable)

We welcome your evaluation of this manual. Your comments and suggestions help us improve our publications. Please use additional pages if necessary.

FROM:

Name   _____

Company   _____

Address   _____

_____

_____

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 141 CUPERTINO, CA.

— POSTAGE WILL BE PAID BY —

Hewlett-Packard Company
Data Systems Division
11000 Wolfe Road
Cupertino, California 95014
ATTN: Technical Marketing Dept.
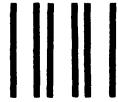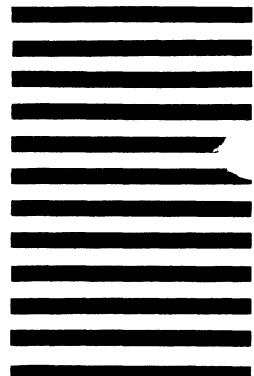
NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

Product Line Sales/Support Key

Key	Product Line
A	Analytical
CM	Components
C	Computer Systems
CP	Computer Systems Primary Service Responsible Office (SRO)
CS	Computer Systems Secondary SRO
E	Electronic Instruments & Measurement Systems
M	Medical Products
MP	Medical Products Primary SRO
MS	Medical Products Secondary SRO
P	Consumer Calculators
*	Sales only for specific product line
**	Support only for specific product line

IMPORTANT: These symbols designate general product line capability. They do not insure sales or support availability for all products within a line, at all locations. Contact your local sales office for information regarding locations where HP support is available for specific products.

HP distributors are printed in italics.

ANGOLA
Telectra
Empresa Tecnica de Equipamentos
Electricos, S.A.R.L.
R. Barbosa Rodrigues, 41.-I. DT.
Caixa Postal 6487
LUANDA
Tel: 35515,35516
A,E,M,P*

ARGENTINA
Hewlett-Packard Argentina S.A.
Avda Santa Fe 2035
Martinez 1640 BUENOS AIRES
Tel: 798-6086, 792-1293
Cable: HEWPACKARG
A,E,CP,P

Biotron S.A.C.I.y.M
Avenida Paseo Colon 221
9 Piso
1399 BUENOS AIRES
Tel: 30-4846, 30-1851, 30-8384,
34-9356, 34-0460, 34-4551
Telex: (33)17595 BIONAR
Cable: BIOTRON Argentina
M

Fate S.A. Electronica
Bartolomeu Mitre 833
1036 BUENOS AIRES
Tel: 74-41011, 74-49277,
74-43459
Telex: 18137, 22754
P

AUSTRALIA
Australia Capital Territory
Hewlett-Packard Australia Pty.Ltd.
121 Wollongong Street
FYSHWICK, A.C.T. 2609
Tel: 804-244
Telex: 62650
Cable: HEWPARD Canberra
A*,CM,CS,E,MS,P

New South Wales
Hewlett-Packard Australia Pty.Ltd.
17-23 Talavera Road
NORTH RYDE, N.S.W. 2113
P.O. Box 308
Tel: 887-1611
Telex: 21561
Cable: HEWPARD Sydney
A,CM,CP,E,MS,P

Queensland
Hewlett-Packard Australia Pty.Ltd.
5th Floor
Teachers Union Building
495-499 Boundary Street
SPRING HILL, Queensland 4000
Tel: 229-1544
Telex: 42133
Cable: HEWPARD Brisbane
A,CM,CS,E,MS,P

South Australia
Hewlett-Packard Australia Pty.Ltd.
153 Greenhill Road
PARKSIDE, S.A. 5063
Tel: 272-5911
Telex: 82536
Cable: HEWPARD Adelaide
A*,CM,CS,E,MS,P

Victoria
Hewlett-Packard Australia Pty.Ltd.
31-41 Joseph Street
BLACKBURN, Victoria 3130
Tel: 89-6351
Telex: 31-024
Cable: HEWPARD Melbourne
A,CM,CP,E,MS,P

Western Australia
Hewlett-Packard Australia Pty.Ltd.
141 Stirling Highway
NEDLANDS, W.A. 6009
Tel: 386-5455
Telex: 93859
Cable: HEWPARD Perth
A,CM,CS,E,MS,P

AUSTRIA
Hewlett-Packard Ges.m.b.h.
Veraufsbuero Graz
Grottenhofstrasse 94
A-8052 GRAZ
Tel: 21-5-66
Telex: 32375
CM,C*,E*

Hewlett-Packard Ges.m.b.h.
Wehlistrasse 29
P.O. Box 7
A-1205 VIENNA
Tel: (222) 35-16-210
Telex: 135823/135066
A,CM,CP,E,MS,P

BAHRAIN
Green Salon
P.O. Box 557
BAHRAIN
Tel: 5503
Telex: 88419
P

Wael Pharmacy
P.O. Box 648
BAHRAIN
Tel: 54886, 56123
Telex: 8550 WAEL GJ
M

BANGLADESH
The General Electric Co. of
Bangladesh Ltd.
Magnet House 72
Dilkusha Commercial Area
MOTIJHELL, Dacca 2
Tel: 252415, 252419
Telex: 734
Cable: GECDAC Dacca
A,E,M

BELGIUM
Hewlett-Packard Belgium S.A./N.V.
Boulevard de la Woluwe 100
Woluwedal
B-1200 BRUSSELS
Tel: (02) 762-32-00
Telex: 23-494 B
A,CM,CP,E,MP,P

BRAZIL
Hewlett-Packard do Brasil I.e.C.
Ltda.
Alameda Rio Negro, 750
ALPHAVILLE 06400 Barueri SP
Tel: 421-1311
Telex: 011 23602 HPBR-BR
Cable: HEWPACK Sao Paulo
A,CM,CP,E,MS

Hewlett-Packard do Brasil I.e.C.
Ltda.
Rua Padre Chagas, 32
90000-PORTO ALEGRE-RS
Tel: 22-2998, 22-5621
Cable: HEWPACK Porto Alegre
A*,CM,E,MS,P*

Hewlett-Packard do Brasil I.e.C.
Ltda.
Avenida Epitacio Pessoa, 4664
20000 RIO DE JANEIRO-RJ
Tel: 286-0237
Telex: 021-21905 HPBR-BR
Cable: HEWPACK Rio de Janeiro
A,CM,E,MS,P*

BURUNDI
Typomeca S.P.R.L.
B.P. 553
BUJUMBURA
Tel: 2659
P

CANADA
Alberta
Hewlett-Packard (Canada) Ltd.
210, 7220 Fisher Street S.W.
CALGARY, Alberta T2H 2H8
Tel: (403) 253-2713
Telex: 610-821-6141
A,CM,CP,E*,MS,P*

Hewlett-Packard (Canada) Ltd.
11620A-168th Street
EDMONTON, Alberta T5M 3T9
Tel: (403) 452-3670
Telex: 610-831-2431
A,CM,CP,E,MS,P*

British Columbia
Hewlett-Packard (Canada) Ltd.
10691 Shellbridge Way
RICHMOND, British Columbia V6X
2W7
Tel: (604) 270-2277
Telex: 610-922-5059
A,CM,CP,E*,MS,P*

Manitoba
Hewlett-Packard (Canada) Ltd.
380-550 Century Street
Saint James
WINNIPEG, Manitoba R3H 0L8
Tel: (204) 786-6701
Telex: 610-671-3531
A,CM,CS,E,MS,P*

Nova Scotia
Hewlett-Packard (Canada) Ltd.
P.O. Box 931
900 Windmill Road
DARTMOUTH, Nova Scotia B2Y 3Z6
Tel: (902) 469-7820
Telex: 610-271-4482
CM,CP,E*,MS,P*

Ontario
Hewlett-Packard (Canada) Ltd.
552 Newbold Street
LONDON, Ontario N6E 2S5
Tel: (519) 686-9181
Telex: 610-352-1201
A,CM,CS,E*,MS,P*

Hewlett-Packard (Canada) Ltd.
6877 Goreway Drive
MISSISSAUGA, Ontario L4V 1M8
Tel: (416) 678-9430
Telex: 610-492-4246
A,CM,CP,E,MP,P

Hewlett-Packard (Canada) Ltd.
1020 Morrison Drive
OTTAWA, Ontario K2H 8K7
Tel: (613) 820-6483
Telex: 610-563-1636
A,CM,CP,E*,MS,P*

Quebec
Hewlett-Packard (Canada) Ltd.
275 Hymus Boulevard
POINTE-CLAIRE, Quebec H9R 1G7
Tel: (514) 697-4232
Telex: 610-422-3022
A,CM,CP,E,MP,P*

CHILE
Jorge Calcagni y Cia. Ltda.
Arturo Burhle 065
Casilla 16475
Correo 9, SANTIAGO
Tel: 220222
Telex: JCALCAGNI
A,E,M,P

Olympia (Chile) Ltd.
Rodrico de Araya 1045
Casilla 256-V
SANTIAGO 21
Tel: 25-50-44
Telex: 40-565
P

COLOMBIA
Instrumentacion
H. A. Langebaek & Kier S.A.
Carrera 7 No. 48-75
BOGOTA 2, DE
Apartado Aereo 6287
BOGOTA 1 D.E.
Tel: 269-8877
Telex: 44400
Cable: AARIS Bogota
A,E,M,P

Instrumentacion
H.A. Langebaek & Kier S.A.
Edif. Camacol, Local 105
Carrera 63 NO. 49-A-31
Apartado 54098
MEDELLIN
Tel: 304475
A,E,M,P

COSTA RICA
Cientifica Costarricense S.A.
Avenida 2, Calle 5
San Pedro de Montes de Oca
Apartado 10159
SAN JOSE
Tel: 24-38-20, 24-08-19
Telex: 2367 GALGUR CR
Cable: GALGUR
A,E,M,P

CYPRUS
Kryponics
19 Gregorios Xenopoulos Street
P.O. Box 1152
NICOSIA
Tel: 45628, 45629
Telex: 3018
E,M,P

CZECHOSLOVAKIA
Vyojova a Provozni Zakladna
Vyzkumnych Ustavu v Bechovicich
CSSR-25097 BECHOVICE U PRAHY
Tel: 89-9341
Telex: 12133
*P**

Hewlett-Packard
Obchodni Zastupitelstvi v CSSR
Pisemny styk
Post. schranka 27
CS 118 01 PRAHA 011
Tel: 66-296
Telex: 121353 1HC
A,C*,E*,M*,P**

DENMARK
Hewlett-Packard A/S
Datavej 52
DK-3460 BIRKEROD
Tel: (02) 81-66-40
Telex: 37409 hpas dk
A,CM,CP,E,MS,P

Hewlett-Packard A/S
Navervej 1
DK-8600 SILKEBORG
Tel: (06) 82-71-66
Telex: 37409 hpas dk
CM,CS,E

ECUADOR
CYEDE Cia. Ltda.
P.O. Box 6423 CCI
Avenida Eloy Alfaro 1749
QUITO
Tel: 450-975, 243-052
Telex: 2548 CYEDE ED
Cable: CYEDE-Quito
A,E,P

Hospitalar S.A.
Casilla 3590
Robles 625
QUITO
Tel: 545-250, 545-122
Cable: HOSPITALAR-Quito
M

EGYPT
Samitro
Sami Amin Trading Office
18 Abdel Aziz Gawish
ABDINE-CAIRO
Tel: 24-932
P

International Engineering Associates
24 Hussein Hegazi Street
Kasr-el-Aini
CAIRO
Tel: 23-829
Telex: 93830
E,M

EL SALVADOR
IPESA
Boulevard de los Heroes
Edificio Sarah 1148
SAN SALVADOR
Tel: 252787
A,C,E,M,P*

ETHIOPA
Abdella Abdulmalik
P.O. Box 2635
ADDIS ABABA
Tel: 11-93-40
A,E,M

FINLAND
Hewlett-Packard Oy
Revontulentie 7
SF-02100 ESPOO 10
Tel: (90) 455-0211
Telex: 121563 hewpa sf
A,CM,CP,E,MS,P

FRANCE
Hewlett-Packard France
Le Ligoures
Bureau de Vente de
*   Aix-en-Provence*
Place Romee de Villeneuve
F-13090 AIX-EN-PROVENCE
Tel: (42) 59-41-02
Telex: 410770F
A,CM,CS,E,MS,P

Hewlett-Packard France
Bureau de Vente de Lyon
Chemin des Mouilles
Boite Postale No. 162
F-69130 ECULLY Cedex
Tel: (78) 33-81-25
Telex: 310617F
A,CM,CP,E,MP

Hewlett-Packard France
Immeuble France Evry
Tour Lorraine
Boulevard de France
F-91035 EVRY Cedex
Tel: (60) 77-96-60
Telex: 692315F
CM,E

Hewlett-Packard France
Batiment Ampere
Rue de la Commune de Paris
Boite Postale 300
F-95153 LE BLANC MESNIL
Tel: (01) 865-44-52
Telex: 211032F
CM,CP,E,MS

Hewlett-Packard France
Avenue du President JF Kennedy
F-33700 MERIGNAC
Tel: (56) 34-00-84
Telex: 550105F
CM,CP,E,MS

Hewlett-Packard France
32 Rue Lothaire
F-57000 METZ
Tel: (87) 65-53-50
CM,CS

Hewlett-Packard France
Avenue des Tropiques
Zone d'activities de Courtaboeuf
Boite Postale 6
F-91401 ORSAY Cedex
Tel: (1) 907-78-25
Telex: 600048F
A,CM,CP,E,MP,P

Hewlett-Packard France
15 Boulevard De L'Amiral Bruix
F-75016 PARIS
Tel: (01) 502-12-20
Telex: 613663F
CM,CP,MS,P

Hewlett-Packard France
2 Allee de la Bourgonette
F-35100 RENNES
Tel: (99) 51-42-44
Telex: 740912F
*CM,CS,E,MS,P**

Hewlett-Packard France
4 Rue Thomas Mann
F-67033 STRASSBOURG Cedex
Tel: (88) 28-56-46
Telex: 890141F
*CM,CS,E,MS,P**

Hewlett-Packard France
20 Chemin de la Cepiere
31081 TOULOUSE Cedex
Tel: (61) 40-11-12
Telex: 531639F
*A,CM,CS,E,P**

Hewlett-Packard France
Bureau de Vente de Lille
Immeuble Pericentre
Rue Van Gogh
F-59650 VILLENEUVE D'ASQ
Tel: (20) 91-41-25
Telex: 160124F
*CM,CS,E,MS,P**

**GERMAN FEDERAL
    REPUBLIC**
Hewlett-Packard GmbH
Technisches Buro Berlin
Keithstrasse 2-4
D-1000 BERLIN 30
Tel: (030) 24-90-86
Telex: 018 3405 hpbln d
CM,CS,E,P

Hewlett-Packard GmbH
Technisches Buro Boblingen
Herrenberger Strasse 110
D-7030 BOBLINGEN
Tel: (07031) 667-1
Telex: 07265739 bbn
A,CM,CP,E,MP,P

Hewlett-Packard GmbH
Technisches Buro Dusseldorf
Emanuel-Leutze-Strasse 1
D-4000 DUSSELDORF
Tel: (0211) 5971-1
Telex: 085/86 533 hpdd d
A,CM,CP,E,MS,P

Hewlett-Packard GmbH
Vertriebszentrale Frankfurt
Berner Strasse 117
Postfach 560 140
D-6000 FRANKFURT 56
Tel: (0611) 50-04-1
Telex: (841) 04 13249 hpffm d
A,CM,CP,E,MP,P

Hewlett-Packard GmbH
Technisches Buro Hamburg
Kapstadtring 5
D-2000 HAMBURG 60
Tel: (040) 63804-1
Telex: 021 63 032 hphh d
A,CM,CP,E,MS,P

Hewlett-Packard GmbH
Technisches Buro Hannover
Am Grossmarkt 6
D-3000 HANNOVER 91
Tel: (0511) 46-60-01
Telex: 092 3259
A,CM,CS,E,MS,P

Hewlett-Packard GmbH
Technisches Buro Nurnberg
Neumeyerstrasse 90
D-8500 NURNBERG
Tel: (0911) 56-30-83
Telex: 0623 860
CM,CS,E,MS,P

Hewlett-Packard GmbH
Technisches Buro Munchen
Eschenstrasse 5
D-8021 TAUFKIRCHEN
Tel: (089) 6117-1
Telex: 0524985
A,CM,CP,E,MS,P

GREAT BRITAIN
Hewlett-Packard Ltd.
Trafalgar House
Navigation Road
ALTRINCHAM
Chesire WA14 1NU
Tel: (061) 928-6422
Telex: 668068
A,CM,CP,E,MS*

Hewlett-Packard Ltd.
Lorrilleaux Bolton Premises
Morely Road, Staplehill
BRISTOL BS16 4QT
Tel: (0272) 570743
CM,CS,MS

Hewlett-Packard Ltd.
14 Wesley Street
CASTLEFORD
Yorkshire WF10 1AE
Tel: (0977) 550016
Telex: 5557355
CM,CP

Hewlett-Packard Ltd.
9 Savoy Street
LONDON WC2R 0BA
Tel: 013797700
CM,CP

Hewlett-Packard Ltd.
Fourier House
257-263 High Street
LONDON COLNEY, St. Albans
Herts., AL21HA
Tel: (0727) 24400
Telex: 1-8952716
CM,CP,E,MS

Hewlett-Packard Ltd
Tradax House, St. Mary's Walk
MAIDENHEAD
Berkshire, SL6 1ST
Tel: (0628) 39151
CM,CP

Hewlett-Packard Ltd.
308/314 Kings Road
READING, Berkshire
Tel: 61022
Telex: 84-80-68
A,CM,E,MS*

Hewlett-Packard Ltd.
Quadrangle
106-118 Station Road
REDHILL, Surrey RHI IPS
Tel: (0737) 68655
A,CM,CP,E,MS,P

Hewlett-Packard Ltd.
Westminster House
190 Stratford Road
SHIRLEY, SOLIHULL
West Midlands B90-3BJ
Tel: (021) 7458800
Telex: 339105
CM,CP,MS

GREECE
Kostas Karaynnis
8 Omirou Street
ATHENS 133
Tel: 32-30-303, 32-37-371
Telex: 21 59 62 RKAR GR
E,M,P

"Plaiso"
G. Gerados
24 Stournara Street
ATHENS
Tel: 36-11-160
Telex: 21 9492
P

GUAM
Guam Medical Supply, Inc.
Jay Ese Bldg., Room 210
P.O. Box 8947
TAMUNING 96911
Tel: 6464513
Cable: EARMED Guam
M,P

GUATEMALA
IPESA
Avenida Reforma 3-48
Zona 9
GUATEMALA CITY
Tel: 316627, 314786, 664715
Telex: 4192 Teltro Gu
A,C,E,M,P

HONG KONG
Hewlett-Packard Hong Kong, Ltd.
Room 105, Austin Center
1st Floor
21 Austin Avenue
TST P.O. Box 98524
KOWLOON, Hong Kong
Tel: 3-721143/8
Telex: 36678 HEWPA HX
Cable: PASIALTO Hong Kong
E,CP,P

Hewlett-Packard Hong Kong, Ltd.
11th Floor,Four Seas Building
212 Nathan Road
P.O. Box 795
KOWLOON, Hong Kong
Tel: 3697446
Telex: 36678 HEWPA HX
Cable: HEWPACK Hong Kong
E,CP,P

Schmidt & Co. (Hong Kong) Ltd.
Wing On Centre, 28th Floor
Connaught Road, C.
HONG KONG
Tel: 5-455644
Telex: 74766 SCHMX HX
A,M

ICELAND
Elding Trading Company Inc.
Hafnarnvoli-Tryggvagotu
P.O. Box 895
IS-REYKJAVIK
Tel: 1-58-20, 1-63-03
M

INDIA
Blue Star Ltd.
Bhavdeep
Stadium Road
AHMEDABAD 380 014
Tel: 42932
Telex: 012-234
Cable: BLUEFROST
E

Blue Star Ltd.
11 Magarath Road
BANGALORE 560 025
Tel: 55668
Telex: 0845-430
Cable: BLUESTAR
A,CM,C,E

Blue Star Ltd.
Band Box House
Prabhadevi
BOMBAY 400 025
Tel: 45-73-01
Telex: 011-3751
Cable: BLUESTAR
A,M

Blue Star Ltd.
Sahas
414/2 Vir Savarkar Marg
Prabhadevi
BOMBAY 400 025
Tel: 46 65 55
Telex: 011-4093
Cable: FROSTBLUE
A,CM,C,E,M

Blue Star Ltd.
7 Hare Street
CALCUTTA 700 001
Tel: 12-01-31
Telex: 021-7655
Cable: BLUESTAR
A,M

Blue Star Ltd.
Meenakshi Mandiram
XXXXV/1379-2 Mahatma Gandhi
*   Road*
COCHIN 682-016
Tel: 32069
Telex: 085-514
Cable: BLUESTAR
*A**

Blue Star Ltd.
133 Kodambakkam High Road
MADRAS 600 034
Tel: 82057
Telex: 041-379
Cable: BLUESTAR
A,M

Blue Star Ltd.
Bhandari House, 7th/8th Floors
91 Nehru Place
NEW DELHI 110 024
Tel: 682547, 682970
Telex: 031-2463
Cable: BLUESTAR
A,CM,C,E,M

Blue Star Ltd.
1-1-117/1 Sarojini Devi Road
SECUNDERABAD 500 033
Tel: 70126
Telex: 0155-459
Cable: BLUESTAR
A,E

Blue Star Ltd.
T.C. 7/603 Poornima
Maruthankuzhi
TRIVANDRUM 695 013
Tel: 65799
Telex: 0884-259
Cable: BLUESTAR
E

INDONESIA
BERCA Indonesia P.T.
P.O.Box 496/Jkt.
JL. Abdul Muis 62
JAKARTA
Tel: 373009
Telex: 46748 BERSAL IA
Cable: BERSAL
A,E,M,P

BERCA Indonesia P.T.
J.L. Jimento 23
SURABAYA
Tel: 42027
Telex: 31146 BERSAL S.D.
Cable: BERCACON
A*,E,M,P

IRAQ
Hewlett-Packard Trading S.A.
Mansoor City 9B/3/7
BAGHDAD
Tel: 5514973
Telex: 2455 HEPAIRAQ 1k
CP

IRELAND
Hewlett-Packard Ltd.
Kestrel House
Clanwilliam Place
Lower Mount Street
DUBLIN 2, Eire
Tel: 680424, 680426
Telex: 30439
A,E,P*

Hewlett-Packard Ltd.
2C Avongberg Ind. Est.
Long Mile Road
DUBLIN 12, Eire
Tel: 514322, 514224
Telex: 30439
A*,CP,E,MS,P*

Cardiac Services Ltd.
Kilmore Road
Artane
DUBLIN 5, Eire
Tel: (OJ) 315820
M

ISRAEL
Electronics & Engineering Div.
Motorola Israel Ltd.
16 Kremenetski Street
P.O. Box 25016
TEL-AVIV
Tel: 338973
Telex: 33569
Cable: BASTEL Tel-Aviv
A,CM,C,E,M,P

ITALY
Hewlett-Packard Italiana S.p.A.
Via Martin Luther King, 38/III
I-40132 **BOLOGNA**
Tel: (051) 402394
Telex: 511630
CM,CS,E,MS

Hewlett-Packard Italiana S.p.A.
Via G. Di Vittorio 9
I-20063 **CERNUSCO SUL NAVLIGLIO**
Tel: (2) 903691
Telex: 334632
A,CM,CP,E,MP,P

Hewlett-Packard Italiana S.p.A.
Via Nuova san Rocco A
   Capadimonte, 62A
I-80135 **NAPOLI**
Tel: (081) 7413544
A,CM,CS,E

Hewlett-Packard Italiana S.p.A.
Via Turazza 14
I-35100 **PADOVA**
Tel: (49) 664888
Telex: 430315
A,CM,CS,E,MS

Hewlett-Packard Italiana S.p.A.
Via G. Armellini 10
I-00143 **ROMA**
Tel: (06) 546961
Telex: 610514
A,CM,CS,E,MS,P*

Hewlett-Packard Italiana S.p.A.
Corso Giovanni Lanza 94
I-10133 **TORINO**
Tel: (011) 682245, 659308
Telex: 221079
CM,CS,E

JAPAN
Yokogawa-Hewlett-Packard Ltd.
Inoue Building
1348-3, Asahi-cho
ATSUGI, Kanagawa 243
Tel: (0462) 24-0451
CM,C*,E

Yokogawa-Hewlett-Packard Ltd.
Kumagaya Ashai Building
4 Tusukuba, 3-chome
KUMAGAYA, Saitama 360
Tel: (0485) 24-6563
CM,CS,E

Yokogawa-Hewlett-Packard Ltd.
Mito Mitsui Building
4-73, San-no-maru, 1-chome
MITO, Ibaragi 310
Tel: (0292) 25-7470
CM,CS,E

Yokogawa-Hewlett-Packard Ltd.
Sunitomo Seimei Bldg.
11-2 Shimo-sasajima-cho
Nakamura-ku
NAGOYA, Aichi 450
Tel: (052) 581-1850
CM,CS,E,MS

Yokogawa-Hewlett-Packard Ltd.
Chuo Bldg., 4TH FLOOR
54-20 Nishinakajima, 5-chome
Yodogawa-ku, Osaka-shi
OSAKA, 532
Tel: (06) 304-6021
Telex: 523-3624 YHPOSA
A,CM,CP,E,MP,P*

Yokogawa-Hewlett-Packard Ltd.
29-21 Takaido-Higashi 3-chome
Suginami-ku **TOKYO** 168
Tel: (03) 331-6111
Telex: 232-2024 YHPTOK
Cable: YUHPMARKET TOK23 724
A,CM,CP,E,MP,P*

Yokogawa-Hewlett-Packard Ltd.
Tanigawa Building
2-24-1 Tsuruya-cho
Kanagawa-ku
YOKOHAMA, Kanagawa 221
Tel: (045) 312-1252
Telex: 382-3204 YHP YOK
CM,CS,E

JORDAN
Mouasher Cousins Company
P.O. Box 1387
AMMAN
Tel: 21456, 24907, 39907
Telex: 21456 SABCO JO
E,M,P

KENYA
International Aeradio (E.A.) Ltd.
P.O. Box 95221
MOMBASA
M

ADCOM Ltd., Inc.
City House, Wabera Street
P.O. Box 30635
NAIROBI
Tel: 331955
Telex: 22639
A*,E,M

International Aeradio (E.A.) Ltd
P.O. Box 19012
Nairobi Airport
NAIROBI
Tel: 336055, 336056
Telex: 22201, 22301
M

KOREA
Samsung Electronics
C.P.O. 2775
SEOUL
Tel: 8334311, 8330002, 8330006
Telex: SAMSAN 27364
A,C,E,M,P

KUWAIT
Al-Khalidya Trading & Contracting
P.O. Box 830 Safat
KUWAIT
Tel: 42-4910, 41-1726
Telex: 2481 Areeg kt
A,E,M

Photo & Cine Equipment
P.O. Box 270 Safat
KUWAIT
Tel: 42-2846, 42-3801
Telex: 2247 Matin
P

LUXEMBOURG
Hewlett-Packard Belgium S.A./N.V.
Boulevard de la Woluwe 100
Woluwedal
B-1200 **BRUSSELS**
Belgium
Tel: 762/32/00
Telex: 23-494 paloben bru
A,CP,E,MP,P

MALAYSIA
Hewlett-Packard Sales (Malaysia)
   Sdn. Bhd.
Suite 2.21/2.22
Bangunan Angkasa Raya
Jalan Ampang
KUALA LUMPUR
Tel: 483544
Telex: MA31011
A,CP,E,M,P*

Protel Engineering
P.O. Box 1917
Lot 319, Satok Road
Kuching, **SARAWAK**
Tel: 53544
Telex: MA 70904 PROMAL
Cable: PROTELENG
A,E,M

MEXICO
Hewlett-Packard Mexicana, S.A. de
   C.V.
Av. Periferico Sur No. 6501
Tepepan, Xochimilco
MEXICO CITY 23, D.F.
Tel: (905) 676-4600
Telex: 017-74-507
A,CP,E,MS,P

Hewlett-Packard Mexicana, S.A. de
   C.V.
Rio Volga #600 Colonia del Valle
MONTERREY, N.L.
Tel: 78-42-93, 78-42-40, 78-42-41
Telex: 038-410
CS

MOROCCO
Dolbeau
81 rue Karatchi
CASABLANCA
Tel: 3041-82, 3068-38
Telex: 23051, 22822
E

Gerep
2 rue d'Agadir
Boite Postale 156
CASABLANCA
Tel: 272093, 272095
Telex: 23 739
P

MOZAMBIQUE
A.N. Goncalves Ltd.
162, 1° Apt. 14 Av. D. Luis
Caixa Postal 107
MAPUTO
Tel: 27091, 27114
Telex: 6-203 NEGON Mo
Cable: NEGON
A,E,M,P

NETHERLANDS
Hewlett-Packard Nederland B.V.
Van Heuven Goedhartlaan 121
NL1181KK **AMSTELVEEN**
Tel: (20) 47-20-21
Telex: 13 216
A,CM,CP,E,MP,P

Hewlett-Packard Nederland B.V.
Bongerd 2
NL2906 VK **CAPELLE** A/D Ijssel
Tel: (10) 51-64-44
Telex: 21261 hepac nl
A,CM,CP

NEW ZEALAND
Hewlett-Packard (N.Z.) Ltd.
169 Manukau Road
P.O. Box 26-189
Epsom, **AUCKLAND**
Tel: 68-7159
Cable: HEWPACK Auckland
CM,CS,E,P*

Northrop Instruments & Systems
   Ltd.
Eden House, 44 Khyber Pass Road
P.O. Box 9682
Newmarket, **AUCKLAND**
Tel: 794-091
A,M

Northrop Instruments & Systems
   Ltd.
Terrace House, 4 Oxford Terrace
P.O. Box 8388
CHRISTCHURCH
Tel: 64-165
A,M

Hewlett-Packard (N.Z.) Ltd.
4-12 Cruickshank Street
P.O. Box 9443
Kilbirnie, **WELLINGTON 3**
Tel: 877-199
Cable: HEWPACK Wellington
CM,CP,E,P

Northrop Instruments & Systems
   Ltd.
Sturdee House
85-87 Ghuznee Street
P.O. Box 2406
WELLINGTON
Tel: 850-091
Telex: NZ 31296
A,M

NIGERIA
The Electronics Instrumentations
   l td.
N6B/770 Oyo Road
Oluseun House
P.M.B. 5402
IBADAN
Tel: 461577
Telex: 31231 TEIL NG
A,E,M,P

The Electronics Instrumentations
   Ltd.
144 Agege Motor Road, Mushin
P.O. Box 6645
Mushin, **LAGOS**
A,E,M,P

NORTHERN IRELAND
Cardiac Services Company
95A Finaghy Road South
BELFAST BT 10 OBY
Tel: (0232) 625-566
Telex: 747626
M

NORWAY
Hewlett-Packard Norge A/S
Nygaardsgaten 114
P.O. Box 4210
N-5013 Nygaardsgaten, **BERGEN**
Tel: (05) 21-97-33
Telex: 16621 hpnas n
CM,CS,E

Hewlett-Packard Norge A/S
Oestendalen 18
P.O. Box 34
N-1345 **OESTERAAS**
Tel: (02) 17-11-80
Telex: 16621 hpnas n
A*,CM,CP,E,MS,P

OMAN
Khimjl Ramdas
P.O. Box 19
MUSCAT
Tel: 72-22-17, 72-22-25
Telex: 3289 BROKER MB MUSCAT
P

PAKISTAN
Mushko & Company Ltd.
10, Bazar Road
Sector G-6/4
ISLAMABAD
Tel: 28624
Cable: FEMUS Rawalpindi
A,E,M,P

Mushko & Company Ltd.
Oosman Chambers
Abdullah Haroon Road
KARACHI 0302
Tel: 511027, 512927
Telex: 2894 MUSHKO PK
Cable: COOPERATOR Karachi
A,E,M,P*

SALES & SUPPORT OFFICES
Arranged alphabetically by country

PANAMA
Electronico Balboa, S.A.
Apartado 4929
Panama 5
Calle Samuel Lewis
Edificio "Alfa" No. 2
CIUDAD DE PANAMA
Tel: 64-2700
Telex: 3480380
Cable: ELECTRON Panama
A,E,M,P

Foto Internacional, S.A.
P.O. Box 2068
Free Zone of Colon
COLON 3
Tel: 45-2333
Telex: 3485126
Cable: IMPORT COLON/Panama
P

PERU
Compania Electro Medica S.A.
Los Flamencos 145, San Isidro
Casilla 1030
LIMA I
Tel: 41-4325
Telex: Pub. Booth 25424 SISIDRO
Cable: ELMED Lima
A,E,M,P

PHILIPPINES
The Online Advanced Systems
    Corporation
Rico House, Amorsolo Cor. Herrera
    Street
Legaspi Village, Makati
P.O. Box 1510
Metro **MANILA**
Tel: 85-35-81, 85-34-91, 85-32-21
Telex: 3274 ONLINE
A,C,E,M

Electronic Specialists and
    Proponents Inc.
690-B Epifanio de los Santos
    Avenue
Cubao, **QUEZON CITY**
P.O. Box 2649 Manila
Tel: 98-96-81, 98-96-82, 98-96-83
Telex: 742-40287
Cable: ESPINC MANILA
P

POLAND
Buro Informacjii Technicznej
Hewlett-Packard
Ul Stawki 2, 6P
PL00-950 **WARSZAWA**
Tel: 39-59-62, 39-67-43
Telex: 81 24 53
A,C*,E*,M*,P*

PORTUGAL
Telectra-Empresa Tecnica de
    Equipmentos Electricos S.a.r.l.
Rua Rodrigo da Fonseca 103
P.O. Box 2531
P-LISBON 1
Tel: (19) 68-60-72
Telex: 12598
A,C,E,P
Mundinter
Intercambio Mundial de Comercio
    S.a.r.l
P.O. Box 2761
Avenida Antonio Augusto de Aguiar
    138
P-LISBON
Tel: (19) 53-21-31, 53-21-37
Telex: 16691 munter p
M

PUERTO RICO
Hewlett-Packard Puerto Rico
Calle 272
#203 Urb. Country Club
RIO PIEDRAS, Puerto Rico 00924
Tel: (809) 762-7255
Telex: 345 0514
A,CP

QATAR
Business Communications Quatar
P.O. Box 3656
DOHA
Tel: 5851
Telex: 4454
P
Nasser Trading & Contracting
P.O. Box 1563
DOHA
Tel: 22170
Telex: 4439 NASSER
M

RHODESIA
Field Technical Sales
45 Kelvin Road North
P.O. Box 3548
SALISBURY
Tel: 705231
Telex: RH 4122
A,E,M,P

ROMANIA
Hewlett-Packard Reprezentanta
Boulevard Nicolae Balcescu 16
BUCURESTI
Tel: 130725
Telex: 10440
C*,E*

SAUDI ARABIA
Modern Electronic Establishment
P.O. Box 193
AL-KHOBAR
Tel: 44-678, 44-813
Telex: 670136
Cable: ELECTA AL-KHOBAR
C,E,M,P
Modern Electronic Establishment
P.O. Box 1228, Baghdadiah Street
JEDDAH
Tel: 27-798
Telex: 401035
Cable: ELECTA JEDDAH
C,E,M,P
Modern Electronic Establishment
P.O. Box 2728
RIYADH
Tel: 62-596, 66-232
Telex: 202049
C,E,M,P

SCOTLAND
Hewlett-Packard Ltd.
Royal Bank Buildings
Swan Street
BRECHIN, Angus, Scotland
Tel: 3101, 3102
CM,CS
Hewlett-Packard Ltd.
SOUTH QUEENSFERRY
West Lothian, EH30 9TG
GB-Scotland
Tel: (031) 3311000
Telex: 72682
CM,CP,E,MS

SINGAPORE
Hewlett-Packard Singapore (Pte.)
    Ltd.
6th Floor, Inchcape House
450-452 Alexandra Road
SINGAPORE 0511
P. O. Box 58 Alexandra Post Office
Singapore 9115
Tel: 631788
Telex: HPSGSO RS 32409
Cable: HEWPACK, Singapore
A,CP,E,MS,P

SOUTH AFRICA
Hewlett-Packard South Africa (Pty.)
    Ltd.
Pine Park Center
Forest Drive, Pinelands
CAPE PROVINCE, 7405
P.O. Box 120
Howard Place
CAPE PROVINCE 7450
Tel: 53-7955, 53-7956, 53-7957,
    53-7958, 53-7959
Telex: 57-0006
A,CM,CS,E,MS,P
Hewlett-Packard South Africa (Pty.)
    Ltd.
P.O. Box 37066
Overport
DURBAN 4067
Tel: 28-4178, 28-4179, 28-4110
CM,CS
Hewlett-Packard South Africa (Pty.)
    Ltd.
Hewlett-Packard Centre
Daphne Street
Private Bag Wendywood
SANDTON 2144
Tel: 802-5111
Telex: 84782
Cable: HEWPACK Johannesburg
A,CM,CP,E,MS,P

SPAIN
Hewlett-Packard Espanola S.A.
c/Entenza 312
E-BARCELONA 29
Tel: (3) 322-24-51, 321-73-54
Telex: 52603 hpbe e
A,CM,CP,E,MS,P
Hewlett-Packard Espanola S.A.
c/San Vicente s/n
Edificio Albia II, 7°B
E-BILBAO 1
Tel: 423-82-06, 423-83-06
A,CM,E,MS
Hewlett-Packard Espanola S.A.
Calle Jerez 3
E-MADRID 16
Tel: (1) 458-2600
Telex: 23515 hpe
A,CM,E,MP,P
Hewlett-Packard Espanola S.A.
Colonia Mirasierra
Edificio Juban
c/o Costa Brava 13
E-MADRID 34
Tel: (1) 734-8061, 734-1162
CM,CP
Hewlett-Packard Espanola S.A.
Av Ramon y Cajal 1
Edificio Sevilla 1, Planta 9
E-SEVILLA 5
Tel: (954) 64-44-54, 64-44-58
A,CM,CS,MS,P
Hewlett-Packard Espanola S.A.
C/Ramon Gordillo 1 (Entlo.)
E-VALENCIA 10
Tel: (96) 361-1354
CM,CS,P

SRI LANKA
Metropolitan Agencies Ltd.
209/9 Union Place
COLOMBO 2
Tel: 35947
Telex:1377METROLTD CE
Cable: METROLTD
A,E,M,P

SUDAN
Radison Trade
P.O. Box 921
KHARTOUM
Tel: 44048
Telex: 375
A,E,M

SURINAM
Surtel Radio Holland N.V.
Grote Hofstr. 3-5
P.O. Box 155
PARAMARIBO
Tel: 72118, 77880
Cable: Surtel
E,M

SWEDEN
Hewlett-Packard Sverige AB
Enighetsvagen 3
S-16120 **BROMMA**
Tel: (08) 730-0550
Telex: (854) 10721 MESSAGES
Cable: MEASUREMENTS
A,CM,CP,E,MS,P
Hewlett-Packard Sverige AB
Sunnanvagen 14K
S-22226 **LUND**
Tel: (46) 13-69-79
CM,CS
Hewlett-Packard Sverige AB
Vastra Vintergatan 9
S-70344 **OREBRO**
Tel: (019) 14-07-20
CM,CS
Hewlett-Packard Sverige AB
Frotallisgatan 30
S-42132 **VASTRA-FROLUNDA**
Tel: (031) 49-09-50
Telex: 85410721
CM,CS,E,P

SWITZERLAND
Hewlett-Packard (Schweiz) AG
Clarastrasse 12
CH-4058 **BASEL**
Tel: (061) 33-59-20
A,CM
Hewlett-Packard (Schweiz) AG
Bahnhoherweg 44
3018 **BERN**
Tel: (031) 56-24-22
CM
Hewlett-Packard (Schweiz) AG
47 Avenue Blanc
CH-1202 **GENEVA**
Tel: (022) 32-30-05, 32-48-00
CM,CP
Hewlett-Packard (Schweiz) AG
29 Chemin Chateau Bloc
CH-1219 **LE LIGNON**-Geneva
Tel: (022) 96-03-22
Telex: 27333 hpag ch
Cable: HEWPACKAG Geneva
A,CM,E,MS,P
Hewlett-Packard (Schweiz) AG
Zurcherstrasse 20
P.O. Box 307
CH-8952 **SCHLIEREN**-Zurich
Tel: (01) 730-5240, 730-1821
Telex: 53933 hpag ch
Cable: HPAG CH
A,CM,CP,E,MS,P

SYRIA
General Electronic Inc.
Nuri Basha-Ahnaf Ebn Kays Street
P.O. Box 5781
DAMASCUS
Tel: 33-24-87
Telex: 11215 ITIKAL
Cable: ELECTROBOR DAMASCUS
E
Sawah & Co.
Place Azme
Boite Postale 2308
DAMASCUS
Tel: 16-367, 19-697, 14-268
Telex: 11304 SATACO SY
Cable: SAWAH, DAMASCUS
M
Suleiman Hilal El Mlawi
P.O. Box 2528
Mamoun Bitar Street, 56-58
DAMASCUS
Tel: 11-46-63
Telex: 11270
Cable: HILAL DAMASCUS
M

TAIWAN
Hewlett-Packard Far East Ltd.
Kaohsiung Branch
68-2, Chung Cheng 3rd Road
Shin Shin, Chu
KAOHSIUNG
Tel: 241-2318, 261-3253
CS,E,MS,P
Hewlett-Packard Far East Ltd.
Taiwan Branch
Bank Tower, 5th Floor
205 Tun Hwa North Road
TAIPEI
Tel: 751-0404
Cable:HEWPACK Taipei
A*,CP,E,MS,P
San Kwang Instruments Co., Ltd.
20 Yung Sui Road
TAIPEI
Tel: 361-5446, 361-5447,
    361-5448, 361-5449
Telex: 22894 SANKWANG
Cable: SANKWANG Taipei
A

TANZANIA
International Aeradio (E.A.) Ltd.
P.O. Box 861
DAR ES SALAAM
Tel: 21251
Telex: 41030
M

THAILAND
UNIMESA Co. Ltd.
Elcom Research Building
2538 Sukhumvit Ave.
Bangchak, **BANGKOK**
Tel: 393-2387, 393-0338
Telex: TH81160, 82938, 81038
Cable: UNIMESA Bangkok
A,E,M
Bangkok Business Equipment Ltd.
5/5-6 Dejo Road
BANGKOK
Tel: 234-8670, 234-8671,
    234-8672, 234-8673
Cable: BUSIQUIPT Bangkok
P

TRINIDAD & TOBAGO
CARTEL
Caribbean Telecoms Ltd.
P.O. Box 732
50/A Jerningham Avenue
PORT-OF-SPAIN
Tel: 62 4214, 62 4213
A,E,M,P

TUNISIA
Tunisie Electronique
31 Avenue de la Liberte
TUNIS
Tel: 280-144
E,P

Corema
1 ter. Av. de Carthage
TUNIS
Tel: 253-821
Telex: 12319 CABAM TN
M

TURKEY
Teknim Company Ltd.
Riza Sah Pehlievi
Caddesi No. 7
Kavaklidere, ANKARA
Tel: 275800
Telex: 42155
E

UNITED ARAB EMIRATES
Emitac Ltd.
P.O. Box 2711
ABU DHABI
Tel: 331370, 331371
E,M,P
Emitac Ltd.
P.O. Box 1641
SHARJAH
Tel: 354121, 354123
Telex: 68136
E,M,P

**UNITED KINGDOM
see: GREAT BRITAIN
NORTHERN IRELAND
SCOTLAND**

**UNITED STATES OF
AMERICA**

Alabama
Hewlett-Packard Co.
700 Century Park South
Suite 128
BIRMINGHAM, AL 35226
Tel: (205) 822-6802
CM,CS,MP

Hewlett-Packard Co.
P.O. Box 4207
8290 Whitesburg Drive, S.E.
HUNTSVILLE, AL 35802
Tel: (205) 881-4591
CM,CP,E,M*

Alaska
Hewlett-Packard Co.
1577 "C" Street, Suite 252
ANCHORAGE, AK 99510
Tel: (206) 454-3971
CM,CS**

Arizona
Hewlett-Packard Co.
2336 East Magnolia Street
PHOENIX, AZ 85034
Tel: (602) 273-8000
A,CM,CP,E,MS

Hewlett-Packard Co.
2424 East Aragon Road
TUCSON, AZ 85706
Tel: (602) 889-4661
CM,CS,E,MS**

Arkansas
Hewlett-Packard Co.
P.O. Box 5646
Brady Station
LITTLE ROCK, AR 72215
Tel: (501) 376-1844
CM,MS

California
Hewlett-Packard Co.
7621 Canoga Avenue
CANOGA PARK, CA 91304
Tel: (213) 702-8300
A,CM,CP,E,P

Hewlett-Packard Co.
1579 W. Shaw Avenue
FRESNO, CA 93771
Tel: (209) 224-0582
CM,MS

Hewlett-Packard Co.
1430 East Orangethorpe
FULLERTON, CA 92631
Tel: (714) 870-1000
CM,CP,E,MP

Hewlett-Packard Co.
5400 W. Rosecrans Boulevard
LOS ANGELES, CA 90260
Tel: (213) 970-7500
CM,CP,MP

Hewlett-Packard Co.
3939 Lankersham Blvd.
NORTH HOLLYWOOD, CA 91604
Tel: (213) 877-1282
regional headquarters

Hewlett-Packard Co.
3200 Hillview Avenue
PALO ALTO, CA 94304
Tel: (415) 857-8000
CM,CP,E

Hewlett-Packard Co.
646 W. North Market Boulevard
SACRAMENTO, CA 95834
Tel: (916) 929-7222
A*,CM,CS,E,MS

Hewlett-Packard Co.
9606 Aero Drive
P.O. Box 23333
SAN DIEGO, CA 92123
Tel: (714) 279-3200
CM,CP,E,MP

Hewlett-Packard Co.
363 Brookhollow Drive
SANTA ANA, CA 92705
Tel: (714) 641-0977
A,CM,C*,E

Hewlett-Packard Co.
3003 Scott Boulevard
SANTA CLARA, CA 95050
Tel: (408) 988-7000
A,CM,CP,E,MP

Hewlett-Packard Co.
454 Carlton Court
SO. SAN FRANCISCO, CA 94080
Tel: (415) 877-0772
CM,CP

Colorado
Hewlett-Packard Co.
5600 DTC Parkway
ENGLEWOOD, CO 80110
Tel: (303) 771-3455
A,CM,CP,E,MS

Connecticut
Hewlett-Packard Co.
47 Barnes Industrial Road South
P.O. Box 5007
WALLINGFORD, CT 06492
Tel: (203) 265-7801
A,CM,CP,E,MS

Florida
Hewlett-Packard Co.
P.O. Box 24210
2727 N.W. 62nd Street
FORT LAUDERDALE, FL 33309
Tel: (305) 973-2600
CM,CP,E,MP

Hewlett-Packard Co.
4080 Woodcock Drive, #132
Brownett Building
JACKSONVILLE, FL 32207
Tel: (904) 398-0663
CM,C*,E*,MS**

Hewlett-Packard Co.
P.O. Box 13910
6177 Lake Ellenor Drive
ORLANDO, FL 32809
Tel: (305) 859-2900
A,CM,CP,E,MS

Hewlett-Packard Co.
P.O. Box 12826
Suite 5, Building 1
Office Park North
PENSACOLA, FL 32575
Tel: (904) 476-8422
A,CM,MS

Hewlett-Packard Co.
110 South Hoover, Suite 120
TAMPA, FL 33609
Tel: (813) 872-0900
A*,CM,CS,E*,M*

Georgia
Hewlett-Packard Co.
P.O. Box I05005
450 Interstate N. Parkway
ATLANTA, GA 30339
Tel: (404) 955-1500
Telex: 810-766-4890
A,CM,CP,E,MP

Hewlett-Packard Co.
Executive Park Suite 306
P.O. Box 816
AUGUSTA, GA 30903
Tel: (404) 736-0592
CM,MS

Hewlett-Packard Co.
P.O. Box 2103
1172 N. Davis Drive
WARNER ROBINS, GA 31098
Tel: (912) 922-0449
CM,E

Hawaii
Hewlett-Packard Co.
2875 South King Street
HONOLULU, HI 96826
Tel: (808) 955-4455
A,CM,CS,E,MS

Idaho
Hewlett-Packard Co.
11311 Chinden Boulevard
BOISE, ID 83707
Tel: (208) 376-6000
CM,CS,M*

Illinois
Hewlett-Packard Co.
211 Prospect Road
BLOOMINGTON, IL 61701
Tel: (309) 663-0383
CM,CS,MS**

Hewlett-Packard Co.
1100 31st Street
DOWNERS GROVE, IL 60515
Tel: (312) 960-5760
CM,CP

Hewlett-Packard Co.
5201 Tollview Drive
ROLLING MEADOWS, IL 60008
Tel: (312) 255-9800
A,CM,CP,E,MP

Indiana
Hewlett-Packard Co.
P.O. Box 50807
7301 No. Shadeland Avenue
INDIANAPOLIS, IN 46250
Tel: (317) 842-1000
A,CM,CS,E,MS

Iowa
Hewlett-Packard Co.
5815 S.W. 5th Street
DES MOINES, IA 50315
Tel: (515) 243-5876
CM,MS**

Hewlett-Packard Co.
2415 Heinz Road
IOWA CITY, IA 52240
Tel: (319) 351-1020
CM,CS,E*,MS

Kansas
Hewlett-Packard Co.
514 South Westview
DERBY, KA 67037
Tel: (316) 265-5200
CM,CS

Kentucky
Hewlett-Packard Co.
10170 Linn Station Rd., Suite 525
Atkinson Square
LOUISVILLE, KY 40223
Tel: (502) 426-0100
A,CM,CS,MS

Louisiana
Hewlett-Packard Co.
P.O. Box 1449
3229 Williams Boulevard
KENNER, LA 70062
Tel: (504) 443-6201
A,CM,CS,E,MS

Maryland
Hewlett-Packard Co.
7121 Standard Drive
HANOVER, MD 21076
Tel: (301) 796-7700
A,CM,CP,E,MS

Hewlett-Packard Co.
2 Choke Cherry Road
ROCKVILLE, MD 20850
Tel: (301) 948-6370
A,CM,CP,E,MP

Massachusetts
Hewlett-Packard Co.
32 Hartwell Avenue
LEXINGTON, MA 02173
Tel: (617) 861-8960
A,CM,CP,E,MP

Michigan
Hewlett-Packard Co.
23855 Research Drive
FARMINGTON HILLS, MI 48024
Tel: (313) 476-6400
A,CM,CP,E,MP

Hewlett-Packard Co.
4326 Cascade Road S.E.
GRAND RAPIDS, MI 49506
Tel: (616) 957-1970
CM,CS,MS

Minnesota
Hewlett-Packard Co.
2025 W. Larpenteur Ave.
ST. PAUL, MN 55113
Tel: (612) 644-1100
A,CM,CP,E,MP

Mississippi
Hewlett-Packard Co.
322 N. Mart Plaza
JACKSON, MS 39206
Tel: (601) 982-9363
CM,MS

Missouri
Hewlett-Packard Co.
11131 Colorado
KANSAS CITY, MO 64137
Tel: (816) 763-8000
Telex: 910-771-2087
A,CM,CS,E,MS

Hewlett-Packard Co.
1024 Executive Parkway
ST. LOUIS, MO 63141
Tel: (314) 878-0200
A,CM,CP,E,MP

Nebraska
Hewlett-Packard
Suite 101
7101 Mercy Road
OMAHA, NE 68106
Tel: (402) 392-0948
CM,MS

Nevada
Hewlett-Packard Co.
Suite D-130
5030 Paradise Blvd.
LAS VEGAS, NV 89119
Tel: (702) 736-6610
CM,MS**

New Jersey
Hewlett-Packard Co.
W120 Century Road
PARAMUS, NJ 07652
Tel: (201) 265-5000
A,CM,CP,E,MP

Hewlett-Packard Co.
60 New England West
PISCATAWAY, NJ 08854
Tel: (201) 981-1199
A,CM,CP,E

New Mexico
Hewlett-Packard Co.
P.O. Box 11634
Station E
11300 Lomas Blvd.,N.E.
ALBUQUERQUE, NM 87192
Tel: (505) 292-1330
Telex: 910-989-1185
CM,CP,E,MS

Hewlett-Packard Co.
156 Wyatt Drive
LAS CRUCES, NM 88001
Tel: (505) 526-2484
Telex: 910-9983-0550
CM,C*,E,M*

New York
Hewlett-Packard Co.
6 Automation Lane
Computer Park
ALBANY, NY 12205
Tel: (518) 458-1550
Telex: 710-444-4691
A,CM,CS,E,MS

Hewlett-Packard Co.
650 Perinton Hill Office Park
FAIRPORT, NY 14450
Tel: (716) 223-9950
Telex: 510-253-0092
CM,CP,E,MS

Hewlett-Packard Co.
No. 1 Pennsylvania Plaza
55th Floor
34th Street & 8th Avenue
NEW YORK, NY 10001
Tel: (212) 971-0800
CM,CP,E*,M*

Hewlett-Packard Co.
5858 East Molloy Road
SYRACUSE NY 13211
Tel: (315) 455-2486
A,CM,CS,E,MS

Hewlett-Packard Co.
3 Crossways Park West
WOODBURY, NY 11797
Tel: (516) 921-0300
Telex: 510-221-2183
A,CM,CP,E,MS

North Carolina
Hewlett-Packard Co.
5605 Roanne Way
GREENSBORO, NC 27409
Tel: (919) 852-1800
A,CM,CP,E,MS

Ohio
Hewlett-Packard Co.
9920 Carver Road
CINCINNATI, OH 45242
Tel: (513) 891-9870
CM,CP,MS

Hewlett-Packard Co.
16500 Sprague Road
CLEVELAND, OH 44130
Tel: (216) 243-7300
Telex: 810-423-9430
A,CM,CP,E,MS

Hewlett-Packard Co.
962 Crupper Ave.
COLUMBUS, OH 43229
Tel: (614) 432-1041
CM,CP,E*

Hewlett-Packard Co.
330 Progress Rd.
DAYTON, OH 45449
Tel: (513) 859-8202
A,CM,CP,E*,MS

Oklahoma
Hewlett-Packard Co.
P.O. Box 32008
6301 N. Meridan Avenue
OKLAHOMA CITY, OK 73122
Tel: (405) 721-0200
A*,CM,CP,E*,MS

Hewlett-Packard Co.
Suite 121
9920 E. 42nd Street
TULSA, OK 74145
Tel: (918) 665-3300
A**,CM,CS,M*

Oregon
Hewlett-Packard Co.
9255 Pioneer Court
WILSONVILLE, OR 97070
Tel: (503) 682-8000
A,CM,CP,E*,MS

Pennsylvania
Hewlett-Packard Co.
Crystal Brook Professional Building
Route 35
EATONTOWN, PA 07724
Tel: (201) 542-1384
A*,CM,C*,E*,P*

Hewlett-Packard Co.
1021 8th Avenue
King of Prussia Industrial Park
KING OF PRUSSIA, PA 19406
Tel: (215) 265-7000
Telex: 510-660-2670
A,CM,CP,E,MP

Hewlett-Packard Co.
111 Zeta Drive
PITTSBURGH, PA 15238
Tel: (412) 782-0400
A,CM,CP,E,MP

South Carolina
Hewlett-Packard Co.
P.O. Box 6442
6941-1 N. Trenholm Road
COLUMBIA, SC 29260
Tel: (803) 782-6493
CM,CS,E,MS

Tennessee
Hewlett-Packard Co.
8906 Kingston Pike
KNOXVILLE, TN 37919
Tel: (615) 691-2371
A*,CM,MS

Hewlett-Packard Co.
3070 Directors Row
Directors Square
MEMPHIS, TN 38131
Tel: (901) 346-8370
A,CM,CS,MS

Hewlett-Packard Co.
Suite 103
478 Craighead Street
NASHVILLE, TN 37204
Tel: (615) 383-9130
CM,MS**

Texas
Hewlett-Packard Co.
Suite 310W
7800 Shoalcreek Blvd.
AUSTIN, TX 78757
Tel: (512) 459-3143
CM,E

Hewlett-Packard Co.
Suite C110
4171 North Mesa
EL PASO, TX 79902
Tel: (915) 533-3555
CM,CS,E*,MS**

Hewlett-Packard Co.
5020 Mark IV Parkway
FORT WORTH, TX 76106
Tel: (817) 625-6361
CM,C*

Hewlett-Packard Co.
10535 Harwin Street
HOUSTON, TX 77036
Tel: (713) 776-6400
A,CM,CP,E,MP

Hewlett-Packard Co.
P.O. Box 1270
930 E. Campbell Rd.
RICHARDSON, TX 75081
Tel: (214) 231-6101
A,CM,CP,E,MP

Hewlett-Packard Co.
205 Billy Mitchell Road
SAN ANTONIO, TX 78226
Tel: (512) 434-8241
CM,CS,E,MS

Utah
Hewlett-Packard Co.
3550 W. 2100 South Street
SALT LAKE CITY, UT 84119
Tel: (801) 974-1700
A,CM,CP,E,MS

Virginia
Hewlett-Packard Co.
P.O. Box 9669
2914 Hungary Spring Road
RICHMOND, VA 23228
Tel: (804) 285-3431
A,CM,CP,E,MS

Hewlett-Packard Co.
3110 Peters Creek Road, N.W.
ROANOKE, VA 24015
Tel: (703) 922-7000
CM,CS,E**

Hewlett-Packard Co.
5700 Thurston Avenue
VIRGINIA BEACH, VA 23455
Tel: (804) 460-2471
CM,CS,MS

Washington
Hewlett-Packard Co.
Bellefield Office Park
1203 114th Ave. S.E
BELLEVUE, WA 98004
Tel: (206) 454-3971
A,CM,CP,E,MP

Hewlett-Packard Co.
Suite A-1
708 North Argonne Road
SPOKANE, WA 99206
Tel: (509) 535-0864
CM,CS

West Virginia
Hewlett-Packard Co.
4604 MacCorkle Ave., S.E.
CHARLESTON, WV 25304
Tel: (304) 925-0492
A,CM,MS

Wisconsin
Hewlett-Packard Co.
150 S. Sunny Slope Road
BROOKFIELD, WI 53005
Tel: (414) 784-8800
A,CM,CS,E*,MP

URUGUAY
Pablo Ferrando S.A.C. e.I.
Avenida Italia 2877
Casilla de Correo 370
MONTEVIDEO
Tel: 40-3102
Telex: 901 Public Booth Para Pablo
Ferrando 919520
Cable: RADIUM Montevideo
A,E,M

Guillermo Kraft del Uruguay S.A.
Avda. Libertador Brig. Gral.
Lavalleja 2083
MONTEVIDEO
Tel: 23 45 88, 23 48 08, 20 88 30
P

VENEZUELA
Hewlett-Packard de Venezuela C.A.
P. O. Box 50933
3a Transversal Los Ruices Norte
Edificio Segre 2Y3
CARACAS 1071
Tel: 239-4133
Telex: 25146 HEWPACK
Cable: HEWPACK Caracas
A,CP,E,MS,P

YUGOSLAVIA
Iskra Commerce, n. sol. o.
Zastopstvo Hewlett-Packard
Obilicev Venac 26
YU 11000 BEOGRAD
Tel: 636955
Telex: 11530
C,E*,M*,P**

Iskra Commerce, n. sol. o.
Zastopstvo Hewlett-Packard
Miklosiceva 38/VII
YU-61000 LJUBLJANA
Tel: 321674, 315870
Telex: 31300
C,E*,M*,P**

ZAMBIA
R.J. Tilbury (Zambia) Ltd.
P.O. Box 2792
LUSAKA
Tel: 81243
A,E,M,P

**FOR COUNTRIES AND
AREAS NOT LISTED:**

CANADA

Ontario
Hewlett-Packard (Canada) Ltd.
6877 Goreway Drive
MISSISSAUGA, Ontario L4V IM8
Tel: (416) 678-9430
Telex: 610-492-4246

EASTERN USA

New Jersey
Hewlett-Packard Co.
W120 Century Road
PARAMUS, NJ 07652
Tel: (201) 265-5000

MIDWESTERN USA

Illinois
Hewlett-Packard Co.
5201 Tollview Drive
ROLLING MEADOWS, IL 60008
Tel: (312) 255-9800

SOUTHERN USA

Georgia
Hewlett-Packard Co.
P.O. Box 105005
450 Interstate N. Parkway
ATLANTA, GA 30339
Tel: (404) 955-1500
Telex: 810-766-4890

WESTERN USA

California
Hewlett-Packard Co.
3939 Lankersham Blvd.
NORTH HOLLYWOOD, CA 91604
Tel: (213) 877-1282

EAST EUROPEAN AREAS

AUSTRIA
Hewlett-Packard Ges.m.b.H.
Wehlistrasse 29
P.O. Box 7
A-1205 VIENNA, Austria
Tel: (222) 35-16-20
Telex: 135823/135066

EUROPEAN AREAS

SWITZERLAND
Hewlett-Packard S.A.
7 Rue du Bois-du-Lan
CH-1217 MEYRIN 2, Switzerland
Tel: (022) 98-96-51
Telex: 27835 hpse
Cable: HEWPACKSA Geneve

**MEDITERRANEAN AND
MIDDLE EAST AREAS**

GREECE
Hewlett-Packard S.A.
Mediterranean & Middle East
Operations
35 Kolokatroni St.
Platia Kefallariou
GR-Kifissia, ATHENS, Greece
Tel: 808-0359, 808-0429
Telex: 21-6588
Cable: HEWPACKSA Athens

OTHER AREAS
Hewlett-Packard Co.
ICON Headquarters
3495 Deer Creek Road
PALO ALTO, CA 94304
U.S.A.
Tel: (415) 857-2824
TWX: 910-373-1267
Telex: 034-8300; 034-8493
Cable: HEWPACK

**HEWLETT
PACKARD**