BASIC/3000 Compiler

Reference Manual

HEWLETT-PACKARD COMPANY
5303 STEVENS CREEK BLVD., SANTA CLARA, CALIFORNIA 95050

PART NO. 32103-90001
PRODUCT NO. 32103A

Printed in U.S.A. 11/74

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

LIST OF EFFECTIVE PAGES

The List of Effective Pages gives the most recent date on which the technical material on any given page was altered. If a page is simply re-arranged due to a technical change on a previous page, it is not listed as a changed page. Within the manual, changes are marked with a vertical bar in the margin.

Pages	Effective Date
Title	Jun 1976
ii to v	Jun 1976
vii	Nov 1974
1-0 to 1-2	Nov 1974
2-1 to 2-5	Nov 1974
3-1 to 3-6	Nov 1974
4-1 to 4-6	Nov 1974

PRINTING HISTORY

New editions incorporate all update material since the previous edition. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The date on the title page and back cover changes only when a new edition is published. If minor corrections and updates are incorporated, the manual is reprinted but neither the date on the title page and back cover nor the edition change.

First Edition. . . . . Nov 1974
Update package #1 . . . . . Jun 1976
Update #1 Incorporated . . . . . Sep 1977

CONTENTS

ILLUSTRATIONS

TABLES

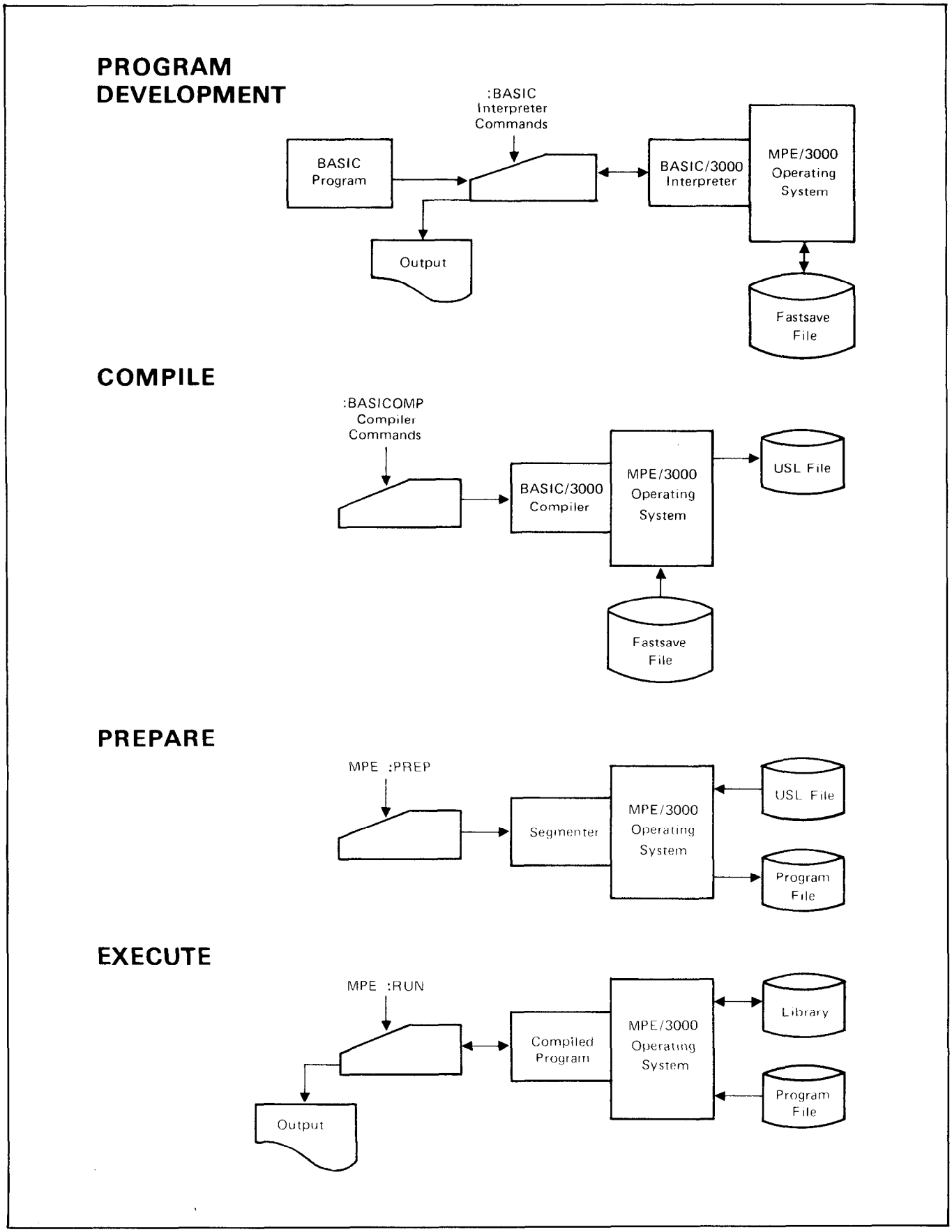


Figure 1-1. Overview of Operations

INTRODUCING THE BASIC/3000 COMPILER

SECTION I

1-1. PURPOSE

The HP 32103A BASIC/3000 Compiler provides a means of converting your BASIC programs, which you have previously debugged and stored via the BASIC/3000 Interpreter, into compiled form. This means that your BASIC programs will exist in the system as actual code segments, rather than as data in a data file. The result is that the programs may be run under the operating system, rather than through line-by-line interpreting via the Interpreter subsystem, with a consequent reduction of execution time.

Typically, compiled BASIC programs will execute many times faster than when run interpretively. Of course, it should be understood that actual speed improvement depends on the type of program and the resource demands on the MPE/3000 operating system. CPU bound programs, for example, will realize a speed improvement on the order of 10 to 30 times faster, with certain extraordinary routines running up to 100 times faster. I/O bound programs will run 1 to 4 times faster.

1-2. OVERVIEW OF OPERATION

Figure 1-1 illustrates the four major phases involved in compiling and using BASIC/3000 compiled programs. Detailed examples will be shown later, in Section III.

1-3. **PROGRAM DEVELOPMENT** The first phase is to write and debug BASIC/3000 programs in the usual way, via Interpreter commands and statements. The Interpreter, which operates as a subsystem under MPE/3000, constructs the interpretive version of your programs in files on disc. When you are satisfied that the programs execute properly in interpretive form, you then save the programs using the Interpreter's SAVE FAST command. The resulting file, called a BASIC fastsave file, is the only acceptable source input for the compiler.

1-4. **COMPILE.** The second phase is to compile the debugged interpretive programs which now reside in one or more BASIC fastsave files. The BASIC/3000 Compiler, operating as a subsystem under MPE/3000, compiles the programs into relocatable binary modules in a USL (User Subprogram Library) file when you give the COMPILE command.

1-5. **PREPARE.** The third phase is to prepare the USL file into runnable form. This is done, typically, by the MPE/3000 command :PREP. This command invokes the MPE/3000 Segmenter, which binds the relocatable binary modules into code segments in a program file.

1-6. **EXECUTE.** The fourth phase is to execute the program file, typically by the MPE/3000 command :RUN. During execution, reference will be made to subroutines in the Compiler Library, and also to a BASIC runtime library which exists in the system SL (Segmented Library).

Note: The "Prepare" and "Execute" phases can be accomplished automatically following compilation by use of the :BASICPREP and :BASICGO commands; see definitions in Section II.

1-7. LANGUAGE CONSIDERATIONS

There are a few semantic differences between interpretive and compiled BASIC that you should be aware of when writing BASIC programs that you intend to compile. These are described below.

1-8. INITIALIZING VARIABLES

Numeric variables are not preinitialized to the undefined number (1 in bit 0, followed by all zeros) unless CONTROL INIT is specified; see command definition in Section II. As in the Interpreter, the UND function returns true if variables have the undefined value. Strings will as usual be set to null by having their logical length set to zero.

1-9. NON-BASIC CALLS

The CALL statement may reference non-BASIC procedures in your USL file as well as those in a segmented library (SL).

1-10. NUMBER OF VARIABLES

The maximum number of program variables is limited to 182. Local variables within functions are limited to 120 per function (including any program variables referenced within the function). Common variables are limited to 239.

1-11. PROGRAM SIZE

The maximum code segment size on the HP 3000 Computer is 16K words. (K = 1024.) Normal system configurations limit the segment to 4K words. Since the compiled code generated is about 1 to 1-1/3 times the size of the interpretive program, the total size of the BASIC programs to be put into a single segment should be less than 3000 words (as determined with the Interpreter's LENGTH command). Programs larger than that should be modularized (using CHAIN and INVOKE) and placed into separate segments using the SEGMENT option of the Compiler's CONTROL command.

1-12. PROGRAM NAMES

Each subprogram is compiled as a Reclocatable Binary Module (RBM) in the USL. The name associated with that RBM is the filename of the BASIC fastsave program file, omitting any qualifying lockword and group and account names. Consequently, all BASIC programs to be compiled into the same USL file must have unique local filenames.

1-13. CHAIN AND INVOKE

You will have to keep track of any optional starting statement numbers, as these must be supplied to the compiler via the COMPILE command. Also, if you specify a program by means of a string expression (e.g., CHAIN A$) instead of by a literal string (e.g., CHAIN "PROGA"), you should keep track of all possible names that the expression can assume; these names should be supplied to the compiler via the ENTRY command.

1-14. COMMON BLOCKS

Even though your BASIC programs will exist in compiled form, there can be no correspondence between BASIC and FORTRAN COMMON areas. This is because the data structures are not compatible. FORTRAN uses fixed length strings; BASIC uses variable length strings. BASIC permits redimensioning of strings; FORTRAN does not.

1-15. FOR STATEMENT

If your program requires branching from one FOR-loop to another which uses the same FOR-NEXT variable name, you should be aware that the compiler will bind the NEXT with its static FOR (nearest preceding matching FOR). This is unlike the dynamic binding (remembers where it came from) provided by the Interpreter. See Figure 1-2 for comparison.

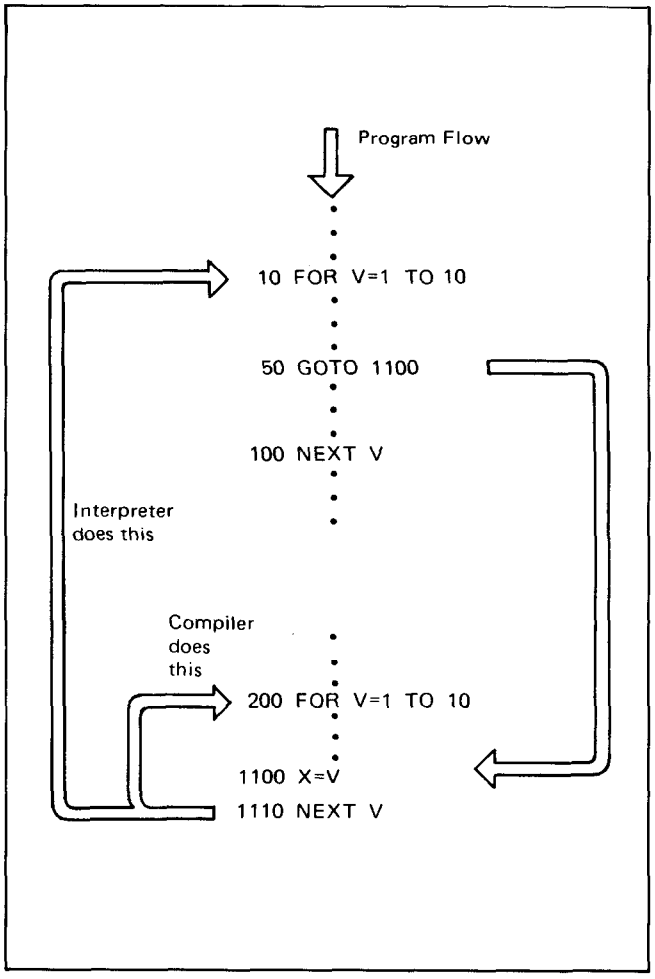


Figure 1-2. Branching into FOR Loop

1-16. ERROR CHECKING

It is assumed that any program which is to be compiled has been debugged previously. Thus there is little run-time error checking, which would slow down execution. Some of the interpretive error checking NOT done by the compiler:

a. subscript bounds checking

b. use of undefined variables

c. checking COMmon compatibilities

d. redimensioning past physical bounds.

1-17. NESTED BASIC

The use of a >BASIC command during an INPUT, LINPUT, or ENTER request is permitted only if the program is prepared with process-handling (PH) capability.

COMPILER COMMANDS

SECTION II

Associated with the BASIC/3000 Compiler are "system" commands and "subsystem" commands. You use the system commands, which are recognized by MPE/3000, to access the compiler subsystem. Then, once the compiler is accessed, you use the subsystem commands to direct compilation of your BASIC programs. After exiting from the compiler subsystem, the system commands control the preparation and execution of the compiled programs.

2-1. SYSTEM COMMANDS

Three of the system commands defined below invoke the BASIC/3000 Compiler. These are :BASICOMP, :BASICPREP, and :BASICGO. The remaining three (:RUN, :PREPRUN, and :PREP) are associated only with preparing and running compiled programs.

The attributes of compiler files are listed in Table 2-1. Run-time input and list file attributes are listed in Table 2-2.

2-2. COMPILE ONLY

The :BASICOMP command invokes the compiler for compilation only. That is, on exiting from the compiler subsystem, the compiled programs are left in a USL file; preparation and execution do not follow automatically. The :BASICOMP command is written as follows:

:BASICOMP [commandfile] [,[uslfile] [,listfile]]

None of the file designators is necessary as they all have defaults associated with them. Any one or two, or all three of them can be present. The file designators are defined as follows:

commandfile
the file from which all subsystem commands are read; default is $STDIN. (Formal file designator is BSCTEXT.)

uslfile
the USL file on which the compiled object program is written. If omitted, $OLDPASS is used if it is an existing USL file; if not, $NEWPASS is used. (Formal file designator is BSCUSL.)

listfile
the file which receives all compiler list output. If omitted, $STDLIST is used. (Formal file designator is BSCLIST.)

2-3. COMPILE AND PREPARE

The :BASICPREP command invokes the compiler for compilation and the Segmenter for preparation. That is, on exiting from the compiler subsystem, preparation of the USL file into an executable program file occurs automatically. (The USL file is left in $OLDPASS unless $NEWPASS was used for the program file.) The :BASICPREP command is written as follows:

:BASICPREP [commandfile] [,[progfile] [,listfile]]

The file designators commandfile and listfile are as defined above. The progfile designator is defined as follows:

progfile
the program file onto which the prepared program segments are to be written. If omitted, the default file $NEWPASS is assigned. (Formal file designator is BSCPROG.)

2-4. COMPILE, PREPARE, AND EXECUTE

The :BASICGO command invokes the compiler for compilation and the Segmenter for preparation, and initiates program execution. That is, on exiting from the compiler subsystem, preparation and execution of the program file occurs automatically. (The program file is left in $OLDPASS.) The :BASICGO command is written as follows:

:BASICGO [commandfile] [,listfile]

2-5. PREPARE AND EXECUTE ONLY

The :RUN command executes a compiled BASIC program, assuming that it exists as a prepared program file. The :RUN command (in simplified form for most BASIC applications) is written as follows:

:RUN progfile [,NOECHO] [;PARM=label]

where

progfile
the name of the program file that contains the prepared program.

NOECHO
secondary entry point; indicates that input is not to be echoed to the output file. This parameter is effective only when input is not echoed automatically by the input device.

Table 2-1. Attributes of Compiler Files

	Command File	List File	USL File
Formal Designator			
User Specified	BSCTEXT	BSCLIST	BSCUSL
Default	$STDIN	$STDLIST	$OLDPASS/$NEWPASS
File Options			
Domain	OLD	NEW	OLD first NEW if old fails
Type	ASCII	ASCII	Binary
Record Format	Undefined	Variable	Fixed
Others	*Disallow :FILE	*Disallow :FILE Carriage control	*Disallow :FILE
Access Options			
Access	Read only	Write only	Input/output
Restriction	Share	Exclusive	Exclusive
Record Size	Default	Default	128
Device	Default	Default	Default
File Size	Default	5000	Default
* (if not user specified)			

label
the statement number in the "main" BASIC program at which execution is to begin. This label must have been declared at compile time as an optional entry label; see COMPILE command.

Refer to the MPE/3000 reference manual if any of the other :RUN parameters is required.

The MPE/3000 commands :PREP and :PREPRUN may also be used for program preparation as described in the MPE/3000 reference manual. The optional NOECHO and PARAM= parameters are available following the uslfile parameter.

2-6. MAXDATA PARAMETER

If the executing BASIC program contains either FILES or COM, the MAXDATA parameter must be provided in the :PREP, :PREPRUN, or :RUN command. This is because the file buffers and common are dynamically allocated on the stack in the DL-to-DB space. MAXDATA must be specified large enough to accommodate the sum of the following:

a. File buffer space. This is the maximum sum of the lengths of the file buffers for all files open simultaneously. The length of a file buffer is the size of one record of the file.

b. Common space. This is the maximum sum of all COM areas active simultaneously.

c. Primary and secondary DB. 600 words maximum.

d. Program space. This is the amount of stack necessary for local variables. Summing of the space requirements for nested programs (INVOKE or multiline function reference) should be included. Program space can be taken from the STACK estimate on the label map.

Table 2-2. Attributes of Run-Time Input and List Files

	Input File	List File
Formal Designator		
User Specified	BASIN	BASLIST
Default	$STDINX	$STDLIST
File Options		
Domain	Old and Temporary	Old and Temporary
Type	ASCII	ASCII
Record Format	Variable	Variable
Others	Allow :FILE	Allow :FILE Carriage control
Access Options		
Access	Input only Non-multirecord access	Output only Non-multirecord access
Restriction	Default	Default
Record Size	-132	Default
Device	Default	Default
File Size	Default	1600

2-7. SUBSYSTEM COMMANDS

2-8. PROMPTS, COMMENTS, CONTINUATION

Examples:

```
$CONTROL<<CONTINUE COMMAND &
$ON NEXT LINE.>>SOURCE
```

The prompt character for compiler commands is the dollar sign ($). It may optionally be placed as the first character of each command in batch or session mode. Comments may be inserted before, after, or between command parameters. A comment is any string delimited by double less-than symbol at the start (<<) and double greater-than at the end (>>). To continue any command to the next line, use an ampersand (&) as the last character on the current line. Command records must be contained in columns 1 through 72.

2-9. CONTROL COMMAND

Example:

```
$CONTROL SOURCE,USLINIT,START=PROGA
```

The CONTROL command is used to set various compile options. It is not required if default or existing options are satisfactory for succeeding COMPILE commands. When used, you specify one or more of the following parameters, in any order, separated by commas. (Default conditions are underscored.)

SOURCE or NOSOURCE

The SOURCE parameter turns on the "list source" condition, requesting that the compiler list the programs to be compiled. Initially, the condition is off (no list). Once on, the condition remains on until turned off by NOSOURCE.

LABEL or NOLABEL

The LABEL parameter turns on the "label map" condition, requesting the compiler to print a label map showing the beginning octal code address for each statement. Initially, the condition is off (no map). Once on, the condition remains on until turned off by NOLABEL.

MAP or NOMAP

The MAP parameter turns on the "symbol table map" condition, requesting the compiler to print information regarding the structure, type, and octal address of each program variable for each program. Initially, the condition is off (no map). Once on, the condition remains on until turned off by NOMAP.

CODE or NOCODE

The CODE parameter turns on the "list code" condition, requesting the compiler to list (in octal) the object code generated for each program. Initially, the condition is off (no code list). Once on, the condition remains on until turned off by NOCODE.

LIST or NOLIST

The LIST parameter turns on the "list enable" condition, which must be on for the SOURCE, LABEL, MAP and CODE options to take effect. Initially, the condition is on (lists enabled). Once on, the condition remains on until turned off by NOLIST. When the condition is off, only error and warning messages and completion messages are sent to the listfile.

WARN or NOWARN

The WARN parameter turns on the "warning messages" condition, requesting the compiler to print warning messages about dubious though possibly acceptable language or command constructs. Initially, the condition is on (warnings printed). The condition remains on until turned off by NOWARN.

USLINIT

This parameter initializes the uslfile to "empty" status. This is necessary when you are compiling to an old USL file and you want to eliminate the current contents. This action is automatically performed by the compiler when the uslfile is currently empty or has invalid contents.

SEGMENT=segname

This parameter sets the segment name to any specified name acceptable to the MPE/3000 Segmenter (i.e., starts with alphabetic, up to 15 characters in length, and may contain alphanumeric and apostrophe characters). All subsequent programs compiled will be put in that segment until a new segment is specified by SEGMENT=. This action is necessary especially when program units being compiled into one segment approach the segment size limit. (The default segment is SEG'.)

START=progname

This parameter specifies which BASIC program (in a CHAIN/INVOKE-connected series) is the main program; i.e., which one is to receive initial control when prepared and run. The default starting program is the first program compiled. This parameter is not accepted when the "subprogram" condition is on; otherwise it can occur anywhere and the last occurence will hold.

SUBPROGRAM

This parameter turns on the "subprogram" condition, which inhibits the normally automatic generation of an outer block in the compiled code. Initially, the condition is off (outerblock automatically generated). Once turned on, this condition cannot be disabled. SUBPROGRAM is mutually exclusive with START and ENTRY during one invocation of the compiler.

LINES=number

This parameter sets the page size of the listfile to any specified number from 10 to 9999 lines per page. Default is 60. Whenever the next line sent to the listfile would overflow the page size, the compiler precedes it with a page eject, a page heading, and two blank lines.

INIT

This parameter inserts initializing code into the programs which will be compiled by the next COMPILE statement; this code initializes all local numeric simple variables and arrays to the "undefined" number (1 in bit 0, followed by all zeros). This value may be checked for with the UND function. INIT is enabled for only one COMPILE command that requires initialization code.

2-10. ENTRY COMMAND

Example:

```
$ENTRY SUBPROG1, SUBPROGA,A
```

The ENTRY command is used to inform the compiler of all possible program names that may be assumed by string

expressions in CHAIN or INVOKE statements in your main program and subprograms. Thus if there is an INVOKE A$ in a source program, all values which could be assumed by A$ should be specified in an ENTRY command; it is not necessary if CHAIN or INVOKE specifies a literal string (e.g., INVOKE "A"). Any number of ENTRY commands may be given; each one simply adds more names to the chain-invoke table. (A reference that is not in the table is treated as a library call. In such a case, your program executes a dynamic LOADPROC from the segmented libraries; this is costly in execution time.) The ENTRY command cannot be used with a SUBPROGRAM compilation.

2-11. TITLE COMMAND

Example:

```
$TITLE "PROGRAM ""C"" LISTING"
```

The TITLE command replaces the title portion (columns 29 through 132) of the standard page heading with the specified string or strings. The title begins in column 29 and is extended by blanks if less than 104 characters. If more than one string is specified, the separating commas, blanks, or comments are not included in the title, and the strings are concatenated. Two adjacent quotes can be used to include one quote within the string. If no parameter is present, the title portion will be blank.

2-12. COMPILE COMMAND

Example:

```
$COMPILE PROG1, PROG2 (100, 250)
```

The COMPILE command causes compilation of the specified BASIC fastsave programs to begin. If a source program has multiple entry points, each entry point (other than the starting statement) must be listed in parentheses following the program name. Programs in other accounts and/or groups may be compiled into your USL file by using a fully qualified file designator, of the form:

```
filename [/lockword]  [.group[.account]]
```

For example:

```
$COMPILE PROGX.PUB.SYS
```

However, only filename is associated with the RBM in the USL file.

If an active entry with the same name already exists in the USL file, the existing entry is made inactive. This ensures that the most recently compiled program is the one prepared into the program file.

Any desired compile options must be established with a CONTROL command prior to issuing COMPILE.

2-13. EXIT COMMAND

Example:

```
$EXIT
```

The EXIT command terminates use of the BASIC/3000 Compiler.

PROGRAMMING EXAMPLES

SECTION III

Figures 3-1 and 3-2 illustrate typical examples of using the BASIC/3000 Compiler. For both examples, it is assumed that the BASIC programs have been entered into the system via the BASIC/3000 Interpreter, have been debugged, and have been SAVEd with the FAST option.

In the printouts shown in the examples, user inputs are distinguished by system prompts (:) or subsystem prompts ($). All other printouts are system or subsystem responses.

3-1. SINGLE-PROGRAM EXAMPLE

Figure 3-1 shows the compilation, preparation, and execution of a simple 12-line BASIC program, the purpose of which is to list the contents of any ASCII file. The name of the program is FILELIST.

The step-by-step commentary facing the printout explains the actions occurring as a result of each input command.

3-2. SEGMENTED PROGRAMS EXAMPLE

Figure 3-2 shows the compilation, preparation and execution of programs which are linked by CHAIN and INVOKE statements and are compiled into several segments.

Program A is the main program. When run, this program will receive initial control from the outer block. It will prompt a user for a program name (B or C) to execute next. Thus program A will selectively INVOKE to either program B or program C. Program B also chains to program D.

As described in Figure 3-2, programs B and D are compiled into SEG3, C into SEG2, and A into SEG1.

To illustrate the actions occurring when these segmented programs are executed, consider a response of "B" to the program prompt, "PROGRAM TO EXECUTE?" In such a case, program control is as follows: when A receives control, it invokes B. B then chains to D. When D executes an END statement, control passes back to A. A terminates by executing a STOP statement.

A time sequence showing the run-time environment is given in Figure 3-3. The sequence shows the code segment and stack conditions at five points in time. Operation is as follows.

TIME 1

The outer block, B'RUNOB, receives initial control from the loader. The outer block calls a library procedure which opens the following files.

BASIN
the input file, which is read from by the statements INPUT, ENTER, and LINPUT. The default for this file is $STDINX. It can be overridden using the formal designator BASIN in an MPE/3000 :FILE command.

BASLIST
The output file, which is written to by the statements PRINT, PRINT USING, MAT PRINT, and MAT PRINT USING. The default for this file is $STDLIST. It can be overridden using the formal designator BASLIST in an MPE/3000 :FILE command.

Once the files are open, buffer space is allocated dynamically on the stack. Other BASIC run-time variables are also initialized in primary and secondary DB. Then the program specified by START= (program A) during the compilation is invoked.

TIME 2

Program A, in segment SEG1, receives control. It allocates its program variables on the stack and then asks for a program name to be input. The name read in was B, so program A invokes to program B.

TIME 3

Program B, in segment SEG3, receives control and does local allocation of program variables. COMmon space is also allocated by B and it passes information to D through that COMmon. The actual COMmon space is dynamically allocated in the DL to DB area and is completely managed for the user. Pointers to the COMmon space are in primary DB. Program B now chains to D.

TIME 4

The program local space for program B is overlayed by the local space of program D. The space does not need to be saved because a CHAIN does not return. Program D, also in segment SEG1 (but not required to be) receives control and allocates its local variables. When it executes an END statement, program D exits back to program A. COMmon is deallocated prior to the exit.

TIME 5

Program A, in segment SEG1, receives control again and requests the next program to execute. Because a null string (zero characters) is input, program A terminates in a STOP statement.

After logging on (or following a :JOB card in batch), specify the file characteristics for the USL file which is to receive the compiled code, namely: it is a NEW file to be SAVEd for future access. For this example, the USL file will be named USLF. Next, load the compiler via the BASICOMP command. This command opens three files. Defaults are accepted for commandfile (note comma) and for listfile. These defaults are $STDIN and $STDLIST respectively. For uslfile, a back reference is made to USLF (note asterisk). (1)

In response to compiler prompts, specify desired compile options via CONTROL commands: initialize the uslfile (USLINIT) and request a SOURCE listing, a symbol MAP, and a LABEL map. The compiled code will be put into a segment called TESTSEG. Then give the COMPILE command. (2)

The compiler now prints out a source listing and compiles the program. (3)

Next the symbol map is printed, showing where each variable is located in the stack. The ",I" following an address indicates an indirect address (see string variables). Addresses are in octal. (4)

Next the label map is printed, giving the starting location in the code list for each statement. (CODE list was not requested in this example.) Statements 10, 20, 100, and 110 are shown "equivalent" to other statements. This indicates, for example, that any branch to line 110 will actually branch to line 80. Lines 120 and 130 are "equivalent" to the end of the program. The ENTRY POINT is the octal address at which execution will begin. STACK indicates how much stack is required for this program; this can be used in estimating the MAXDATA parameter when preparing the program file (see "MAXDATA Parameter" in Section II). (5)

Compilation is now complete. In response to the next command prompt, exit back to MPE/3000. (6)

Now prepare the USL file into a job-temporary program file named BASPROG. MAXDATA must be specified; 2000 words is estimated for this example. A PMAP is requested, though it is not necessary. (7)

After the file is prepared, MPE/3000 prints out the PMAP. Note that two segments were prepared: the default segment SEG' (which contains an outer block generated by the compiler with an external reference to the compiled program, FILELIST) and the TESTSEG segment which contains the program. (8)

To make the job-temporary program file BASPROG permanent, specify a SAVE command. (9)

Now that the program exists as a program file, it can be run. (10)

The compiled BASIC program runs, prints out a prompt for input. We respond with a file name which, for illustrative purposes here, is a test file called TESTFILE consisting of 5 records. We terminate by typing a carriage return in response to the next prompt. MPE/3000 prints an END OF PROGRAM message and prompts for the next command. (11)

Figure 3-1A. Compilation of Single Program

```
:FILE USLF,NEW; SAVE
:BASICOMP ,*USLF

PAGE 0001   HP32103A.00.00

$CONTROL SOURCE, MAP, LABEL, SEGMENT=TESTSEG
$COMPILE FILELIST
  10 REM  --  FILE LISTER
  20 FILES *
  30 PRINT "FILE NAME? ";
  40 LINPUT B$
  50 IF LEN(B$)<>0 THEN DO
  60   ASSIGN B$,1,I
  70   ON END #1 THEN 30
  80   LINPUT #1;A$
  90   PRINT USING 100;A$
 100   IMAGE 80A
 110   GOTO 80
 120 DOEND
 130 DIM A$[80],B$[8]

   SYMBOL MAP

   FILELIST

 NAME    TYPE          STRUCTURE       ADDRESS

 A$      STRING                        Q+  5,I
 B$      STRING                        Q+  6,I
 I       REAL          SIMPLE VAR      Q+  7

 LABEL MAP    ENTRY POINT     1   STACK    74
  LABEL  LOCATION   LABEL  LOCATION   LABEL  LOCATION   LABEL  LOCATION
   10=  30    63    20=  30    63     30         63     40         100
   50        104    60        111     70         121    80         142
   90        155   100=  80     0    110=  80   171   120= END   173
  130= END   173
$EXIT

 END OF COMPILE
:PREP USLF,BASPROG; MAXDATA=2000; PMAP

PROGRAM FILE BASPROG.BASICOMP.LANG

SEG'              0
   NAME            STT  CODE ENTRY SEG
   B'LLBL            1     0     0
   UNLOADPROC        4                 ?
   LOADPROC          5                 ?
   B'ABORTUSER       6                 ?
   B'RUNOB           2   232   232
   FILELIST          7                 1
   B'INITIAL        10                 ?
   TERMINATE'       11                 ?
   NOECHO            3   232   235
   SEGMENT LENGTH        254
TESTSEG           1
   NAME            STT  CODE ENTRY SEG
   FILELIST          1     0     1
   B'ABORTUSER       2                 ?
   B'ASSIGN          3                 ?
   B'DEALLOCATE      4                 ?
   B'FILES           5                 ?
   B'FLINPUT         6                 ?
   B'LINPUT          7                 ?
   B'PRINTSTR       10                 ?
   B'PRINTUSTR      11                 ?
   SEGMENT LENGTH        234

PRIMARY DB         17    INITIAL STACK    1440    CAPABILITY        600
SECONDARY DB      400    INITIAL DL          0    TOTAL CODE        510
TOTAL DB          417    MAXIMUM DATA     3720    TOTAL RECORDS      12
ELAPSED TIME   00:02:06.607                    PROCESSOR TIME   00:01.275

 END OF PREPARE
:SAVE BASPROG
:RUN BASPROG

FILE NAME? TESTFILE
<<---------------TEST FILE--------------------->>
<<                                             >>
<<  TOTAL LINES = 5                            >>
<<                                             >>
<<---------------------------------------------->>
FILE NAME?

 END OF PROGRAM
:
```

Figure 3-1B. Compilation of Single Program

Defaults are accepted for uslfile ($NEWPASS) and listfile ($STDLIST). Commandfile is TEXT. (1)

Page title. (2)

Compile programs B and D into segment SEG3. Program B has optional entry points at statements 50 and 100. Also, program B may be called by a string expression, so its name is entered in the Chain-Invoke table. (3)

Compile program C into SEG2 with optional entry points at statements 50 and 100. Put name C in Chain-Invoke table. (4)

Compile program A into SEG1. Declare this to be the main program. Request source listing. (5)

Source listing of main program. (6)

Exit to MPE/3000. (7)

Request to prepare and run programs in uslfile (now $OLDPASS) and provide a PMAP. (8)

PMAP begins. (9)

Outer block in SEG' contains STT links to programs, B, C, and A, in segments 3, 2, and 1 respectively. (10)

Segment 1 (SEG1) contains program A. (11)

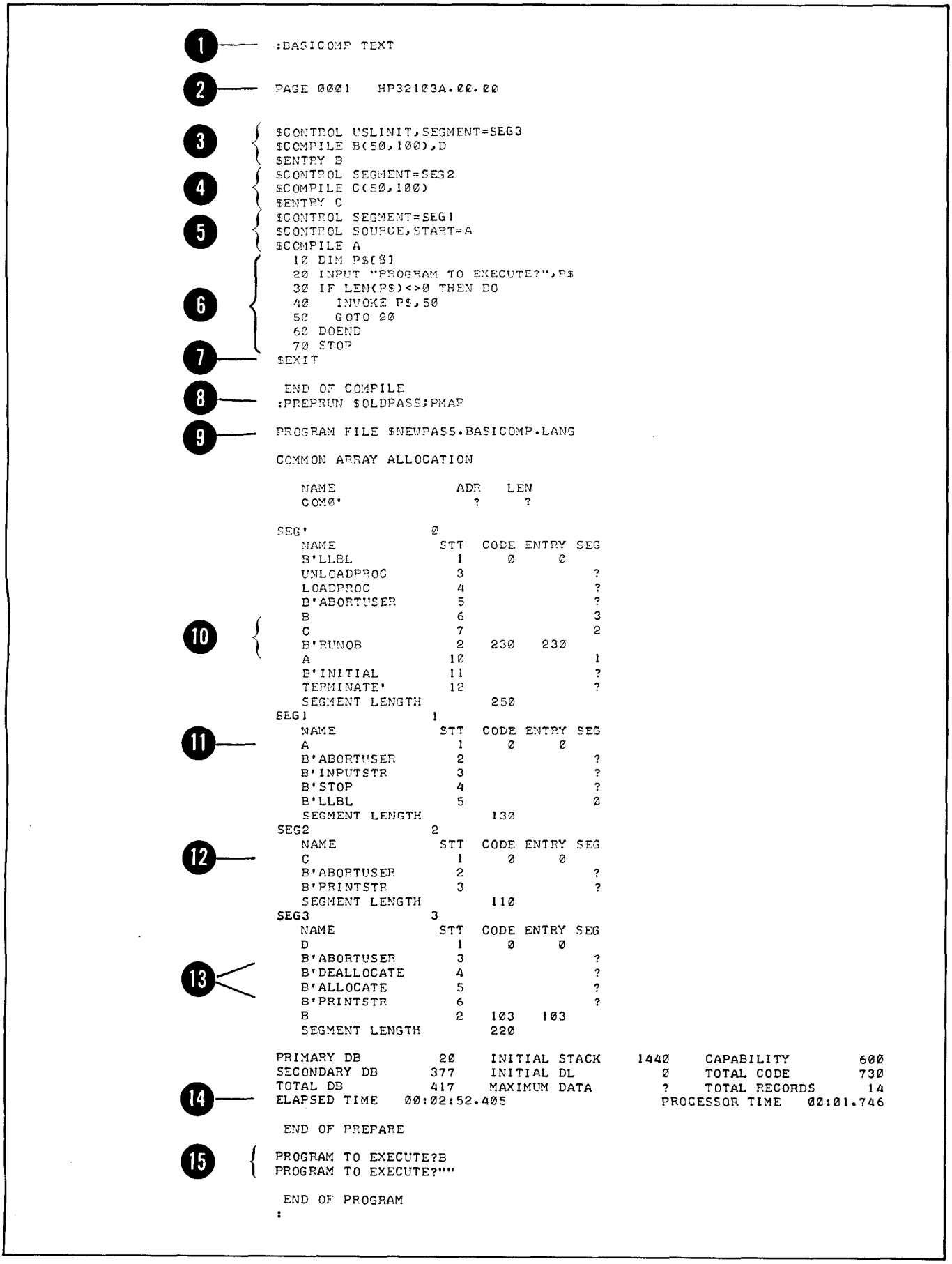
Segment 2 (SEG2) contains program C. (12)

Segment 3 (SEG3) contains programs D and B. (13)

End of PMAP. (14)

Sample run of program. Terminates by input of null string. (15)

Figure 3-2A. Compilation of Segmented Programs



```
:BASICOMP TEXT

PAGE 0001   HP32103A.00.00

$CONTROL USLINIT,SEGMENT=SEG3
$COMPILE B(50,100),D
$ENTRY B
$CONTROL SEGMENT=SEG2
$COMPILE C(50,100)
$ENTRY C
$CONTROL SEGMENT=SEG1
$CONTROL SOURCE,START=A
$COMPILE A
  10 DIM P$[8]
  20 INPUT "PROGRAM TO EXECUTE?",P$
  30 IF LEN(P$)<>0 THEN DO
  40   INVOKE P$,50
  50   GOTO 20
  60 DOEND
  70 STOP
$EXIT

 END OF COMPILE
:PREPRUN $OLDPASS;PMAP

PROGRAM FILE $NEWPASS.BASICOMP.LANG

COMMON ARRAY ALLOCATION

   NAME                ADR   LEN
   COM0'                 ?     ?

SEG'                  0
   NAME                STT  CODE ENTRY SEG
   B'LLBL                1     0     0
   UNLOADPROC            3                 ?
   LOADPROC              4                 ?
   B'ABORTUSER           5                 ?
   B                     6                 3
   C                     7                 2
   B'RUNOB               2   230   230
   A                    10                 1
   B'INITIAL            11                 ?
   TERMINATE'           12                 ?
   SEGMENT LENGTH          250
SEG1                  1
   NAME                STT  CODE ENTRY SEG
   A                     1     0     0
   B'ABORTUSER           2                 ?
   B'INPUTSTR            3                 ?
   B'STOP                4                 ?
   B'LLBL                5                 0
   SEGMENT LENGTH          130
SEG2                  2
   NAME                STT  CODE ENTRY SEG
   C                     1     0     0
   B'ABORTUSER           2                 ?
   B'PRINTSTR            3                 ?
   SEGMENT LENGTH          110
SEG3                  3
   NAME                STT  CODE ENTRY SEG
   D                     1     0     0
   B'ABORTUSER           3                 ?
   B'DEALLOCATE          4                 ?
   B'ALLOCATE            5                 ?
   B'PRINTSTR            6                 ?
   B                     2   103   103
   SEGMENT LENGTH          220

PRIMARY DB           20    INITIAL STACK    1440    CAPABILITY        600
SECONDARY DB        377    INITIAL DL          0    TOTAL CODE        730
TOTAL DB            417    MAXIMUM DATA        ?    TOTAL RECORDS      14
ELAPSED TIME   00:02:52.405                       PROCESSOR TIME   00:01.746

 END OF PREPARE

PROGRAM TO EXECUTE?B
PROGRAM TO EXECUTE?""

 END OF PROGRAM
:
```

Figure 3-2B. Compilation of Segmented Programs

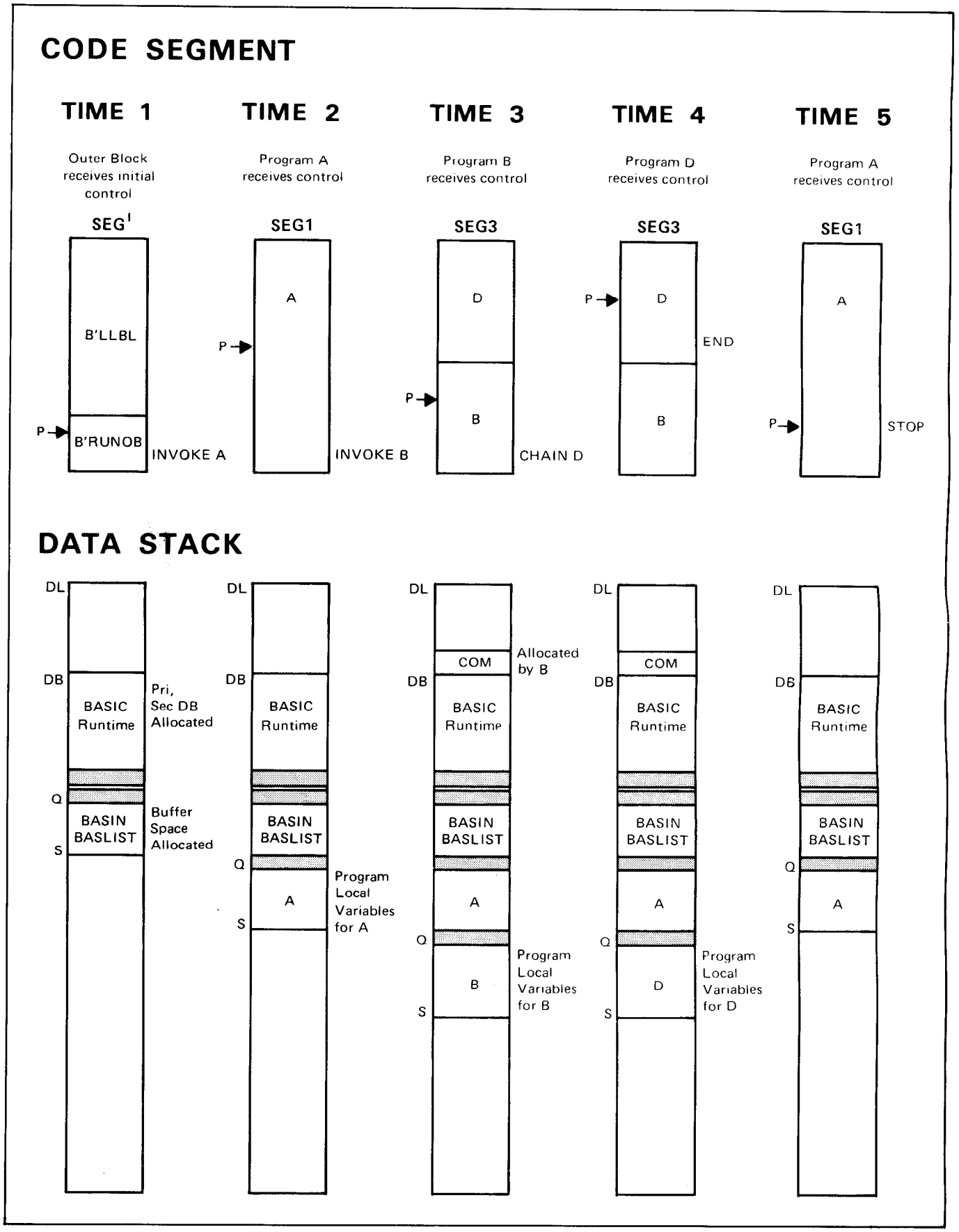


Figure 3-3. Run-Time Environment of Segmented Example

INTERPRETING MESSAGES

SECTION IV

4-1. COMPILE-TIME ERROR MESSAGES

Error messages which may occur during the compile phase are presented as a WARNING or ERROR in the following format:

```
**WARNING
**ERROR        message [: addendum]
```

where the message is one of those listed below, and the addendum is additional information given with some error messages to further identify the cause of the error.

1 COMPILER OUT OF SPACE
Space requirements for compiling program exceed the available memory. See your system manager. The compiler must be prepared (:PREP) with a larger MAXDATA, and/or the maximum stack segment size (defined during system configuration) must be expanded.

2 INVALID COMMAND
Not a valid compiler command.

3 INVALID CHARACTER
Extraneous character in command processing.

4 USL FILE PREVIOUSLY INITIALIZED
Given when USLINIT parameter is specified but USL file has already been initialized.

5 USLINIT INVALID AFTER $COMPILE COMMAND
Initialization of the USL file is invalid after a program has been compiled.

6 INVALID NUMBER OF LINES
Number of lines cannot be set less than 10 or greater than 9999.

7 EXPECTED A QUOTE
TITLE string must be delimited by a quote.

8 START INVALID UNDER SUBPROGRAM MODE
When SUBPROGRAM is specified, all subsequent START parameters are ignored.

9 EXPECTED PROGRAM NAME
A program name is expected as the parameter to COMPILE command.

10 EXPECTED A <
Comments must begin with<<; given if only a single< is detected.

11 EXPECTED A &
Commands which are continued must have ampersand as last character of the line; given if comma is last character of the line.

12 EXPECTED A =
LINES, START, and SEGMENT all must be followed by an equal sign.

13 INVALID SUBPARAMETER
Subparameter must be a name for START= and SEGMENT=.

14 SUBPARAMETER NAME TOO LONG, TRUNCATED TO 15 CHARACTERS

15 $CONTROL PARAMETER EXPECTED

16 USL FILE INITIALIZED DUE TO BAD CHARACTERISTICS
Invalid directory or directory element. Given on COMPILE if initialization is required but user fails to specify USLINIT.

17 INVALID $CONTROL PARAMETER

18 $ENTRY INVALID UNDER SUBPROGRAM MODE
When SUBPROGRAM is specified, all subsequent ENTRY commands are ignored.

19 $TITLE STRING TOO LONG, TRUNCATED TO 104 CHARACTERS

20 EXTRANEOUS CHARACTER

21 UNABLE TO OPEN FASTSAVE FILE
The file may not exist or cannot be opened because of access or security restrictions.

22 FILE NOT TYPE FASTSAVE
File specified in COMPILE command is not a BASIC fastsave file.

23 I/O ERROR ON FASTSAVE FILE

24 BAD FASTSAVE FILE ATTRIBUTES
Recreate the fastsave file through the BASIC/3000 Interpreter.

26 SUBPROGRAM INVALID AFTER $ENTRY OR START
After ENTRY command or START parameter has been specified, SUBPROGRAM parameter is ignored.

27 TOO MANY PROGRAM VARIABLES
The number of program variables is limited; see paragraph 1-10 in Section I. Reduce number of variables by "reusing" variables, putting values into arrays, or putting variables in COMmon.

28 INVALID PROGRAM NAME
Program names for COMPILE command must be alphanumeric and begin with alphabetic.

29 QUALIFIER DELETED
Group and account names and lockword are deleted from program name and not used as part of the RBM symbolic name.

30 UNABLE TO PROCESS $ENTRY COMMANDS
A temporary file is necessary and could not be opened.

31 TOO MANY FUNCTION VARIABLES
Function references too many local and program variables; see paragraph 1-10 in Section I.

32 $ENTRY NAME TOO LONG, TRUNCATED TO 8 CHARACTERS
ENTRY names cannot exceed eight characters.

33 CODE SEGMENT > 16K WORDS
The program being compiled creates a code segment which is more than 16,384 words, the largest code segment size possible. The program should be divided into smaller programs.

34 MORE THAN 16 FILES
A maximum of 16 files is allowed.

35 USL FILE SPACE EXCEEDED
The space in the USL file was exceeded. Build the USL file with more records and recompile.

Messages 36 through 52 refer to PRINT USING strings specified as string expressions. The format string is echoed with a caret character or up-arrow (depending on particular device) indicating the point at which the error was detected.

36 INVALID CHARACTER
37 MISSING GROUP REPLICATOR
38 REPLICATOR NOT ALLOWED
39 MISSING RIGHT QUOTE
40 MISSING NUMBER AFTER APOSTROPHE
41 INVALID NUMERIC SPECIFICATION
42 NUMBER > 255
43 MISSING D-PART
44 REPLICATOR > 255
45 INVALID STRING SPECIFICATION
46 MISSING LEFT PARENTHESIS IN C-SPECIFICATION
47 MISSING SEPARATOR
48 INVALID C-SPECIFICATION
49 UNBALANCED PARENTHESES
50 MISSING RIGHT PARENTHESIS
51 INVALID COMPLEX SPECIFICATION
52 REPLICATOR = 0

63 COMMAND FILE RECORD TOO LONG, TRUNCATED TO 80 CHARACTERS

64 BAD LIST FILE CHARACTERISTICS
Listfile is not type ASCII, or record size is less than 40 characters, or the compiler could not get write access to listfile.

65 UNABLE TO OPEN LIST FILE

66 UNABLE TO OPEN COMMAND FILE

67 BAD COMMAND FILE CHARACTERISTICS
Commandfile is not type ASCII or is the $NULL file, or the compiler could not get read access to the commandfile.

68 UNABLE TO OPEN USL FILE

69 BAD USL FILE CHARACTERISTICS
USL file is not binary, or the file is not type USL, or the record size is not 128 words, or the compiler could not get read and write access to the file.

70 END OF FILE ON COMMAND FILE
The EXIT command is missing or misplaced in the job stream.

71 END OF FILE ON USL FILE
The USL file is invalid. Recompile into a new USL file.

72 END OF FILE ON FASTSAVE FILE
The fastsave file is invalid. Recreate the file through the BASIC/3000 Interpreter.

73 END OF FILE ON TEMPORARY FILE
This is a compiler error. See your system manager.

74 END OF FILE ON LIST FILE

75 I/O ERROR ON COMMAND FILE

76 I/O ERROR ON USL FILE

77 I/O ERROR ON TEMPORARY FILE

78 I/O ERROR ON LIST FILE

79 I/O ERROR ON UNKNOWN FILE
This is a compiler error. See your system manager.

80 UNABLE TO CLOSE COMMAND FILE

81 UNABLE TO CLOSE USL FILE

82 UNABLE TO CLOSE FASTSAVE FILE

83 UNABLE TO CLOSE LIST FILE

84 UNABLE TO CLOSE UNKNOWN FILE
This is a compiler error. See your system manager.

4-2. RUN-TIME WARNING MESSAGES

The execution of compiled BASIC programs is similar to running a program in the interpreter with the NOWARN option; i.e., most warning messages (such as overflow and underflow warnings) are suppressed. The few warnings given while running compiled programs allow for recovery during input when the input file is interactive with either the output file or $STDLIST.

BAD INPUT—RETYPE FROM ITEM nn

A string was input to a numeric variable or vice versa; or the form of a numeric item is incorrect.

STRING OVERFLOW—RETYPE FROM ITEM nn

The string item exceeds 255 characters.

>BASIC NOT ALLOWED

The >BASIC option was disabled with BRK function; the user is not allowed to use this feature.

TRANSMISSION ERROR—RE-INPUT LINE

A transmission error occured during input, causing characters to be lost.

4-3. RUN-TIME ERROR MESSAGES

When an error is detected during execution, one or more messages are printed to aid the user in isolating the problem. If the BASLIST file and $STDLIST files are different, the messages are printed on both files.

Run-time error messages include the following information:

a. A message of the form

   **ERROR nn: message

where nn and message are as listed below.

b. For file errors, a file information display indicating the characteristics of the file and the exact nature of the file error.

c. A stack traceback indicating where the error occurred. Compiler library routines are denoted by prefix S; user library routines by P or G; program modules have no prefix. An example of a stack traceback is shown in Figure 4-1.

0 OUT OF DATA
Exhausted the DATA list while attempting to execute a READ or MAT READ statement.

1 NUMERIC DATA FOR STRING ITEM
The destination variable in a READ or MAT READ statement was string but the next item in the DATA list was numeric.

2 SUBSTRING DESIGNATOR EXCEEDS LOGICAL LENGTH PLUS ONE
The first substring designator for a destination string must always be less than or equal to string's logical length plus one to ensure a "contiguous" string.

4 DESTINATION SUBSTRING EXCEEDS PHYSICAL SIZE
The first substring designator for a destination string must be less than or equal to the physical length; and the substring must not extend beyond the physical length.

5 FIRST SUBSTRING DESIGNATOR < = ZERO

6 UNABLE TO FIND CHAIN/INVOKE PROCEDURE: name
The CHAIN/INVOKE procedure does not exist in the program file or in an SL file.

7 INVALID CHAIN/INVOKE PROCEDURE NAME
CHAIN/INVOKE name resulting from a string expression did not have the form of a file name:

   filename [/lockword] [.group[.acct]]

Specifically, the name erroneously began with a non-alphabetic or contained non-alphanumeric characters other than a slash (/) or period, or slash or period was followed by non-alphabetic character.

```
**ERROR 29: ATTEMPT TO ACCESS UNOPENED FILE

      ***       STACK  DISPLAY       ***


                    S=000727    DL=177732    Z=002057
 Q=000733 P=000724  LCST= S130  STAT=U,1,1,L,0,0,CCE    X=000037

 Q=000652 P=003662  LCST= S126  STAT=U,1,1,L,0,0,CCG    X=000001
 Q=000645 P=000310  LCST=  000  STAT=U,1,1,L,0,0,CCG    X=000000
 Q=000606 P=002745  LCST= S127  STAT=U,1,1,L,0,0,CCL    X=000000
 Q=000431 P=000234  LCST=  000  STAT=U,1,1,L,0,0,CCG    X=000000


ERR 2 :
ABNORMAL PROGRAM TERMINATION
:
```

Figure 4-1. Stack Traceback Example

8 INTEGER OVERFLOW
Integer expression resulted in a value less than -32768 or greater than +32767.

9 INTEGER DIVIDE BY ZERO

10 LOG OF NEGATIVE NUMBER

11 SQR OF NEGATIVE NUMBER

12 ZERO TO ZERO POWER

13 ZERO TO INTEGER POWER<= ZERO

14 NEGATIVE NUMBER TO REAL POWER
Floating point base (type real or long) to a floating point power (results in type complex value).

15 STRING EXCEEDS 255 CHARACTERS
Maximum source string length in any expression is 255 characters.

16 INVALID SUBSTRING DESIGNATOR
For a string variable of the form
S$(f,l)
the second substring designator must not be less than the first minus one. For a string variable of the form
S$(f;l)
the second substring designator must be greater than or equal to zero.

17 FAILURE TO "CREATE" BASIC.PUB.SYS
The BASIC/3000 Interpreter does not exist in the system. See system manager.

18 FAILURE TO "ACTIVATE" BASIC.PUB.SYS
Cannot run the BASIC/3000 Interpreter. See system manager.

21 ">EOD" INPUT
End of input detected due to ">EOD".

23 INVALID STRING INPUT
Unbalanced quotes in string read by INPUT statement.

25 INVALID NUMBER INPUT
Expected numeric constant input but the constant had invalid form, or the destination variable is type integer and the constant is less than -32767 or greater than +32767.

26 INVALID FILE NUMBER
File number specified in a FILES statement is less than or equal to zero, or exceeds the number of local files.

27 NEGATIVE FILE SIZE
File limit specified in CREATE statement is negative.

28 INVALID RECORD SIZE
Record size specified in a CREATE statement is less than 4 or greater than 319.

29 ATTEMPT TO ACCESS UNOPENED FILE
Reference was made (e.g., in a READ# statement) to a file which is not currently open.

30 ATTEMPT TO LOCK NON-LOCKABLE FILE
Attempted to execute LOCK# for a file not opened with NL or WL option.

31 ATTEMPT TO LOCK SECOND FILE
Only one file may be locked (with LOCK#) at a time.

32 ATTEMPT TO UNLOCK NON-LOCKABLE FILE
Attempted to execute UNLOCK# for a file not opened with NL or WL option.

33 ADVANCE ATTEMPTED ON NON-BASIC FILE
ADVANCE# may be used only for BASIC formatted files.

34 UNABLE TO OPEN FILE: TOO MANY FILES
FILES or ASSIGN statement cannot open file because 16 files are currently open.

35 UNABLE TO OPEN USER FILE: filename
FILES statement cannot open file. A slash following filename indicates the presence of a password, which is not shown.

36 UPDATE ATTEMPTED ON NON-BASIC FILE
UPDATE# may be used only for BASIC formatted files.

37 UNABLE TO OPEN FILE: INSUFFICIENT SPACE FOR BUFFER
FILES or ASSIGN statement cannot open file. Use larger MAXDATA in :RUN or :PREP command.

38 INVALID LINKED-FILE NUMBER
Linked-file number (specified on FILES statement in the form "#integer") exceeds the number of global files.

39 INSUFFICIENT SPACE FOR COMMON
Unable to expand stack for COMmon area specified in COM statement. Use larger MAXDATA in :RUN or :PREP command.

40 LINPUT# ATTEMPTED ON NON-ASCII FILE
LINPUT# may be used only for ASCII files.

41 ATTEMPT TO WRITE ON READ-ONLY FILE

42 ATTEMPT TO READ FROM WRITE-ONLY FILE

43 DIRECT ACCESS ATTEMPTED ON SERIAL FILE

44 END OF FILE
End-of-file detected and no ON END# for that file was in effect.

45 NUMERIC UPDATE ITEM FOR STRING DATA
The expression in the UPDATE# statement was numeric but the next item in the BASIC formatted file was a string.

46 CHAIN/INVOKE/RUN TO UNDECLARED LABEL
Destination label specified in a CHAIN or INVOKE statement or the PARM= parameter in the :RUN command was not declared in a label list at compile time. Destination program must be recompiled.

47 STRING EVALUATES TO INVALID NUMBER IN CONVERT
For CONVERT of string to number, the string expression resulted in a number of invalid form, or the destination variable is type integer and the resulting number is less than -32767 or greater than +32767.

48 INCOMPATIBLE ARRAY DIMENSIONS
In a MAT statement, the destination array is one-dimensional but the right-hand side results in a two-dimensional with more than one column.

49 INVALID DAT$ PARAMETER
The first-character designator is less than 1 or exceeds the last-character designator. Or the last-character designator exceeds 27.

50 COLUMNS < > ROWS IN MAT MULTIPLY
For MAT multiplication, the multiplicand must have dimensions m-by-p while the multiplier must have dimensions p-by-n.

51 DESTINATION ARRAY NOT LARGE ENOUGH
In a MAT statement, the number of elements contained in the right-hand side exceeds the maximum number of elements in the destination array.

55 ATTEMPT TO EXECUTE FNEND
Missing function RETURN statement.

56 MISSING SUBSCRIPT IN FIRST OCCURRENCE OF COM
First occurrence of COM declaration uses (*) or (*,*) indicators. Dimensions must be declared explicity or must CHAIN/INVOKE from a program which declares COM explicitly.

Messages 57 through 75 all refer to a PRINT USING format specified as a string expression. The format string is echoed with a caret character or up-arrow (depending on particular device) indicating the point at which the error was detected.

57 INVALID CHARACTER
58 MISSING GROUP REPLICATOR
59 REPLICATOR NOT ALLOWED
60 MISSING RIGHT QUOTE
61 MISSING NUMBER AFTER APOSTROPHE
62 INVALID NUMERIC SPECIFICATION
63 UNBALANCED PARENTHESES
64 MISSING RIGHT PARENTHESIS

65 NUMBER > 255
66 MISSING D-PART
67 MISSING SEPARATOR
68 MISSING LEFT PARENTHESIS
69 INVALID C-SPECIFICATION
70 INVALID COMPLEX SPECIFICATION
71 REPLICATOR = ZERO
72 INVALID STRING SPECIFICATION
73 REPLICATOR > 255

74 STRING SPECIFICATION FOR NUMERIC ITEM
Attempt was made to output numeric item in PRINT USING with a string specification in format string.

75 NUMERIC SPECIFICATION FOR STRING ITEM
Attempt was made to output string item in PRINT USING with a numeric specification in format string.

86 I/O ERROR ON USER FILE

87 UNABLE TO OPEN LIST FILE

88 LIST FILE NOT TYPE ASCII

89 UNABLE TO OPEN INPUT FILE

90 INPUT FILE NOT TYPE ASCII

91 I/O ERROR ON LIST FILE

92 I/O ERROR ON INPUT FILE

93 I/O ERROR ON $STDLIST

94 NUMERIC FILE DATA FOR STRING ITEM
The destination variable in a READ# or MAT READ# statement was string but the next item in the file was numeric.

95 STRING FILE DATA FOR NUMERIC ITEM
The destination variable in a READ# or MAT READ# statement was numeric but the next item in the file was string.

96 STRING DATA FOR NUMERIC ITEM
The destination variable in a READ or MAT READ statement was numeric but the next item in the DATA list was string.

97 STRING UPDATE ITEM FOR NUMERIC DATA
The expression in the UPDATE# statement was string but the next item in the BASIC formatted file was numeric.

READER COMMENT SHEET

BASIC/3000 Compiler

Reference Manual

32103-90001 November 1974

We welcome your evaluation of this manual. Your comments and suggestions help us improve our publications. Please use additional pages if necessary.

Is this manual technically accurate?

Is this manual complete?

Is this manual easy to read and use?

Other comments?

FROM:

Name _____

Company _____

Address _____

_____

_____

FOLD

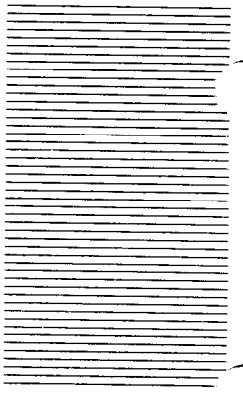
FOLD

FIRST CLASS
PERMIT NO. 1020
SANTA CLARA
CALIFORNIA

BUSINESS REPLY MAIL

No Postage Necessary if Mailed in the United States. Postage will be paid by

Publications Manager, Product Support Group
Hewlett-Packard Company
General Systems Division
5303 Stevens Creek Boulevard
Santa Clara, California 95050



FOLD

FOLD