HEWLETT hp PACKARD

TRACE/3000

Reference Manual

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Computer Museum

TRACE/3000

Reference Manual

HEWLETT PACKARD

HEWLETT-PACKARD COMPANY
11000 WOLFE ROAD, CUPERTINO, CALIFORNIA, 95014

PART NO. 03000-90015

Printed in U.S.A. 10/75

LIST OF EFFECTIVE PAGES

Changed pages are identified by a change number adjacent to the page number. Changed information is indicated by a vertical line in the outer margin of the page. Original pages do not include a change number and are indicated as change number 0 on this page. Insert latest changed pages and destroy superseded pages.

Pages	Effective Date
Title	Oct 1975
iii to vi	Oct 1975
1-1 to 1-4	Oct 1975
2-1 to 2-47	Oct 1975
3-1 to 3-17	Oct 1975
A-1	Oct 1975
B-1 to B-2	Oct 1975
I-1 to I-2	Oct 1975

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.

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another program language without the prior written consent of Hewlett-Packard Company.

Copyright © 1975 by HEWLETT-PACKARD COMPANY

PREFACE

This publication is the reference manual for TRACE/3000. TRACE/3000 is a subsystem of the MPE/3000 Operating System that is used to find errors in programs written in SPL/3000 (Systems Programming Language for the HP 3000 Computer System) and FORTRAN/3000 (a version of FORTRAN IV for the HP 3000 Computer System).

The content of this publication is

Section I	introduces the TRACE/3000 subsystem. The features of TRACE/3000 are discussed and a summary of all commands is included.
Section II	explains TRACE/3000 commands and files.
Section III	explains how to use TRACE/3000.
Appendix A	defines how TRACE/3000 treats constants of different types.
Appendix B	explains how TRACE/3000 treats abbreviations.
Index	contains an alphabetical listing of the main topics of this manual.

Other publications which should be available for reference when using this manual are:

HP 3000 Computer System Reference Manual, (03000-90019)

MPE/3000 Operating System Reference Manual, (32000-90002)

FORTRAN/3000 Reference Manual, (32102-90001)

SPL/3000 Reference Manual, (03000-90002)

EDIT/3000 Reference Manual, (03000-90012)

CONVENTIONS USED IN THIS MANUAL

Parameters

[A]	optional
[A] [B]	optional, select one
[A] [B] [C]	optional, select one or more in any order
A	mandatory
{ A } { B }	mandatory, select one
	italics denote a parameter which must be replaced by a variable parameter that the user assigns Example: CALL *name* *name* one to 15 alphanumeric characters
Dialogue:	where it is necessary to distinguish user input from computer output the user input is underlined Example: NEW NAME? ALPHA 1
Control Characters:	Control characters are indicated by a superscript c Example: Y^c
return	indicates a carriage return
linefeed	indicates a line feed

CONTENTS

ILLUSTRATIONS

TABLES

INTRODUCING TRACE/3000

SECTION I

1-1. WHAT IS TRACE/3000?

TRACE/3000 is a subsystem of the HP 3000 Multiprogramming Executive Operating System (MPE/3000) that is used to find errors in programs written in SPL/3000 (Systems Programming Language for the HP 3000 Computer System) and FORTRAN/3000 (a version of FORTRAN IV for the HP 3000 Computer System).

TRACE/3000 can be used during a batch job or an interactive session.

1-2. TRACE/3000 FEATURES

By inserting TRACE/3000 *control statements* into a source program, and TRACE/3000 *commands* into a *BATCH* or *INTERACTIVE* file, you can monitor the step-by-step execution of a program, or change the values of variables and array elements during program execution.

For example, depending on the TRACE/3000 commands used, it is possible to

- Monitor the points in a source program where control is passed to a procedure or subroutine. TRACE/3000 will display information showing when the routine is called, when it is entered, and when control is returned back to the calling program unit. In addition, TRACE/3000 will display the values of all parameters (passed to the routine) at the time of the *call* to the routine.

- Monitor the values of variables during program execution. TRACE/3000 will display the name and current value each time a variable is encountered during program execution, except when a variable is used on the right side of an assignment statement or as an actual parameter.

- Monitor the values of array elements during program execution. TRACE/3000 will display the array name and subscript, and the current value of this element each time it is encountered, except when the element is used on the right side of an assignment statement or as an actual parameter. In addition, in SPL/3000, an array element will not be displayed when used in a MOVE or SCAN operation.

- Monitor program labels. TRACE/3000 will display all labels for SPL/3000 and FORTRAN/3000 programs as they are encountered during program execution.

  Note: FORTRAN/3000 programs use numeric statement labels, as, for example:

  ```
  10      DO 20 I = 1,15
  ```

  statement label

  SPL/3000 programs use alphabetic statement labels, as, for example:

  ```
  ENDSORT:
  ```

- Check the correct spelling of program unit names, array names, and variable names if you enter the first letter of the name. Once the first letter is entered, TRACE/3000 then displays the correct *full* name.

  Note: If there is more than one item starting with a given letter, only the first item in alphabetic sequence is displayed. See Appendix B.

- Change the values of variables and array elements (in an interactive session only) without terminating program execution. TRACE/3000 displays the current value and you may enter a new value at that time.

- Determine the relative addresses of variables and array elements.

1-3. TRACE/3000 OPERATING MODES

TRACE/3000 can be run in either of two modes:

- Interactive mode. In an interactive session, you can enter TRACE/3000 commands from the terminal. Control returns to the terminal after a HALT command (see Section II) is executed, or you can regain control from the program by entering CONTROL Y (holding the CONTROL key, or equivalent, down and pressing Y).

- Batch mode. In batch mode, commands cannot be entered once execution has started. Execution terminates if a HALT command is executed.

  Note: In either mode of operation, TRACE/3000 output will be listed on the standard list device unless the TRACE/3000 list file designator, TRCLIST, has been equated to another device with a :FILE command. (See Section III.)

1-4. TRACE/3000 CONTROL STATEMENTS AND COMMANDS

TRACE/3000 is invoked by *control statements*, and, once accessed, is controlled by *commands*. A summary of these control statements and commands is presented in table 1-1. Included is the command or control statement *name*, its *purpose*, and the *page number* in this manual where a complete description of the command or control statement can be found. (Note that control statements are distinguished from commands by the $ sign.)

Table 1-1. TRACE/3000 Commands and Control Statements

COMMAND NAME	PURPOSE	PAGE
CHECK	Confirms correct application of program unit name and variable and array element name abbreviations.	2-41
DROP	Deletes PRINT and HALT commands or sentences from the PRINT/HALT table. (See Section II for a complete discussion of the PRINT/HALT table.)	2-40
GO	Starts or resumes execution of the program in an interactive session.	2-46
HALT	In an interactive session, HALT stops program execution and returns control to the user. In a batch job, HALT terminates program execution and flushes the remainder of the program from the system.	2-16
MODE	Displayed by TRACE/3000 in an interactive session to indicate that the INTERACTIVE file has been activated. (See Section II for a discussion of the INTERACTIVE file.)	2-12
PRINT	Used as the first command in a PRINT paragraph, PRINT causes TRACE/3000 to display information as defined in *sentences* following the PRINT command.	2-16
SET	1. Displays the current value of variables and array elements and, if so directed, changes the values. 2. Displays the relative addresses of variables and array elements according to their positions in the stack area. (See the *HP 3000 Computer System Reference Manual* for a discussion of the stack.)	2-42
$TRACE	Control statement. Used to inform MPE/3000 that the program unit identified in the $TRACE record will be monitored by TRACE/3000.	2-1
$TRACEEND	The last statement in the BATCH file, $TRACEEND closes the file.	2-3
$TRACESTART	$TRACESTART is used as the first statement in the BATCH file and opens the file.	2-3

Computer Museum

1-5. STRUCTURE POINTS

During execution of a source program, TRACE/3000 monitors program *structure points* if so directed by specific commands.

A program *structure point*, if defined for a program unit that calls another program unit (such as a routine), is the point at which the *call* to the routine is made, or the point at which control *returns* from the routine. If defined for a program unit being called, a *structure point* is the point at which control *enters* this program unit, or the point where control *exits* this program unit.

Thus, TRACE/3000 recognizes four structure points:

Call	A point at which a program unit *calls* another program unit.
Enter	The point where execution *enters* a program unit.
Exit	The point where execution *exits* a program unit.
Return	The point in the calling program unit where execution *returns* from a called program unit.

TRACE/3000 COMMANDS

SECTION II

TRACE/3000 execution is initiated by a $TRACE *control statement* inserted in the program to be monitored. During execution of the program, TRACE/3000 is controlled by *commands* which are entered interactively or contained in a file.

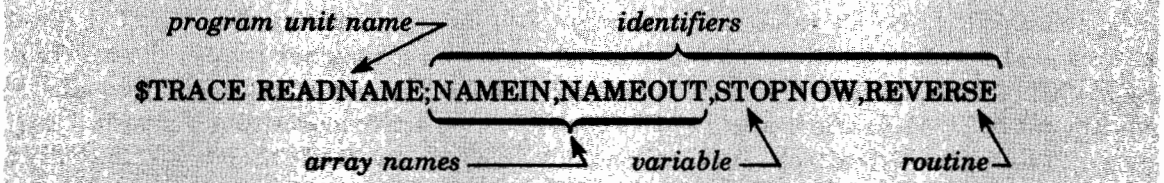
2-1. $TRACE CONTROL STATEMENT

To monitor a source program unit, and identifiers within the program unit, the $TRACE control statement is used. (A program unit is the main program unit or any routine operating with the main program.)

The form of the $TRACE control statement is

$TRACE[*program unit name*];*identifier*,*identifier*, . . . , *identifier*

For example,



where

program unit name is the name of the program unit (main program or routine) to which this $TRACE control statement pertains. You may use MAIN' (in FORTRAN/3000) or OB' (in SPL/3000) to signify the main program unit if this unit has no name. The *program unit name* parameter can be omitted from the $TRACE control statement, in which case TRACE/3000 assumes the main program unit. (If you want TRACE/3000 to monitor any program unit except the main program unit, however, you must specify the program unit name.)

identifier is the name of a variable, array, or routine which is contained in the program unit identified by *program unit name*. If you specify an *identifier* that is not contained in this program unit, TRACE/3000 displays an error message. If $DELETE is specified as the first item in the *identifier* list, the compiler suppresses all calls to TRACE/3000 to monitor any identifiers except those explicitly identified in the *identifier* list.

During compilation of a program referenced in a $TRACE control statement, the source language compiler inserts calls to TRACE/3000 to monitor all labels and structure points during execution. For example, if a program unit contains calls to subroutines A, B, and C, TRACE/3000 would monitor these procedure calls even if they were *not* identified in the *identifier* list of the $TRACE control statement. Using $DELETE as the first item in the *identifier* list, however, suppresses this monitoring and allows TRACE/3000 to monitor only those items which are specifically identified in the $TRACE control statement.

In a $TRACE control statement, $TRACE starts in column 1. The *program unit name*, if present, is separated from $TRACE by one or more spaces. If the program unit name is omitted, the semi-colon (;) is separated from $TRACE by one or more spaces. Spaces can be used freely within the *identifier* list, but not within specific identifier names. The examples below show legal $TRACE records.

```
$TRACE MAIN' ;      MAX,MIN, VAL
$TRACE     ;      A,  B,  C,  D
$TRACE ;      ALPHA,        OMEGA
$TRACE  ; $DELETE,  IFIX,10,   100,   PROG11
```

2-2. TRACE/3000 FILES AND TABLES

TRACE/3000 uses a BATCH file (see paragraph 2-3), an INTERACTIVE file (see paragraph 2-4), and a PRINT/HALT table (see paragraph 2-5) during execution. The BATCH file is always used during a batch job and is optional during an interactive session. The INTERACTIVE file is always used during an interactive session and cannot be used during a batch job. The PRINT/HALT table is used in both batch and interactive modes, and is merely a table used by TRACE/3000 to consolidate all commands from both the BATCH and INTERACTIVE files.

2-3. BATCH FILE

In a batch job, a BATCH file must exist for any program unit to be monitored by TRACE/3000. The use of a BATCH file in an interactive session is optional. When TRACE/3000 begins operation during an interactive session, it displays

```
BATCHFILE=
```

If you do not wish to use a BATCH file, press RETURN, or its equivalent. (See Section III for a discussion of TRACE/3000 operation.)

The first record in a BATCH file must be a $TRACESTART control statement. An optional body of PRINT and HALT *paragraphs* follows the $TRACESTART statement and the last record in the BATCH file must be a $TRACEEND statement. PRINT and HALT commands are the only commands allowed in a BATCH file; DROP, SET, CHECK, and GO commands may not be used.

Note: *Paragraph* is the name used to signify a list of TRACE/3000 commands and identifiers. For example,

```
$TRACESTART
PRINT OB'  }
INFILE     }   PRINT paragraph
LEN1       }

HALT       }
STOP       }   HALT paragraph
```

In the BATCH file shown above, the PRINT OB' command starts a PRINT *paragraph*, INFILE and LEN1 are identifiers and are called *sentences* of the paragraph. The HALT command starts a HALT paragraph; STOP identifies a statement label at which execution will halt and is called a sentence of the HALT paragraph.

The form of the $TRACESTART statement is

$TRACESTART [ABORT]

The $TRACESTART statement must begin in column 1. The optional parameter ABORT informs TRACE/3000 to terminate the program being monitored if the PRINT/HALT table overflows (becomes too large) or if TRACE/3000 discovers an error in a PRINT or HALT paragraph.

If the ABORT parameter is not used, TRACE/3000 ignores the error (and the rest of the paragraph) or the PRINT/HALT table overflow condition and continues executing.

PRINT and HALT paragraphs constitute the body of the BATCH file. The BATCH file can contain as many PRINT and HALT paragraphs as the size of the PRINT/HALT table will allow. If the BATCH file contains more PRINT and HALT paragraphs than the PRINT/HALT table can hold, TRACE/3000 displays an OVERFLOW message and terminates the program if the ABORT paragrameter was specified in the $TRACESTART statement, or ignores the paragraph and continues execution of the program if ABORT was not specified.

Each line of a PRINT or HALT paragraph must exist on a separate record and must start in column 1 of the record. The first PRINT or HALT paragraph in the BATCH file must contain a program unit name.

The last record in the BATCH file must be a $TRACEEND statement.

The form of the $TRACEEND statement is

$TRACEEND

The $TRACEEND statement must begin in column 1, must be the last record in a BATCH file and must be separated from the next-to-last record by a blank record. Failure to observe the proper form for BATCH file records causes TRACE/3000 to abort the program and display the message

BAD TRACE FILE

Examples of BATCH files are shown in figures 2-1 through 2-3.

In figure 2-1, the first record is $TRACESTART. The PRINT OB' command starts a PRINT paragraph and identifies the program unit (OB') that is to be monitored. The next record is a PRINT paragraph sentence and consists of a $FORM command. The $FORM command causes a listing of structure points encountered during program execution. INFILE and LEN1 also are sentences of the PRINT paragraph and identify variables which will be monitored by TRACE/3000. Each variable and its current value will be displayed by TRACE/3000 when the variable is encountered during program execution.

Note: Unless the $DELETE parameter is used in the $TRACE control statement in the source program, the $FORM command causes all structure points (for example, calls to subroutines, procedures, and function subprograms) to be monitored.

The HALT command starts a HALT paragraph. The next line (STOP) is a sentence of the HALT paragraph and informs TRACE/3000 to halt at the statement labelled STOP. If the HALT command is omitted from the BATCH file (as in figure 2-2), the program will not halt until the end of the program is reached.

```
$TRACESTART
PRINT OB'
$FORM
INFILE
LEN1

HALT
STOP

$TRACEEND
```

Figure 2-1. BATCH File BATCH1

```
$TRACESTART
PRINT OB'
$FORM
INFILE
LEN1

$TRACEEND
```

Figure 2-2. BATCH File BATCH2

```
$TRACESTART
PRINT OB'
$LABEL

$TRACEEND
```

Figure 2-3. BATCH File BATCH3

Figure 2-3 illustrates a BATCH file using the $LABEL command as the first sentence of the PRINT paragraph instead of the $FORM command used in figures 2-1 and 2-2. The $LABEL command causes TRACE/3000 to display all labels encountered when the program executes.

Figure 2-4 contains a short SPL program which reads a file and displays the contents of the file on the standard list device. The $TRACE control statement (see the second statement of the program) causes the compiler to invoke the TRACE/3000 subsystem when the :RUN command is entered. Figure 2-5 shows the output when the program is run. TRACE/3000 displays a "HELLO TRACE" message, then displays

BATCHFILE=

(the program was run interactively). Pressing the return key informs TRACE/3000 that there is no BATCH file. TRACE/3000 then activates the INTERACTIVE file and displays

MODE=

to ask for the mode (NORMAL or RESTRICTED, see paragraph 2-4). The response, N for NORMAL, causes TRACE/3000 to display an asterisk as a prompt for the first TRACE/3000 command of the INTERACTIVE file.

The first command entered is

*PRINT OB'

which starts a PRINT paragraph. The next two lines, INFILE and LEN1, identify variables which are to be monitored by TRACE/3000. A carriage return, which is the equivalent of entering a blank record, terminates the PRINT paragraph and TRACE/3000 displays an asterisk to prompt for the next command. The GO command de-activates the INTERACTIVE file and starts program execution.

```
:SPLPREP TRACE1

PAGE 0001    HP32100A.05.1

          00000 0    $CONTROL USLINIT
          00000 0    $TRACE OB';INFILE,LEN1,FOPEN,FREAD,PRINT,FCLOSE
          00000 0    << SPL EXAMPLE >>
          00000 0      BEGIN
          00000 1        BYTE ARRAY MAILLIST(0:8):="MAILLIST ";
          00006 1        BYTE ARRAY ERRBUF(0:1);
          00006 1        INTEGER ARRAY ERROUT(*)=ERRBUF;
          00006 1        ARRAY FOPENERR(0:5):="FOPEN ERROR ";
          00006 1        ARRAY FREADERR(0:5):="FREAD ERROR ";
          00006 1        ARRAY CHKERR(0:5):="FCHECK ERROR";
          00006 1        ARRAY OKCLO(0:11):="FILE CLOSED SUCCESSFULLY";
          00014 1        ARRAY FCERR(0:5):="FCLOSE ERROR";
          00006 1        ARRAY BUF(0:39);
          00006 1        INTEGER INFILE,LEN1,LEN2,ERRCODE;
          00006 1        INTRINSIC FOPEN,FCHECK,FREAD,FCLOSE;
          00006 1        INTRINSIC PRINT,ASCII;
          00006 1        INFILE:=FOPEN(MAILLIST,%605,%305);
          00037 1        IF < THEN GOTO OPENCHECK;
          00040 1    DISPLAY:
          00046 1        LEN1:=FREAD(INFILE,BUF,40);
          00070 1        IF < THEN GOTO READERR;
          00071 1        IF > THEN GOTO CLOSE;
          00072 1        PRINT(BUF,LEN1,0);
          00136 1        GOTO DISPLAY;
          00123 1    OPENCHECK:
          00131 1        PRINT(FOPENERR,6,0);
          00145 1        FCHECK(0,ERRCODE);
          00152 1        IF < THEN GOTO CHECKERR;
          00153 1        LEN2:=ASCII(ERRCODE,10,ERRBUF);
          00161 1        PRINT(ERROUT,LEN2,0);
          00175 1        GOTO STOP;
          00203 1    READERR:
          00211 1        PRINT(FREADERR,6,0);
          00225 1    CLOSE:
          00233 1        FCLOSE(INFILE,0,0);
          00246 1        IF < THEN GOTO CLOERR;
          00247 1        PRINT(OKCLO,12,0);
          00263 1        GOTO STOP;
          00272 1    CLOERR:
          00300 1        PRINT(FCERR,6,0);
          00314 1        GOTO STOP;
          00321 1    CHECKERR:
          00327 1        PRINT(CHKERR,6,0);
          00343 1    STOP:
          00351 1      END.
 PRIMARY DB STORAGE=%015;    SECONDARY DB STORAGE=%00244
 NO. ERRORS=000;             NO. WARNINGS=000
 PROCESSOR TIME=0:00:04;     ELAPSED TIME=0:04:00

 END OF COMPILE

 END OF PREPARE
:SAVE $OLDPASS,PROG
```

Figure 2-4. SPL/3000 Sample Program, TRACE1

```
:RUN PROG

HELLO TRACE  HP32222A.02.2
BATCHFILE= return
MODE=N

*PRINT 0B'
INFILE
LEN1

*GO
INFILE=      3
LEN1=     30
LOIS       ANYONE     6190 COURT ST.       METROPOLIS      NY
LEN1=     30
KING       ARTHUR     329 EXCALIBUR ST.    CAMELOT         CA
LEN1=     30
ALI        BABA       40 THIEVES WAY       SESAME          CO
LEN1=     30
JOHN       BIGTOWN    965 APPIAN WAY       METROPOLIS      NY
LEN1=     30
KNEE       BUCKLER    974 FISTICUFF DR.    PUGILIST        ND
LEN1=     30
SWASH      BUCKLER    497 PLAYACTING CT.   MOVIETOWN       CA
LEN1=     30
JAMES      DOE        4193 ANY ST.         ANYTOWN         MD
LEN1=     30
JANE       DOE        3959 TREEWOOD LN.    BIGTOWN         MA
LEN1=     30
JOHN       DOUGHE     239 MAIN ST.         HOMETOWN        MA
LEN1=     30
JENNA      GRANDTR    493 TWENTIETH ST.    PROGRESSIVE     CA
LEN1=     30
KARISSA    GRANDTR    7917 BROADMOOR WAY   BIGTOWN         MA
LEN1=     30
SPACE      MANN       9999 GALAXY WAY      UNIVERSE        CA
LEN1=      0
FILE CLOSED SUCCESSFULLY

BYE TRACE

 END OF PROGRAM
```

Figure 2-5. Output Generated by Program TRACE1

TRACE/3000 displays

```
INFILE=  3
LEN1=  30
```

then the first record of the file (MAILLIST) being read is displayed on the terminal. The value of LEN1 is displayed each time a record is read from the file. After the last record is read, TRACE/3000 displays

```
LEN1=  0
```

and, at the end of program execution, displays

```
BYE TRACE
```

Figure 2-6 illustrates the results of executing the program shown in figure 2-4 and specifying BATCH1 (see figure 2-1) as the BATCH file. Note that in addition to displaying values for INFILE and LEN1, TRACE/3000 displays the following structure points:

```
CALL FOPEN
RETURN FOPEN
CALL FREAD
RETURN FREAD
CALL PRINT
RETURN PRINT
CALL FCLOSE
RETURN FCLOSE
```

The above structure points are displayed because of the $FORM command in BATCH file BATCH1.

The structure points CALL FOPEN, RETURN FOPEN (when the file is opened) and CALL FCLOSE and RETURN FCLOSE (when the file is closed) are displayed only once. CALL FREAD, RETURN FREAD, and CALL PRINT, RETURN PRINT, however, are displayed each time a record is read from the file and listed on the terminal.

```
:RUN PROG

HELLO TRACE  HP32222A.02.2
BATCHFILE=BATCH1


MODE=N

*GO
CALL FOPEN
RETURN FOPEN
INFILE=     4
CALL FREAD
RETURN FREAD
LEN1=    30
CALL PRINT
LOIS       ANYONE     6190 COURT ST.      METROPOLIS    NY
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN1=    30
CALL PRINT
KING       ARTHUR     329 EXCALIBUR ST.   CAMELOT       CA
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN1=    30
CALL PRINT
ALI        BABA       40 THIEVES WAY      SESAME        CO
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN1=    30
CALL PRINT
JOHN       BIGTOWN    965 APPIAN WAY      METROPOLIS    NY
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN1=    30
CALL PRINT
KNEE       BUCKLER    974 FISTICUFF DR.   PUGILIST      ND
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN1=    30
CALL PRINT
SWASH      BUCKLER    497 PLAYACTING CT.  MOVIETOWN     CA
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN1=    30
CALL PRINT
JAMES      DOE        4193 ANY ST.        ANYTOWN       MD
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN1=    30
CALL PRINT
JANE       DOE        3959 TREEWOOD LN.   BIGTOWN       MA
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN1=    30
CALL PRINT
JOHN       DOUGHE     239 MAIN ST.        HOMETOWN      MA
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN1=    30
CALL PRINT
JENNA      GRANDTR    493 TWENTIETH ST.   PROGRESSIVE   CA
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN1=    30
CALL PRINT
KARISSA    GRANDTR    7917 BROADMOOR WAY  BIGTOWN       MA
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN1=    30
CALL PRINT
SPACE      MANN       9999 GALAXY WAY     UNIVERSE      CA
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN1=     0
CALL FCLOSE
RETURN FCLOSE
CALL PRINT
FILE CLOSED SUCCESSFULLY
RETURN PRINT
STOP
OB'

*GO

BYE TRACE

 END OF PROGRAM
```

Figure 2-6. Output Generated by BATCH File BATCH1

After the last record has been read and the file has been closed successfully, TRACE/3000 halts at the statement labelled STOP and displays

STOP
OB'

to show the label (STOP) and the program unit (OB') at which the halt occurred. TRACE/3000 then re-activates the INTERACTIVE file and displays an asterisk to prompt for another command. The GO command starts program execution at the statement following the point where the program halted (an END statement in this case) and program execution terminates.

Figure 2-7 shows the use of BATCH file BATCH2. Operation is identical to that of figure 2-6 except that TRACE/3000 does not halt operation at statement STOP (there is no HALT command in BATCH file BATCH2).

Figure 2-8 illustrates the use of BATCH3. Note that the variables INFILE and LEN1 are not displayed. Instead, the $LABEL command in BATCH file BATCH3 causes TRACE/3000 to display the label DISPLAY each time it is encountered until the last record is read. TRACE/3000 then displays the labels

DISPLAY
CLOSE
STOP

```
:RUN PROG

HELLO TRACE  HP32222A.02.2
BATCHFILE=BATCH2

MODE=N

*GO
CALL FOPEN
RETURN FOPEN
INFILE=     4
CALL FREAD
RETURN FREAD
LEN:=    30
CALL PRINT
LOIS       ANYONE     6198 COURT ST.      METROPOLIS     NY
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN:=    30
CALL PRINT
KING       ARTHUR     329 EXCALIBUR ST.   CAMELOT        CA
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN:=    30
CALL PRINT
ALI        BABA       40 THIEVES WAY      SESAME         CO
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN:=    30
CALL PRINT
JOHN       BIGTOWN    965 APPIAN WAY      METROPOLIS     NY
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN:=    30
CALL PRINT
KNEE       BUCKLER    974 FISTICUFF DR.   PUGILIST       ND
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN:=    30
CALL PRINT
SWASH      BUCKLER    497 PLAYACTING CT.  MOVIETOWN      CA
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN:=    30
CALL PRINT
JAMES      DOE        4193 ANY ST.        ANYTOWN        MD
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN:=    30
CALL PRINT
JANE       DOE        3959 TREEWOOD LN.   BIGTOWN        MA
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN:=    30
CALL PRINT
JOHN       DOUGHE     239 MAIN ST.        HOMETOWN       MA
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN:=    30
CALL PRINT
JENNA      GRANDTR    493 TWENTIETH ST.   PROGRESSIVE    CA
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN:=    30
CALL PRINT
KARISSA    GRANDTR    7917 BROADMOOR WAY  BIGTOWN        MA
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN:=    30
CALL PRINT
SPACE      MANN       9999 GALAXY WAY     UNIVERSE       CA
RETURN PRINT
CALL FREAD
RETURN FREAD
LEN:=     0
CALL FCLOSE
RETURN FCLOSE
CALL PRINT
FILE CLOSED SUCCESSFULLY
RETURN PRINT

BYE TRACE

 END OF PROGRAM
```

Figure 2-7. Output Generated by BATCH File BATCH2

```
:RUN PROG

HELLO TRACE  HP32222A.02.2
BATCHFILE=BATCH3

MODE=N

*GO
DISPLAY
LOIS        ANYONE      6190 COURT ST.       METROPOLIS      NY
DISPLAY
KING        ARTHUR      329 EXCALIBUR ST.    CAMELOT         CA
DISPLAY
ALI         BABA        40 THIEVES WAY       SESAME          CO
DISPLAY
JOHN        BIGTOWN     965 APPIAN WAY       METROPOLIS      NY
DISPLAY
KNEE        BUCKLER     974 FISTICUFF DR.    PUGILIST        ND
DISPLAY
SWASH       BUCKLER     497 PLAYACTING CT.   MOVIETOWN       CA
DISPLAY
JAMES       DOE         4193 ANY ST.         ANYTOWN         MD
DISPLAY
JANE        DOE         3959 TREEWOOD LN.    BIGTOWN         MA
DISPLAY
JOHN        DOUGHE      239 MAIN ST.         HOMETOWN        MA
DISPLAY
JENNA       GRANDTR     493 TWENTIETH ST.    PROGRESSIVE     CA
DISPLAY
KARISSA     GRANDTR     7917 BROADMOOR WAY   BIGTOWN         MA
DISPLAY
SPACE       MANN        9999 GALAXY WAY      UNIVERSE        CA
DISPLAY
CLOSE
FILE CLOSED SUCCESSFULLY
STOP

BYE TRACE

 END OF PROGRAM
```

Figure 2-8. Output Generated by BATCH File BATCH3

The examples in figures 2-4 through 2-8 all were run in interactive mode. See Section III for more complete descriptions of running TRACE/3000 in both the interactive and batch modes.

2-4. INTERACTIVE FILE

TRACE/3000 opens the INTERACTIVE file *only* if the source program is running in an interactive session.

The INTERACTIVE file accepts PRINT, HALT, SET, DROP, CHECK, and GO commands (unlike the BATCH file, which accepts only PRINT and HALT commands).

TRACE/3000 activates the INTERACTIVE file under the following three conditions only:

1. Before starting execution of the source program.
2. After a HALT command is executed.
3. After CONTROL Y is used.

In the first case above, TRACE/3000 indicates that the INTERACTIVE file has been activated by displaying

```
MODE=
```

You must answer N (for NORMAL) or R (for RESTRICTED). NORMAL mode gives you full use of all TRACE/3000 facilities, while RESTRICTED mode places two restraints on TRACE/3000 operation, as follows:

1. You cannot use a label with a GO command (see paragraph 2-23).

2. You can modify the value of only simple variables by entering SET paragraphs into the INTERACTIVE file (you cannot modify the values of arrays or array elements in RESTRICTED mode). See paragraph 2-19 for a discussion of the SET command.

After you have entered N or R, TRACE/3000 displays an asterisk in column 1 of the line on the terminal. Now you can enter commands and paragraphs.

In the second and third cases (after a HALT command has been executed or CONTROL Y has been used), TRACE/3000 does not display MODE=, but displays an asterisk in column 1 of the line to indicate that the INTERACTIVE file is ready to accept more commands. Again, you now may enter commands and paragraphs.

After you have entered all required commands and paragraphs, de-activate the INTERACTIVE file and return control to the program by entering the GO command (see paragraph 2-23).

An example of INTERACTIVE file usage is shown in figure 2-9. The same source program is used as in figure 2-4. This time, however, all commands are entered through the INTERACTIVE file. The PRINT command is entered to start a PRINT paragraph and the sentence $LABEL is entered after the PRINT command. The end of the PRINT paragraph is signalled by pressing RETURN (thus entering a blank record into the INTERACTIVE file). When TRACE/3000 again displays an asterisk, the GO command is entered and TRACE/3000 monitors and displays all labels. Note that procedure calls and returns are not displayed because the $FORM command was not used in the PRINT paragraph.

```
:RUN PROG

HELLO TRACE  HP32222A.02.2
BATCHFILE= return
MODE=N

*PRINT OB'
$LABEL

*GO
DISPLAY
LOIS        ANYONE     6190 COURT ST.        METROPOLIS      NY
DISPLAY
KING        ARTHUR     329 EXCALIBUR ST.     CAMELOT         CA
DISPLAY
ALI         BABA       40 THIEVES WAY        SESAME          CO
DISPLAY
JOHN        BIGTOWN    965 APPIAN WAY        METROPOLIS      NY
DISPLAY
KNEE        BUCKLER    974 FISTICUFF DR.     PUGILIST        ND
DISPLAY
SWASH       BUCKLER    497 PLAYACTING CT.    MOVIETOWN       CA
DISPLAY
JAMES       DOE        4193 ANY ST.          ANYTOWN         MD
DISPLAY
JANE        DOE        3959 TREEWOOD LN.     BIGTOWN         MA
DISPLAY
JOHN        DOUGHE     239 MAIN ST.          HOMETOWN        MA
DISPLAY
JENNA       GRANDTR    493 TWENTIETH ST.     PROGRESSIVE     CA
DISPLAY
KARISSA     GRANDTR    7917 BROADMOOR WAY    BIGTOWN         MA
DISPLAY
SPACE       MANN       9999 GALAXY WAY       UNIVERSE        CA
DISPLAY
CLOSE
FILE CLOSED SUCCESSFULLY
STOP

BYE TRACE

 END OF PROGRAM
```

Computer Museum

Figure 2-9. Using the INTERACTIVE File

See Section III for a further discussion of the use of the INTERACTIVE file during TRACE/3000 operation.

2-5. PRINT/HALT TABLE

The PRINT/HALT table consists of PRINT and HALT sentences entered into the BATCH *or* INTERACTIVE files. TRACE/3000 consolidates *all* PRINT and HALT sentences from both files into the PRINT/HALT table.

During program execution, TRACE/3000 searches the PRINT/HALT table for any PRINT and HALT sentences applying to program identifiers appearing in $TRACE control records in the program, and takes the appropriate action. For example, the $TRACE control statement in the source program

```
$TRACE OB';INFILE,LEN1,FOPEN,FREAD,PRINT,FCLOSE
```

and the PRINT sentences

```
$FORM
INFILE
LEN1
```

cause TRACE/3000 to monitor the program structure points (calls and returns from procedures) and to monitor the variables INFILE and LEN1 and display the values of these variables whenever they are encountered during program execution. Note that the identifiers must appear in the $TRACE control statement *and* in the PRINT paragraph.

During an interactive session, PRINT and HALT sentences can be added or deleted from the PRINT/HALT table once program execution begins by modifying the INTERACTIVE file.

See Section III for a discussion of PRINT/HALT table size and using the INTERACTIVE file to modify the PRINT/HALT table.

2-6. PARAGRAPHS

TRACE/3000 recognizes five paragraph types: PRINT, HALT, DROP, SET, and CHECK. PRINT and HALT paragraphs can be entered into both the BATCH and INTERACTIVE files; while DROP, SET, and CHECK paragraphs can be entered into the INTERACTIVE file only. Paragraphs entered into the BATCH file differ slightly from paragraphs entered into the INTERACTIVE file (TRACE/3000 prompts with an asterisk for paragraphs in the INTERACTIVE file whereas the asterisk is not used in the BATCH file), but basically BATCH and INTERACTIVE paragraphs are structured in the same way.

Each paragraph starts with a record containing the paragraph type: PRINT, HALT, DROP, SET, or CHECK. The paragraph type is optionally followed by one or more blanks and a program unit name (see paragraph 2-7). For the BATCH file, the paragraph type starts in column 1 of the record. For the INTERACTIVE file, TRACE/3000 displays an asterisk in column 1 to indicate its readiness to accept paragraphs. The paragraph type is then started in column 2 of the line.

The paragraph sentences follow the paragraph type on the next record/line. Each sentence starts in column 1 of the record/line. You can enter as many sentences as you wish, subject to the size of the PRINT/HALT table (for PRINT/HALT paragraphs). Sentences must be of the proper paragraph type and form.

The paragraph end is signified in the BATCH file by a blank record. You then can insert another paragraph immediately following the blank record. For the INTERACTIVE file,

paragraph end is signified by a carriage return in column 1 of the current line. TRACE/3000 responds by typing an asterisk in column 1 of the next line to prompt for the next paragraph.

A PRINT paragraph is shown in BATCH file form in figure 2-10 and in INTERACTIVE file form in figure 2-11.

2-7. PROGRAM UNIT NAME IN PARAGRAPHS

TRACE/3000 associates a name of a program unit with each paragraph in either the BATCH or INTERACTIVE file. All the sentences within the paragraph apply to the program unit associated with the paragraph. Only those program unit names mentioned in $TRACE control records in program units can be used in paragraphs. If the main program has no name, the compiler assigns the name MAIN' (for FORTRAN/3000 programs) or OB' (for SPL/3000 programs).

The first paragraph of the BATCH and INTERACTIVE files must contain a program unit name. If no program unit name appears in the first paragraph, TRACE/3000 treats the paragraph as having incorrect form, and displays a BAD SYNTAX error message.

In each paragraph following the first one, you can include a program unit name along with the paragraph type. If a program unit name is included, TRACE/3000 assigns that name to the paragraph. If you do not include a program unit name with the paragraph type, TRACE/3000 assigns the program unit name of the paragraph immediately preceding the current paragraph. This means that only the first paragraph in the BATCH and INTERACTIVE files must have a program unit name. All other paragraphs can omit the program unit name. Those paragraphs then will apply to the same program unit as the first paragraph.

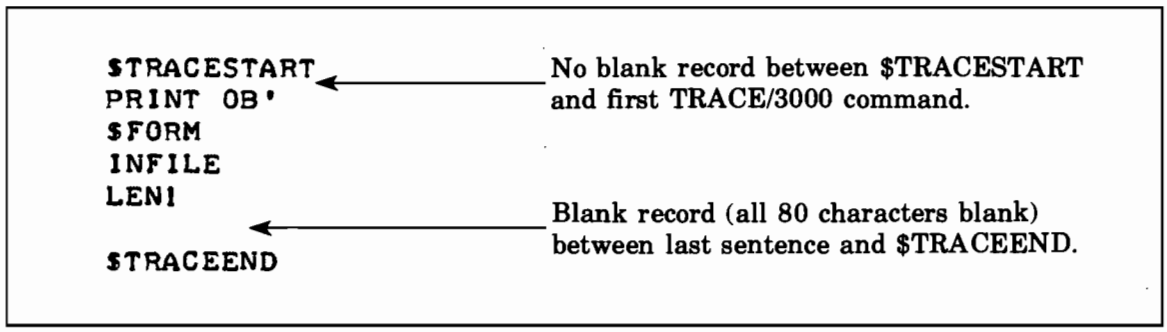


Figure 2-10. BATCH File PRINT Paragraph Example

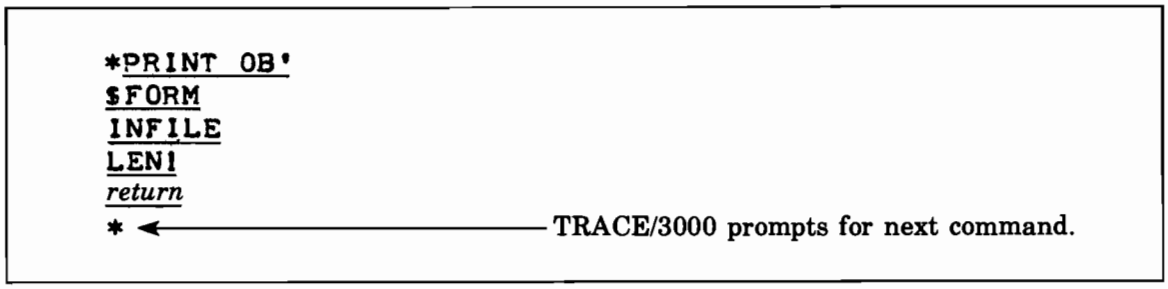


Figure 2-11. INTERACTIVE File PRINT Paragraph Example

2-8. PRINT AND HALT COMMANDS AND PARAGRAPHS

PRINT and HALT commands are used as the first commands in PRINT and HALT paragraphs. The PRINT and HALT commands are entered into the BATCH or INTERACTIVE files and various parameters, or *sentences*, are added after the PRINT and HALT commands to form *paragraphs*.

PRINT and HALT sentences are used to request the displaying of program data and logic flow during program execution.

The form of the PRINT command is

PRINT [*program unit name*]

For example,

PRINT TRACE1

program unit name

where

program unit name is the name of the program unit to which the paragraph will apply. The first PRINT command in the BATCH and INTERACTIVE files must contain a *program unit name*. All other PRINT commands need not contain a *program unit name* unless the command applies to a program unit other than the one in the preceding paragraph.

The form of the HALT command is

HALT[*program unit name*]

For example,

HALT TRACE1

where

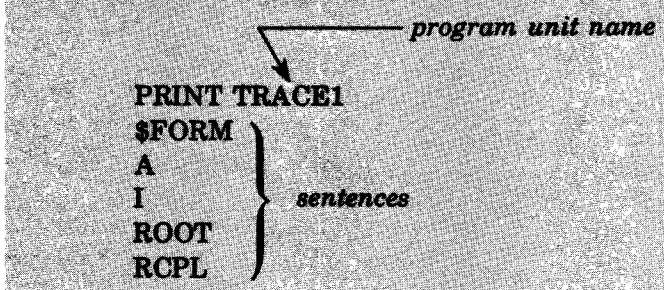
program unit name is the same as described for the PRINT command.

The PRINT and HALT commands are used to start PRINT and HALT paragraphs. After the PRINT or HALT command, parameters, or sentences, are added to inform TRACE/3000 what action it is to take.

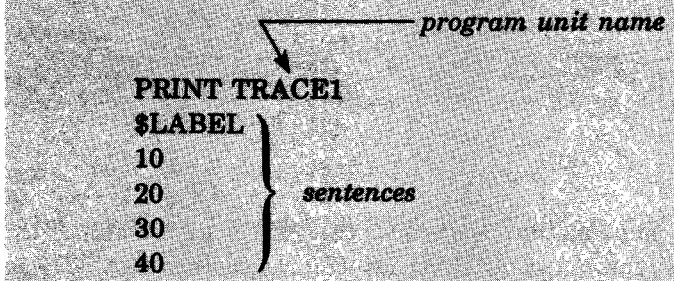
The form of a PRINT paragraph is

```
PRINT [program unit name]
sentence
sentence
sentence
.
.
sentence
blank record terminator
```

For example,



or



where

program unit name is the same as described for the PRINT command. The first paragraph in the BATCH and INTERACTIVE files must contain a *program unit name*. All other paragraphs need not contain a *program unit name* unless the paragraph applies to a program unit other than the one in the preceding paragraph.

sentence consists of an identifier name, optionally followed by one or more *condition clauses* (see paragraph 2-9) allowed for the identifier type (array, variable, label, or routine). The $FORM and $LABEL commands can be used as *sentences* and can be modified by condition clauses. $FORM (if unmodified by condition clauses) gives a complete listing of structure points during program execution (unless $DELETE was specified in the $TRACE control statement, in which

case only those structure points included in the $TRACE control statement are listed). $LABEL (if unmodified by condition clauses) gives a complete listing of the labels encountered during program execution (unless $DELETE was specified in the $TRACE control statement, in which case only those labels specified in the $TRACE control statement are listed). See paragraph 2-15 for a discussion of PRINT sentence execution.

blank record terminator — is a blank record (all 80 columns blank) for the BATCH file, or a carriage return in column 1 of the line for the INTERACTIVE file.

The form of a HALT paragraph is

```
HALT program unit name
sentence
sentence
.
.
sentence
blank record terminator
```

For example,

```
HALT   TRACE1
40
```

where

program unit name — is the same as that defined for the PRINT paragraph. If the HALT command is the first command in a BATCH or INTERACTIVE file, *program unit name* must be specified, or if the program unit in which the halt is to be executed is different that the program unit specified in the preceding paragraph, *program unit name* must be specified.

sentence — consists of an identifier name, optionally followed by one or more condition clauses. Every *sentence* in a HALT paragraph will cause program execution to stop once the identifier specified in the paragraph is reached. For example, if $LABEL (with no condition clauses) is used as a sentence, program execution will stop at the first label encountered. If $FORM is used, program execution will stop at the first program structure point that is encountered. If a HALT sentence consists of a variable, program execution will stop when the variable is encountered in the program unit. See paragraph 2-15 for a discussion of HALT sentence execution.

blank record terminator — is the same as described for the PRINT paragraph.

2-9. SENTENCE CONDITION CLAUSES

Identifiers used in sentences in PRINT and HALT paragraphs can be modified by *condition clauses*. There are five types of condition clauses, as follows:

- *Subscript value condition clause (S1)*. This condition clause, abbreviated as S1, is used only with array-type identifiers. See paragraph 2-10.
- *Identifier value condition clause (I1)*. The I1 condition clause can be used with array and variable-type identifiers. See paragraph 2-11.
- *Label condition clause (L1)*. The L1 condition clause can be used with all identifier types. See paragraph 2-12.
- *Use condition clause (U1)*. The U1 condition clause can be used with all identifier types. See paragraph 2-13.
- *Routine parameter clause (R1)*. The R1 condition clause is used only with routine-type identifiers. See paragraph 2-14.

Condition clauses, if used with identifiers, must appear in the order shown below. The clauses must be separated from the identifier, and from each other, by one or more blanks, although no blanks can exist within the condition clauses themselves.

array S1 I1 L1 U1

or

variable I1 L1 U1

or

label L1 U1

or

$LABEL L1 U1

or

routine R1 L1 U1

or

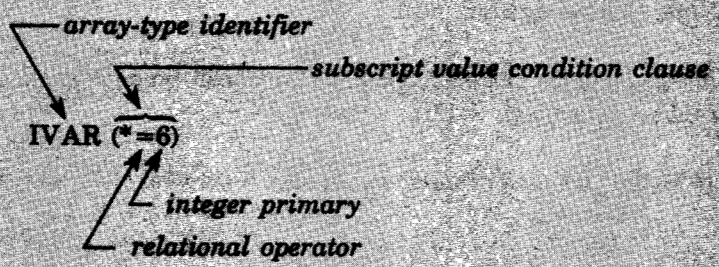
$FORM R1 L1 U1

2-10. SUBSCRIPT VALUE CONDITION CLAUSE. The subscript value condition clause is used only with array-type identifiers (identifier type is determined by its source program definition).

The form of the subscript value condition clause is

(relational operator integer primary)*

For example,



where

* represents the effective subscript in the source program. For example, in the subscript value condition clause (*=6), the value 6 represents the sixth element of the array identifier which is modified by the condition clause (IVAR in the example shown) and the asterisk causes TRACE/3000 to search for a reference to array element IVAR(6) in the source program, thus forming an expression that can be evaluated (if array element IVAR(6) is encountered during program execution, TRACE/3000 will display its value).

relational operator is one of the following relational operations:

= equal
< > not equal
> greater than
< less than
>= greater than or equal
<= less than or equal

integer primary is either a constant or variable of type integer. If a variable, it must have been included in a $TRACE control statement.

Examples of subscript value condition clauses used in PRINT and HALT paragraphs are shown below. The first paragraph appears as in a BATCH file; the second paragraph appears as in an INTERACTIVE file.

```
PRINT PROGRAM1
ARRAY1 (*=6)
ARRAY2 (*<MAXIMUM)
ARRAY1 (*>=34)
blank record terminator
```

```
*HALT PROGRAM1
ARRAY2 (*=36)
ARRAY1 (*=MIN)
return
*
```

Figures 2-12 and 2-13 illustrate subscript value condition clause usage.

Figure 2-12 is a sample FORTRAN/3000 source program, which is compiled and prepared into the program file XMPL2. The MPE/3000 :RUN command accesses TRACE/3000, which responds (the program was run interactively) with:

```
BATCHFILE=
```

A carriage return causes TRACE/3000 to use only the INTERACTIVE file and display

```
MODE=
```

An N response informs TRACE/3000 that the program will run in the NORMAL mode and TRACE/3000 prompts for a command by displaying an asterisk.

The HALT paragraph

```
*HALT TRACE2
70
```

informs TRACE/3000 to execute the program until statement label 70 is reached and to halt at that point. When statement label 70 is reached, the program halts, control is returned to the user, and TRACE/3000 displays

```
70
TRACE2
```

The SET paragraph (see paragraph 2-22)

```
*SET
STOPNOW=    0/1
```

causes TRACE/3000 to display the current value (0) of variable STOPNOW and the /1 enters a new value for this variable. The second GO command (now that STOPNOW is not equal to 0) causes the STOP and END statements to be executed and the program terminates.

```
:FORTPREP TRACE2,XMPL2

PAGE 0001    HP32102A.01.4


00001000   $CONTROL USLINIT
00002000   $TRACE TRACE2;I,J,IARR,STOPNOW
00003000   $TRACE SB;IVAR,L,M
00004000         PROGRAM TRACE2
00005000    100  FORMAT('0',T8,S//)
00006000    200  FORMAT(T5,5I4)
00007000         DIMENSION IARR(5,5)
00008000         CHARACTER A*10
00009000         INTEGER STOPNOW
00010000         STOPNOW=0
00011000    10   I=5
00012000    20   J=5
00013000    30   K=10
00014000    40   CALL SB(IARR,A,I,J,K)
00015000    50   WRITE(6,100)A
00016000    60   WRITE(6,200)IARR
00017000    70   CONTINUE
00018000         IF(STOPNOW.EQ.0)GOTO 10
00019000         STOP
00020000         END


**** NO ERRORS,  NO WARNINGS;   PROGRAM UNIT COMPILED  ****
COMPILATION TIME    1.138 SECONDS    ELAPSED TIME    76.285 SECONDS
00021000         SUBROUTINE SB(IVAR,Z,L,M,N)
00022000         DIMENSION IVAR(L,M)
00023000         CHARACTER Z*(N)
00024000    10   DO 30 NR=1,L
00025000    20   DO 30 NC=1,M
00026000    30   IVAR(NR,NC)=NR*NC
00027000         Z="THE START"
00028000         RETURN
00029000         END

**** NO ERRORS,  NO WARNINGS;   PROGRAM UNIT COMPILED  ****
COMPILATION TIME    0.682 SECONDS    ELAPSED TIME    43.420 SECONDS
TOTAL COMPILATION TIME  0:00:02
TOTAL ELAPSED TIME      0:02:13

 END OF COMPILE

 END OF PREPARE
:RUN XMPL2

HELLO TRACE  HP32222A.02.1
BATCHFILE= return
MODE=N

*HALT TRACE2
70

*GO


      THE START


      1   2   3   4   5
      2   4   6   8  10
      3   6   9  12  15
      4   8  12  16  20
      5  10  15  20  25
70
TRACE2

*SET
STOPNOW=      0/1

*GO

 END OF PROGRAM
```

Figure 2-12. FORTRAN/3000 Sample Program, TRACE2

```
:RUN XMPL2

HELLO TRACE  HP32222A.02.1
BATCHFILE= return
MODE=N

*HALT TRACE2
70

*PRINT SB
IVAR (*=6)

*GO
IVAR(6)=      2


      THE START


      1   2   3   4   5
      2   4   6   8  10
      3   6   9  12  15
      4   8  12  16  20
      5  10  15  20  25
70
TRACE2

*PRINT SB
IVAR (*=23)

*GO

IVAR(6)=      2
IVAR(23)=     15

      THE START


      1   2   3   4   5
      2   4   6   8  10
      3   6   9  12  15
      4   8  12  16  20
      5  10  15  20  25
70
TRACE2

*DROP
SB

*PRINT SB
IVAR (*<26)

*GO

IVAR(1)=      1
IVAR(6)=      2
IVAR(11)=     3
IVAR(16)=     4
IVAR(21)=     5
IVAR(2)=      2
IVAR(7)=      4
IVAR(12)=     6
IVAR(17)=     8
IVAR(22)=     10
IVAR(3)=      3
IVAR(8)=      6
IVAR(13)=     9
IVAR(18)=     12
IVAR(23)=     15
IVAR(4)=      4
IVAR(9)=      8
IVAR(14)=     12
IVAR(19)=     16
IVAR(24)=     20
IVAR(5)=      5
IVAR(10)=     10
IVAR(15)=     15
IVAR(20)=     20
IVAR(25)=     25

      THE START


      1   2   3   4   5
      2   4   6   8  10
      3   6   9  12  15
      4   8  12  16  20
      5  10  15  20  25
70
TRACE2

*SET TRACE2
STOPNOW=      0/1

*GO

 END OF PROGRAM
```

Figure 2-13. Subscript Value Condition Clause Usage

Figure 2-13 illustrates subscript value condition clause usage. Again, the HALT paragraph

```
*HALT TRACE2
70
```

informs TRACE/3000 to halt program execution at statement label 70. The PRINT paragraph

```
*PRINT SB
IVAR (*=6)
```

causes TRACE/3000 to monitor array IVAR in program unit SB. The (*=6) value condition clause informs TRACE/3000 that, if array element IVAR(6) is encountered during program execution, its value is to be displayed.

TRACE/3000 displays the value (2) of array element IVAR(6), executes the remainder of the program up to statement label 70, and halts at this point.

The next PRINT paragraph

```
*PRINT SB
IVAR (*=23)
```

adds element IVAR(23) to the array elements to be monitored and TRACE/3000 displays the value of IVAR(6) and IVAR(23), again halting at statement label 70.

The DROP command (see paragraph 2-17) is used to start the DROP paragraph

```
*DROP
SB
```

which deletes all sentences from the PRINT SB paragraphs and the new PRINT paragraph

```
*PRINT SB
IVAR (*<26)
```

is entered. The value condition clause (*<26) will cause TRACE/3000 to display the values of IVAR array elements as long as the subscript is less than 26. TRACE/3000, therefore, displays all elements of IVAR and the value of each element. When the program halts at statement label 70, the SET paragraph

```
*SET TRACE2
STOPNOW=    0/1
```

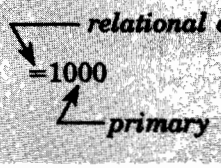
sets the value of STOPNOW to 1 and the next GO command causes program termination.

2-11. IDENTIFIER VALUE CONDITION CLAUSE. The identifier value condition clause is used with array and variable-type identifiers to test the value of a data element. The clause is separated from the subscript value condition clause by one or more blanks, or from the identifier by one or more blanks if a subscipt value condition clause is not used.

The form of the identifier value condition clause is

relational operator primary

For example,



where

relational operator	is the same as that defined for the subscript value condition clause.
primary	is either a variable or a constant.

The following examples (in a BATCH file) show how the identifier value condition clause joins with identifiers and subscript value condition clauses to form arithmetic expressions. If the arithmetic expression is true, then the identifier value condition clause is true.

```
PRINT
ARRAY1 (*=10)=MAXIMUM
ITEM >MINIMUM
ARRAY2 (*=24)=1000
blank record terminator
```

Figure 2-14 illustrates identifier value condition clause usage. The same program (TRACE2) is used as in figure 2-13.

The PRINT paragraph

```
*PRINT TRACE2
I <6
J >1
```

informs TRACE/3000 to monitor the variables I and J if the identifier value condition clauses are true. That is, if the value of I is less than 6 and the value of J is greater than 1.

The second PRINT paragraph

```
*PRINT SB
IVAR (*=13) =9
```

uses the *subscript value condition clause* (*=13) and the *identifier value condition clause* to modify the sentence. Thus, the sentence will execute *only* if both clauses are true. That is, if array element IVAR(13) has a value of 9.

The HALT paragraph instructs TRACE/3000 to halt at label 70 and the GO command de-activates the INTERACTIVE file and starts program execution. TRACE/3000 displays

```
I=   5
J=   5
IVAR(13)=   9
```

(the condition clauses were true), executes the remainder of the program, and halts at statement 70.

The DROP paragraph

```
*DROP SB
$ALL
```

deletes all PRINT sentences from the PRINT/HALT table for program unit SB. The new PRINT paragraph

```
*PRINT SB
IVAR (*=13)  =10
```

```
:RUN XMPL2

HELLO TRACE  HP32222A.02.1
BATCHFILE= return
MODE=N

*PRINT TRACE2
I  <6
J  >1

*PRINT SB
IVAR (*=13) =9

*HALT TRACE2
70

*GO
I=      5
J=      5
IVAR(13)=      9


       THE START


       1    2    3    4    5
       2    4    6    8   10
       3    6    9   12   15
       4    8   12   16   20
       5   10   15   20   25
70
TRACE2

*DROP SB
$ALL

*PRINT SB
IVAR (*=13) =10

*GO

I=      5
J=      5

       THE START


       1    2    3    4    5
       2    4    6    8   10
       3    6    9   12   15
       4    8   12   16   20
       5   10   15   20   25
70
TRACE2

*SET TRACE2
STOPNOW=      0/1

*GO

  END OF PROGRAM
```

Figure 2-14. Identifier Value Condition Clause Usage

changes the identifier value condition clause for IVAR element 13 (the subscript value condition clause (* = 13) is the same as previously). After the GO command is entered and program execution begins, TRACE/3000 displays

```
I=   5
J=   5
```

(these identifier value condition clauses were not changed) but does not display a value for IVAR(13) because the identifier value condition clause is false (array element IVAR(13) does not equal 10).

2-12. LABEL CONDITION CLAUSE. The label condition clause can be used with all identifier types. This condition clause describes an area within a program unit bounded by two labels. The area includes the source statement of the first label and all statements up to (but not including) the source statement of the second label. If the identifier in the identifier value condition clause falls between the two labels, the label condition clause is true. If the identifier is encountered after the second label or before the first label, the label condition clause is false. The label condition clause is separated from the preceding condition clause or the identifier itself (if no preceding condition clause is used) by one or more blanks.

The form of the label condition clause is

label-label

For example,

10-40 (for FORTRAN/3000 source programs)

or

LABEL1-LABEL2 (for SPL/3000 source programs)

where

label is either a label-type identifier or an asterisk. **-label* denotes "from the beginning" of the program unit to the second label (defined by *label*). *label-** denotes from the first label to the "end of program unit".

The second label must physically follow the first label in the program unit.

The following are examples (in a BATCH file) of label condition clauses:

```
PRINT EXON
ARRAY1 (*=10) =MAX  10-30
MAXVAL *-60
VALUE (*=8) 60-*
blank record terminator
```

Figures 2-15 and 2-16 illustrate label condition clause usage.

Figure 2-15 is a sample FORTRAN/3000 source program, TRACE3, which is compiled and prepared into program file XMPL3. The MPE/3000 :FILE command

```
:FILE FTN20=NAMES,OLD
```

equates FORTRAN/3000 logical unit number 20, used in statement 10 in the program, to the old file NAMES. See the *MPE/3000 Operating System Reference Manual* for a discussion of the :FILE command and the *FORTRAN/3000 Reference Manual* for a discussion of FORTRAN/3000 logical unit numbers.

The HALT paragraph

```
*HALT TRACE3
40
```

causes the program to halt at statement label 40. The SET paragraph

```
*SET
STOPNOW=    0/1
```

is used to set the integer variable STOPNOW equal to 1 so that when the second GO command is executed, the program terminates.

The program reads names from a file (NAMES), and reverses and displays the names.

Figure 2-16 illustrates label condition clause usage. Again, the HALT paragraph instructs TRACE/3000 to halt at statement label 40.

The PRINT paragraph

```
*PRINT REVERSE
K 40-60
OUT (*<8) 30-50
```

specifies label condition clauses for the variable K and the array OUT. When the GO command is entered, TRACE/3000 displays the value of OUT for all elements less than 8 which are bounded by labels 30 and 50 in program unit REVERSE. (The program statements between labels 30 and 50 write the last name into array OUT, so this is the information displayed in OUT array elements OUT(1) through OUT(7).) Also, the values of K occurring between the labels 40 and 60 are displayed.

2-13. USE CONDITION CLAUSE. The use condition clause can be used with all identifier types. This condition clause is separated from the preceding condition clause or from the identifier by one or more blanks. The use condition clause is referenced only if all the preceding condition clauses (if any) in the sentence are true.

The form of the use condition clause is

@integer primary

For example,

@44

```
:FORTPREP TRACE3

PAGE 0001    HP32102A.01.6


00001000  $CONTROL USLINIT
00002000  $TRACE TRACE3;NAMEIN,NAMEOUT,STOPNOW
00003000  $TRACE REVERSE;IN,OUT,I,J,K
00004000        PROGRAM TRACE3
00005000   100  FORMAT(T10,"NAME",T30,"LAST NAME FIRST"//)
00006000   200  FORMAT(20A1)
00007000   300  FORMAT(T7,20A1,T32,20A1)
00008000        CHARACTER NAMEIN(20),NAMEOUT(20)
00009000        INTEGER STOPNOW
00010000        STOPNOW=0
00011000   10   WRITE(6,100)
00012000   20   READ(20,200,END=40)NAMEIN
00013000   30   CALL REVERSE(NAMEIN,NAMEOUT)
00014000        WRITE(6,300)NAMEIN,NAMEOUT
00015000        GOTO 20
00016000   40   CONTINUE
00017000        IF(STOPNOW.NE.0)STOP
00018000        REWIND 20
00019000        GOTO 10
00020000   50   STOP
00021000        END


****  NO ERRORS,  NO WARNINGS;   PROGRAM UNIT COMPILED  ****
COMPILATION TIME    1.395 SECONDS    ELAPSED TIME    84.161 SECONDS
00022000        SUBROUTINE REVERSE(IN,OUT)
00023000        CHARACTER IN(20),OUT(20)
00024000  C
00025000  C FIND END OF FIRST NAME
00026000  C
00027000   10   DO 20 I=1,20
00028000        IF(IN(I).EQ." ")GOTO 30
00029000   20   CONTINUE
00030000   30   J=I+1
00031000  C
00032000  C WRITE LAST NAME INTO OUT
00033000  C
00034000        K=0
00035000        DO 43 I=J,20
00036000        K=K+1
00037000        IF(IN(I).EQ." ")GOTO 50
00038000   43   OUT(K)=IN(I)
00039000   50   OUT(K)=","
00040000        K=K+1
00041000        OUT(K)=" "
00042000  C
00043000  C WRITE FIRST NAME INTO OUT
00044000  C
00045000        DO 63 I=1,20
00046000        K=K+1
00047000        IF(IN(I).EQ." ")GOTO 70
00048000   60   OUT(K)=IN(I)
00049000  C
00050000  C FILL REMAINDER OF OUT WITH BLANKS
00051000  C
00052000   70   DO 80 I=K,20
00053000   80   OUT(I)=" "
00054000        RETURN
00055000        END


****  NO ERRORS,  NO WARNINGS;   PROGRAM UNIT COMPILED  ****
COMPILATION TIME    1.425 SECONDS    ELAPSED TIME   132.633 SECONDS
TOTAL COMPILATION TIME  0:03:03
TOTAL ELAPSED TIME      0:03:51

 END OF COMPILE

 END OF PREPARE
```

```
:FILE FTN20=NAMES,OLD
:RUN XMPL3

HELLO TRACE  HP32223A.02.1
BATCHFILE= return
MODE=N

*HALT TRACE3
40

*GO

        NAME                    LAST NAME FIRST

     JOHN BIGTOWN               BIGTOWN, JOHN
     LOIS ANYONE                ANYONE, LOIS
     ALI BABA                   BABA, ALI
     JAMES DOE                  DOE, JAMES
     JOHN DOUGHE                DOUGHE, JOHN
     MARY MEEK                  MEEK, MARY
     SPACE MANN                 MANN, SPACE
     KING ARTHUR                ARTHUR, KING
     KARISSA GRANDTR            GRANDTR, KARISSA
     JENNA GRANDTR              GRANDTR, JENNA
     SWASH BUCKLER              BUCKLER, SWASH
     KNEE BUCKLER               BUCKLER, KNEE
40
TRACE3

*SET
STOPNOW=      0/1

*GO

 END OF PROGRAM
```

Computer Museum

Figure 2-15. FORTRAN/3000 Sample Program, TRACE3

```
:RUN XMPL3

HELLO TRACE  HP32222A.02.1
BATCHFILE= return
MODE=N

*HALT TRACE3
40

*PRINT REVERSE
K 40-60
OUT (*<8) 30-50

*GO

        NAME                    LAST NAME FIRST


OUT(1)="B"
OUT(2)="I"
OUT(3)="G"
OUT(4)="T"
OUT(5)="O"
OUT(6)="W"
OUT(7)="N"
K=     9
K=    10
K=    11
K=    12
K=    13
K=    14
     JOHN BIGTOWN               BIGTOWN, JOHN
OUT(1)="A"
OUT(2)="N"
OUT(3)="Y"
OUT(4)="O"
OUT(5)="N"
OUT(6)="E"
K=     8
K=     9
K=    10
K=    11
K=    12
K=    13
     LOIS ANYONE                ANYONE, LOIS
OUT(1)="B"
OUT(2)="A"
OUT(3)="B"
OUT(4)="A"
K=     6
K=     7
K=     8
K=     9
K=    10
     ALI BABA                   BABA, ALI
OUT(1)="D"
OUT(2)="O"
OUT(3)="E"
K=     5
K=     6
K=     7
K=     8
K=     9
K=    10
K=    11
     JAMES DOE                  DOE, JAMES
OUT(1)="D"
OUT(2)="O"
OUT(3)="U"
OUT(4)="G"
OUT(5)="H"
OUT(6)="E"
K=     8
K=     9
K=    10
K=    11
K=    12
K=    13
     JOHN DOUGHE                DOUGHE, JOHN
OUT(1)="M"
OUT(2)="E"
OUT(3)="E"
OUT(4)="K"
K=     6
K=     7
K=     8
K=     9
K=    10
K=    11
     MARY MEEK                  MEEK, MARY
OUT(1)="M"
OUT(2)="A"
OUT(3)="N"
OUT(4)="N"
K=     6
K=     7
K=     8
K=     9
K=    10
K=    11
K=    12
     SPACE MANN                 MANN, SPACE
OUT(1)="A"
OUT(2)="R"
OUT(3)="T"
OUT(4)="H"
OUT(5)="U"
OUT(6)="R"
K=     8
K=     9
K=    10
K=    11
K=    12
K=    13
     KING ARTHUR                ARTHUR, KING
OUT(1)="G"
OUT(2)="R"
OUT(3)="A"
OUT(4)="N"
OUT(5)="D"
OUT(6)="T"
OUT(7)="R"
K=     9
K=    10
K=    11
K=    12
K=    13
K=    14
K=    15
K=    16
K=    17
     KARISSA GRANDTR            GRANDTR, KARISSA
OUT(1)="G"
OUT(2)="R"
OUT(3)="A"
OUT(4)="N"
OUT(5)="D"
OUT(6)="T"
OUT(7)="R"
K=     9
K=    10
K=    11
K=    12
K=    13
K=    14
K=    15
     JENNA GRANDTR              GRANDTR, JENNA
OUT(1)="B"
OUT(2)="U"
OUT(3)="C"
OUT(4)="K"
OUT(5)="L"
OUT(6)="E"
OUT(7)="R"
K=     9
K=    10
K=    11
K=    12
K=    13
K=    14
K=    15
     SWASH BUCKLER              BUCKLER, SWASH
OUT(1)="B"
OUT(2)="U"
OUT(3)="C"
OUT(4)="K"
OUT(5)="L"
OUT(6)="E"
OUT(7)="R"
K=     9
K=    10
K=    11
K=    12
K=    13
K=    14
     KNEE BUCKLER               BUCKLER, KNEE
40
TRACE3

*SET TRACE3
STOPNOW=     0/1

*GO

 END OF PROGRAM
```

Figure 2-16. Label Condition Clause Usage

where

integer primary is either an integer variable or an integer constant. The type is determined by the primary's definition in the source program. If an integer variable, it must have been included in a $TRACE control statement.

TRACE/3000 initially stores the constant or evaluates and stores the value of the variable. A value of less than one is stored as one. If *integer primary* has a value of 4, the use condition clause is false for the first 3 (n - 1) times it is referenced. (Remember that the use condition clause is not referenced unless all previous condition clauses in the sentence are true.) The fourth time the use condition clause is referenced, it is true and the PRINT or HALT sentence is executed. Upon execution of the sentence, TRACE/3000 re-evaluates and stores *integer primary* in the same manner as initially.

The following are examples (in a BATCH file) of use condition clauses:

```
HALT PROG2
VALUE @2
ARRAY1 (* <24) =1000 10-40 @44
NUM >MAXIMUM @ITEM
blank record terminator
```

Figure 2-17 is an example of the use condition clause. The same program is used as in figure 2-16.

The use condition clause @4 is added to the PRINT paragraph sentences K 40-60 and OUT (* <8) 30-50 to produce the PRINT paragraph

```
*PRINT REVERSE
K 40-60 @4
OUT (* <8) 30-50 @4
```

As you recall from figure 2-16, TRACE/3000 had displayed all values of K occurring between labels 40 and 60 and values for array OUT elements less than 8 occurring between labels 30 and 50. Adding the *use condition* clause @4, however, causes TRACE/3000 to display values for variable K and array OUT elements only every *fourth* time.

Comparing figures 2-16 and 2-17, observe that in figure 2-16 for the name "JOHN BIG-TOWN", TRACE/3000 displays OUT(1) through OUT(7) and values for K of 9 through 14. In figure 2-17, however, TRACE/3000 displays OUT(4)="T" (OUT(1), OUT(2), and OUT(3) are not displayed), and K=12. K is not displayed when its value is 9, 10, or 11 (the first 3 times it occurs). This sequence continues through the remainder of the program, with TRACE/3000 displaying every fourth occurrence of array OUT and variable K.

2-14. ROUTINE PARAMETER CLAUSE. The routine parameter clause consists of a pair of empty parentheses "( )" separated from the routine name or $FORM by one or more blanks. This condition clause causes TRACE/3000 to print the current value of the parameters for the specific routine (and the routine's assigned value if it is a function subprogram) when the routine is entered or called during program execution (the program *structure points*).

```
:RUN XMPL3

HELLO TRACE  HP32222A.02.1
BATCHFILE= return
MODE=N

*HALT TRACE3
40

*PRINT REVERSE
K 40-60 @4
OUT (*<8) 30-50 @4

*GO

        NAME                    LAST NAME FIRST


OUT(4)="T"
K=    12
      JOHN BIGTOWN              BIGTOWN, JOHN
OUT(1)="A"
OUT(5)="N"
K=     9
K=    13
      LOIS ANYONE               ANYONE, LOIS
OUT(3)="B"
K=     9
      ALI BABA                  BABA, ALI
OUT(3)="E"
K=     7
K=    11
      JAMES DOE                 DOE, JAMES
OUT(4)="G"
K=    11
      JOHN DOUGHE               DOUGHE, JOHN
OUT(2)="E"
K=     7
K=    11
      MARY MEEK                 MEEK, MARY
OUT(2)="A"
K=     9
      SPACE MANN                MANN, SPACE
OUT(2)="R"
OUT(6)="R"
K=     8
K=    12
      KING ARTHUR               ARTHUR, KING
OUT(4)="N"
K=    11
K=    15
      KARISSA GRANDTR           GRANDTR, KARISSA
OUT(1)="G"
OUT(5)="D"
K=    10
K=    14
      JENNA GRANDTR             GRANDTR, JENNA
OUT(2)="U"
OUT(6)="E"
K=    11
K=    15
      SWASH BUCKLER             BUCKLER, SWASH
OUT(3)="C"
OUT(7)="R"
K=    12
      KNEE BUCKLER              BUCKLER, KNEE
40
TRACE3

*SET TRACE3
STOPNOW=      0/1

*GO

 END OF PROGRAM
```

Figure 2-17. Use Condition Clause Usage

The form of the routine parameter clause is

routine name ( )

or

$FORM ( )

For example,

REVERSE ( )

Figures 2-18 and 2-19 illustrate the use of the routine parameter clause.

Figure 2-18 uses program TRACE3. The PRINT paragraph

```
*PRINT TRACE3
REVERSE ( )
```

specifies the routine REVERSE followed by a pair of empty parentheses. When the program executes, TRACE/3000 displays the values assigned to the parameters passed to the subroutine REVERSE.

The first call to REVERSE results in the display

```
CALL REVERSE("J"," ")
```

"J" is the value of element 1 of character array NAMEIN (the first letter of the name "JOHN BIGTOWN"). NAMEIN is one of the parameters passed to REVERSE. The second part of the display (" ") shows that character array NAMEOUT is null at this point (it will be given a value by the subroutine). The second call to REVERSE shows that NAMEIN(1)="L" (the first letter of "LOIS ANYONE") and NAMEOUT(1)="B", which is the first letter assigned to NAMEOUT by the subroutine REVERSE on the previous call. (NAMEOUT retains its previous value until subroutine REVERSE executes again, filling NAMEOUT with new values.)

The program executes until statement label 40 is reached, then displays

```
40
TRACE3
```

The SET paragraph is used to set the value of STOPNOW to 1 and the second GO command terminates the program.

Figure 2-19 uses a short FORTRAN program which increments a value from 1.0 to 10.0 and computes the square root and the reciprocal of the value.

The PRINT paragraph

```
*PRINT TRACE1
$FORM ( )
```

illustrates the use of the $FORM ( ) type of routine parameter condition clause. Using $FORM causes TRACE/3000 to display all program structure points during program execution. The empty parentheses causes TRACE/3000 to display values of parameters passed to routines.

```
:RUN XMPL3

HELLO TRACE  HP32222A.02.1
BATCHFILE= return
MODE=N

*PRINT TRACE3
REVERSE ()

*HALT
40

*GO
        NAME                    LAST NAME FIRST


CALL REVERSE("J","")
RETURN REVERSE
     JOHN BIGTOWN               BIGTOWN, JOHN
CALL REVERSE("L","B")
RETURN REVERSE
     LOIS ANYONE                ANYONE, LOIS
CALL REVERSE("A","A")
RETURN REVERSE
     ALI BABA                   BABA, ALI
CALL REVERSE("J","B")
RETURN REVERSE
     JAMES DOE                  DOE, JAMES
CALL REVERSE("J","D")
RETURN REVERSE
     JOHN DOUGHE                DOUGHE, JOHN
CALL REVERSE("M","D")
RETURN REVERSE
     MARY MEEK                  MEEK, MARY
CALL REVERSE("S","M")
RETURN REVERSE
     SPACE MANN                 MANN, SPACE
CALL REVERSE("K","M")
RETURN REVERSE
     KING ARTHUR                ARTHUR, KING
CALL REVERSE("K","A")
RETURN REVERSE
     KARISSA GRANDTR            GRANDTR, KARISSA
CALL REVERSE("J","G")
RETURN REVERSE
     JENNA GRANDTR              GRANDTR, JENNA
CALL REVERSE("S","G")
RETURN REVERSE
     SWASH BUCKLER              BUCKLER, SWASH
CALL REVERSE("K","B")
RETURN REVERSE
     KNEE BUCKLER               BUCKLER, KNEE
40
TRACE3


*SET
STOPNOW=      0/1

*GO

 END OF PROGRAM
```

Figure 2-18. Routine Parameter Condition Clause, Example 1

```
:FORTGO TEST

PAGE 0001    HP32102A.01.6


00001000  $CONTROL USLINIT
00002000  $TRACE TEST,A,1,SQRT,ROOT,RCPL
00003000        PROGRAM TEST
00004000   100  FORMAT('0',T2,"NUMBER",T12,"SQUARE ROOT",T27
00005000       #,"RECIPROCAL"//)
00006000   200  FORMAT(T2,F4.1,T14,F7.4,T28,F7.4)
00007000   10   WRITE(6,100)
00008000        A=1.0
00009000   20   DO 30 I=1,10
00010000        ROOT=SQRT(A)
00011000        RCPL=1/A
00012000        WRITE(6,200)A,ROOT,RCPL
00013000   30   A=A+1.0
00014000        STOP
00015000        END



****  NO ERRORS,  NO WARNINGS;   PROGRAM UNIT COMPILED  ****
COMPILATION TIME    1.359 SECONDS    ELAPSED TIME    62.638 SECONDS
TOTAL COMPILATION TIME  0:00:02
TOTAL ELAPSED TIME      0:01:17

 END OF COMPILE

 END OF PREPARE

HELLO TRACE  HP32222A.02.2
BATCHFILE= return
MODE=N

*PRINT TEST
$FORM ()

*GO


NUMBER    SQUARE ROOT    RECIPROCAL


CALL SQRT( 1.000000    )
RETURN SQRT= 1.000000
 1.0          1.0000          1.0000
CALL SQRT( 2.000000    )
RETURN SQRT= 1.414214
 2.0          1.4142           .5000
CALL SQRT( 3.000000    )
RETURN SQRT= 1.732051
 3.0          1.7321           .3333
CALL SQRT( 4.000000    )
RETURN SQRT= 2.000000
 4.0          2.0000           .2500
CALL SQRT( 5.000000    )
RETURN SQRT= 2.236068
 5.0          2.2361           .2000
CALL SQRT( 6.000000    )
RETURN SQRT= 2.449490
 6.0          2.4495           .1667
CALL SQRT( 7.000000    )
RETURN SQRT= 2.645751
 7.0          2.6458           .1429
CALL SQRT( 8.000000    )
RETURN SQRT= 2.828427
 8.0          2.8284           .1250
CALL SQRT( 9.000000    )
RETURN SQRT= 3.000000
 9.0          3.0000           .1111
CALL SQRT( 10.00000    )
RETURN SQRT= 3.162278
10.0          3.1623           .1000
 END OF PROGRAM
```

Figure 2-19. Routine Parameter Condition Clause, Example 2

When the program executes, TRACE/3000 displays

```
CALL SQRT ( 1.000000 )
```

where 1.000000 is the value passed to SQRT. TRACE/3000 then displays

```
RETURN SQRT= 1.000000
```

SQRT is a basic external *function*, and a value, associated with its name, is returned to the calling program unit. Using the $FORM ( ) type of routine parameter condition clause causes TRACE/3000 to display the value of SQRT on the return.

Note: The form SQRT ( ) also would cause TRACE/3000 to display the value assigned to SQRT on the return from this function.

2-15. PRINT AND HALT SENTENCE EXECUTION CONDITIONS

A PRINT or HALT sentence executes *only if all of the following conditions* are satisfied by the source program:

1. The program unit assigned to the PRINT or HALT paragraph is referenced in a $TRACE control statement in the source program.

2. The identifier in a PRINT or HALT paragraph sentence has been referenced in a $TRACE control statement in the source program. Note, however, that if $LABEL is used in a sentence, label identifiers need not have been mentioned in a $TRACE control statement and if $FORM is used in a sentence, routine identifiers need not have been referenced in a $TRACE control statement unless $DELETE was used. ($LABEL causes TRACE/3000 to monitor *all* labels in a program unit and $FORM causes TRACE/3000 to monitor *all* structure points in a program unit.)

3. All condition clauses in the sentence are true.

For example, the PRINT sentence below executes only if REVERSE has been referenced in a $TRACE control statement, and if the $TRACE control statement specified an array called OUT (the *identifier*) whose subscript must be less than 8 (the *subscript value condition clause*) and whose value must be equal to "T" (the *identifier value condition clause*). The call to TRACE/3000 must also occur between statement labels 30 and 50 (the *label condition clause*). All the former conditions must have been true three times previously, thus making the *use condition clause* true and the entire PRINT statement true.

```
PRINT REVERSE
OUT (* <8) ="T" 30-50 @4
```

A PRINT or HALT sentence can exist without any condition clauses at all. The PRINT statement below executes if the program unit REVERSE and the identifier K have been referenced in a $TRACE control statement.

```
PRINT REVERSE
K
```

2-16. PRINT AND HALT SENTENCE RESULTS

When a PRINT or HALT sentence is evaluated as true, a message is displayed on the standard

output device (a terminal in an interactive session or a line printer in a batch job) unless the TRACE/3000 list file, TRCLIST, has been equated to another device (see Section III). The form of the message depends on the type of identifier referenced in the sentence, as follows:

Sentence Type	Message
variable	*variable = value*
array	*array (subscript value) = value*
label	*name of label*
routine	*CALL routine name*
	RETURN routine name
$FORM	*CALL routine name*
	ENTER routine name
	EXIT routine name
	RETURN routine name

- *Variable Identifier.* If the identifier is a variable, the variable name is displayed along with the value of the variable. For example, the PRINT paragraph below produces the results as shown.

```
*PRINT TRACE2
I
J

*GO
I=        5
J=        5
```

- *Array Identifier.* An array identifier in a PRINT sentence produces all values of the array unless a subscript value condition clause is used in the sentence. For example, the first PRINT paragraph below produces values for all elements of the array IVAR whereas the second PRINT paragraph produces values for subscripts 16 and 9 only because of the use of the subscript value condition clauses.

```
*PRINT SB
IVAR

*GO
IVAR(1)=       1
IVAR(6)=       2
IVAR(11)=      3
IVAR(16)=      4
IVAR(21)=      5
IVAR(2)=       2
IVAR(7)=       4
IVAR(12)=      6
IVAR(17)=      8
```

```
IVAR(22)=      10
IVAR(3)=       3
IVAR(8)=       6
IVAR(13)=       9
IVAR(18)=      12
IVAR(23)=      15
IVAR(4)=       4
IVAR(9)=       8
IVAR(14)=      12
IVAR(19)=      16
IVAR(24)=      20
IVAR(5)=       5
IVAR(10)=      10
IVAR(15)=      15
IVAR(20)=      20
IVAR(25)=      25


*PRINT SB
IVAR (*=16)
IVAR (*=9)

*GO

IVAR(16)=       4
IVAR(9)=       8
```

- *Label Identifier.* A label identifier causes TRACE/3000 to display the name of the label (an alphabetic name for SPL/3000 programs or a numeric name for FORTRAN/3000 programs). For example,

```
*PRINT TRACE2
10
20
30

*GO

10
20
30
```

- *Routine Identifier.* If a routine identifier is used in a sentence, TRACE/3000 displays CALL *routine name* when the call to the routine is executed and RETURN *routine name* when control is passed back to the calling program unit. For example,

```
*PRINT TRACE2
SB

*GO

CALL SB
RETURN SB
```

If $FORM is used in a sentence, TRACE/3000 displays the same information as above, even though the routine identifier was not used in the sentence. For example,

```
*PRINT TRACE2
$FORM

*GO

CALL SB
RETURN SB
```

If $FORM appears in a sentence in a PRINT or HALT paragraph that referenced the routine itself, TRACE/3000 displays ENTER *routine name* when the routine is entered and EXIT *routine name* when control exits the routine. For example,

```
*PRINT SB
$FORM

*GO

ENTER SB
EXIT SB
```

If $FORM is used in a paragraph referencing the calling program unit and in a paragraph referencing the routine, TRACE/3000 displays the following information:

```
*PRINT TRACE2
$FORM

*PRINT SB
$FORM

*GO

CALL SB
ENTER SB
EXIT SB
RETURN SB
```

If the sentence contains a routine parameter clause, for example SB ( ), the display format will include a list of the parameter values at the time of the call to the routine. For example,

```
*PRINT TRACE2
SB ()

*GO
CALL SB(    341,"",      5,      5,     10)
RETURN SB
```

If the routine is a function (that is, a value associated with the routine's name is returned after execution of the routine) and the routine parameter clause is used, the RETURN form of the display includes the value of the function. For example,

```
*PRINT TRACE1
SQRT ()

*GO

CALL SQRT( 1.000000     )
RETURN SQRT= 1.000000
```

2-17. DROP COMMAND

The DROP command is used as the first statement in a DROP paragraph and deletes PRINT and HALT sentences from the PRINT/HALT table. Unlike PRINT and HALT paragraphs, DROP paragraphs may be entered into the INTERACTIVE file only. The INTERACTIVE file is opened and accessed only if the source program is running in interactive mode.

The form of the DROP command is

DROP [*program unit name*]

or

DROPALL

The DROP paragraph acts on the PRINT/HALT table in one of three ways:

1. Deletes all PRINT and HALT sentences in the PRINT/HALT table.
2. Deletes all PRINT and HALT sentences pertaining to a specific program unit.
3. Deletes PRINT and HALT statements pertaining to specific identifiers in a program unit.

To delete the entire PRINT/HALT table, enter the DROPALL form of the DROP command.

Note: The asterisk is output by the computer in the following examples.

```
*DROPALL
```

To delete all sentences pertaining to a specific program unit, use the DROP *program unit name* form of the DROP command, followed by $ALL. For example,

```
*DROP TRACE2
$ALL
```

To delete sentences pertaining to specific identifiers within a program unit, use the DROP

program unit name form of the DROP command, followed by the specific identifiers that you wish to drop. For example,

```
*DROP TRACE2
I
J
```

Note: The *program unit name* parameter is optional if the DROP paragraph pertains to the same program unit as the preceding paragraph.

2-18. CHECK COMMAND

The CHECK command is used as the first command in a CHECK paragraph. The CHECK paragraph confirms the correct application of program unit name and identifier name abbreviations as they appear in the paragraphs (see Appendix B, "Abbreviations in TRACE/3000 Paragraphs"). CHECK paragraphs can be entered only through the INTERACTIVE file during an interactive session.

The form of the CHECK command is

CHECK [*program unit name*]

or

CHECK [*program unit name abbreviation*]

For example,

```
*CHECK TRACE2
```

or

```
*CHECK T
```

After you enter the CHECK command (optionally followed by *program unit name* or *program unit name abbreviation*), TRACE2 responds with the complete spelling of *program unit name*.

For example,

```
*CHECK TRACE3
TRACE3

*CHECK T
TRACE3
```

Note: If there is more than one item starting with a given letter, only the first item in alphabetic sequence is displayed. See Appendix B.

If *program unit name* and *program unit name abbreviation* are omitted from the CHECK command, TRACE/3000 assigns *program unit name* from the paragraph immediately preceding. For example,

```
*CHECK
TRACE3
```

In the above example, TRACE3 was the *program unit name* in effect for the preceding paragraph and is now assigned to the current paragraph by TRACE/3000.

Once you enter the CHECK command and TRACE/3000 responds by displaying the full spelling of the *program unit name* assigned to the CHECK paragraph, you can enter identifier name abbreviations for identifiers pertaining to that program unit. To accomplish this, enter the identifier abbreviation followed by an equal sign (=). TRACE/3000 responds by typing the full spelling of the abbreviated identifier. For example,

```
*CHECK
TRACE3
N=NAMEIN
S=STOPNOW
R=REVERSE
```

If the abbreviation entered into a CHECK paragraph is not the abbreviation for any identifier within the program unit assigned to the paragraph, TRACE/3000 responds with a BAD SYNTAX message. For example,

```
*CHECK
TRACE3
R=REVERSE
Q=
  BAD SYNTAX      1
```

2-19. SET COMMAND

The SET command is used as the first command in a SET paragraph. The SET paragraph performs two main functions:

1. Examines and changes the values of *terms* within program units.

2. Reports the relative addresses of *terms* according to their positions within the user stack area. (See the *HP 3000 Computer System Reference Manual* for a discussion of the stack.)

TRACE/3000 recognizes two classes of *terms*: *variables* and *elements*. A *variable* is one of the following:

1. A simple variable name without subscript. For example,

   I
   K
   STOPNOW

2. An array name without subscript (FORTRAN/3000 or SPL/3000) or a pointer name without subscript (SPL/3000 only). For example,

   IVAR
   ARRAY1
   MAXPOINT

3. @array name or @pointer name without subscript (SPL/3000 only). For example,

   @ARRAY1
   @SORT
   @MAXPOINT

An *element* is either:

1. An array name with subscript (FORTRAN/3000 and SPL/3000) or a pointer name with subscript (SPL/3000 only). For example,

   IVAR(1)
   ARRAY1(13)
   MAXPOINT(3)

2. A stack element specified by a stack register name and an increment. For example,

   DB+100
   Q−13
   S−2

The stack element takes one of three forms:

DB ± *unsigned octal number*
Q ± *unsigned octal number*
S − *unsigned octal number*

S, Q, and DB stand for *stack registers* which indicate the start of areas within the stack. All references to stack elements must be within the bounds of your current stack. It is out of the scope of this manual to discuss stack limits; therefore, refer to the *HP 3000 Computer System Reference Manual* and the *MPE/3000 Operating System Reference Manual* for a complete discussion of the stack.

A SET paragraph can be entered into the INTERACTIVE file at any time the file is accessed. If a SET paragraph is entered when the INTERACTIVE file is initially accessed and before program execution, only global or common variables, or DB relative stack elements may be used. If a SET paragraph is entered after the program has started execution, then all DB relative, Q relative, and S relative terms may be used, providing they are contained in the currently executing program unit. (See the *MPE/3000 Operating System Reference Manual* for a discussion of *global* variables, *common* variables, and *DB, Q,* and *S relative* terms.)

Term usage in SET paragraphs further depends on whether TRACE/3000 is operated in NORMAL or RESTRICTED mode. In RESTRICTED mode, only simple variables can be modified in a SET paragraph sentence. @array name, @pointer name, or *elements* of any kind are not modified by TRACE/3000 in RESTRICTED mode (although their values can be listed). In NORMAL mode, TRACE/3000 modifies terms of all kinds subject to the terms definition as *global, common,* or *program local*. (See the *MPE/3000 Operating System Reference Manual*.)

SET commands and paragraphs may be entered into the INTERACTIVE file only. To enter a SET paragraph, enter SET followed by the (optional) *program unit name*. If *program unit name* is omitted, TRACE/3000 assigns the program unit from the preceding paragraph. If the SET paragraph is the first paragraph in the INTERACTIVE file, *program unit name* must be specified.

To enter a SET paragraph, enter, for example,

```
*SET TRACE2
```

You now can enter as many SET sentences as desired. To display the value of a term, type the term name, followed immediately by an equal sign. TRACE/3000 responds by displaying the current value of the term. For example,

```
*SET TRACE2
I=      5
J=      5
```

Once TRACE/3000 displays the current value of the term, you can change the value of the term by typing a slash and the new value of the term. For example,

```
*SET TRACE2
I=      5/6
J=      5/6
```

You can change the value of a term repeatedly within the same sentence. Entries are limited to the current line, however (you are not allowed to continue entries to the next line without respecifying the name of the term). An example of repeated changes to a term's value is

```
*SET TRACE2
I=      5/6/10/23/45
```

2-20. BLOCK LISTING OF ELEMENTS

To list a block of related elements (such as several elements of the same array), enter the array name, the array element subscript (in parentheses) indicating the element at which the listing will start, a comma, and an unsigned positive decimal integer indicating how many elements TRACE/3000 is to list. Simple variables (terms without subscripts or indexes) cannot be listed in block form, since their definition implies only one element in their group. TRACE/3000 displays the elements line by line. The number of elements per line depends on the element type. For string data, TRACE/3000 leaves three spaces between elements. If the complete listing cannot be displayed on one line, TRACE/3000 continues it on the next line.

An example of array block listing is

```
*SET TRACE2
I ARR(1),10=
       1       2       3       4       5       2       4       6       8
      10
```

The listing started with array element 1 and continued for the next 9 elements of the array.

Block data listing also can be used for stack elements by entering the stack register, a plus or minus sign, the element at which the listing is to begin, a comma, and the number of elements to be listed. For example,

```
*SET TRACE2
DB+0,6=
  %000041   %000000   %000000   %000100   %040006   %001123
```

2-21. STRING DATA IN SET PARAGRAPHS

TRACE/3000 displays string data enclosed in quotes ("). A quote mark within a string is displayed twice to distinguish it from the quote marks at the beginning and end of the string. For example, the string value ABC"D is displayed by TRACE/3000 as

"ABC""D"

String values entered by you must follow the same rule. For example,

```
*SET
CHAR= "ABC""D"/"X""CDF"
```

would replace the value ABC"D with X"CDF.

When you change the value of a string variable, take care to replace the original value with a string value of equal or shorter length. An attempt to replace a string value with a string value of longer length results in a BAD SYNTAX message. For example,

```
*SET TRACE2
A="THE START "/"THERE IS A START"
  BAD SYNTAX     27 <-------- Column number where input is incorrect.
```

If a string value is replaced with a value of shorter length, then only the number of characters in the replacement value are changed in the original value. For example,

```
*SET TRACE2
A="THE START"/"A START"
A="A STARTRT"
```

2-22. USING THE SET COMMAND FOR TERM ADDRESS

You can enter SET paragraphs into the INTERACTIVE file to locate the stack addresses of terms. To accomplish this, start the SET paragraph by entering the SET command and the program unit name (if required), then enter a hatch mark (#), the term, and an equal sign. For example,

```
*SET TRACE2
#I=
```

TRACE/3000 responds with the stack address of the term. Unlike SET paragraphs which allow you to change the value of terms, this form of SET does not allow you to change the term's address in memory.

The address of the term is always given by TRACE/3000 relative to either the DB, Q, or S register. TRACE/3000 follows the register name by the proper displacement (in computer words). If the term is addressed in the stack through an indirect reference, TRACE/3000 follows the address with ,I. If the term is addressed through the index register, TRACE/3000 follows the address with ,X. The register used in the address depends on the registers used by the MPE/3000 operating system. If the term is a globally defined variable in the source program, TRACE/3000 uses the DB register. If the term is a local variable in the program unit, TRACE/3000 displays the address in terms of the Q or S register. See the *MPE/3000 Operating System Reference Manual* for a complete discussion of addressing.

Some examples of SET paragraphs used for addressing are

```
*SET TRACE2
# I=Q+2
# J=Q+4
# STOPNOW=Q+3
# IARR(3)=Q+1,I,X
```

2-23. GO COMMAND

The GO command de-activates the INTERACTIVE file and starts execution of the program.

The form of the GO command is

GO [*label*]

For example,

GO

or

GO 10

The GO command can be used only in the INTERACTIVE file and starts in column 2 of the line (following the asterisk displayed by TRACE/3000 as a prompt character). The *label* parameter is optional, and if used, causes program execution to start at the point in the program specified by *label*. The GO *label* form of the command is not allowed if TRACE/3000 is operating in the RESTRICTED mode. If no label is included in the first GO command, execution starts at the main entry point of the main program unit.

CAUTION

Do not go to a label while in a DO loop (FORTRAN) or a FOR statement (SPL). This can cause unpredictable results due to the stack related dependency of this construct. See the MTBA instruction in the *HP 3000 Reference Manual.*

The GO command resumes execution only in the currently executing program unit (or initially, the main program unit). TRACE/3000 displays a BAD SYNTAX message and ignores the GO command if a label is specified that is not contained in the currently executing program unit.

2-24. BAD SYNTAX ERROR MESSAGES

Whenever TRACE/3000 discovers a sentence with improper form, it displays a BAD SYNTAX *n* message, where *n* is the first character position in the line where the error occurred.

If an error occurs in the BATCH file, TRACE/3000 ignores the rest of the paragraph following the improper sentence and either aborts the program (if the ABORT parameter was used in the $TRACESTART record) or reads the next paragraph if the ABORT parameter was not used in the $TRACESTART record.

If an error occurs in the INTERACTIVE file, TRACE/3000 ignores the sentence and prints a BAD SYNTAX message. A corrected sentence can then be entered.

USING TRACE/3000

SECTION III

TRACE/3000 can be used in either batch mode or interactive mode. In batch mode, you must use a BATCH file and you may not use the INTERACTIVE file. In interactive mode, you may use both files, or the INTERACTIVE file only.

3-1. PREPARING A BATCH FILE

The BATCH file is opened with a $TRACESTART statement (which may contain the optional parameter ABORT, see Section II) and is closed with a $TRACEEND statement. PRINT and HALT paragraphs are inserted in the BATCH file between the $TRACESTART and $TRACEEND statements. You can insert as many PRINT and HALT paragraph sentences as desired, subject to the PRINT/HALT table size (see paragraph 3-5). No PRINT or HALT paragraphs need be entered into the BATCH file, but the $TRACESTART and $TRACEEND statements must be present. If a $TRACESTART or $TRACEEND statement is missing or misspelled, TRACE/3000 displays BAD TRACE FILE and terminates the program. In batch mode, the program terminates and is flushed from the system; in interactive mode, control passes to TRACE/3000, which activates the INTERACTIVE file.

A blank record must be inserted between PRINT and HALT paragraphs and between the last PRINT or HALT paragraph and the $TRACEEND statement; however, no blank record may be inserted between the $TRACESTART and $TRACEEND statements if no PRINT or HALT paragraphs are included in the BATCH file.

A BATCH file can be prepared and saved through either of two methods:

1. For a batch job, the BATCH file must be prepared on a batch input medium such as punched cards. In addition, the BATCH file must be input to the computer when the program is run before the program data from $STDIN. Figure 3-1 shows a BATCH file punched on cards. See paragraph 3-4 for a discussion of a source program which uses this BATCH file during a batch job.

2. For an interactive session, the BATCH file can be prepared, given a file name, and can be stored on disc and then referenced when the source program is run. Figure 3-2 shows a BATCH file prepared using EDIT/3000, kept under the file name BATCH3, and stored on disc. See paragraph 3-3 for a discussion of a source program which uses this BATCH file during an interactive session.

3-2. PREPARING A SOURCE PROGRAM TO BE MONITORED BY TRACE/3000

Source programs which are to be monitored by TRACE/3000 must contain $TRACE control statements. The $TRACE control statement contains the name of the program unit and a list of *identifiers* which specify those items (simple variables, arrays, and routines) in the program unit which are to be monitored by TRACE/3000.

Figure 3-3 shows a FORTRAN/3000 source program containing two $TRACE control statements, as follows:

```
$TRACE TRACE3;NAMEIN,NAMEOUT,STOPNOW
$TRACE REVERSE;IN,OUT,I,J,K
```

The first $TRACE control statement specifies TRACE3 as the program unit; and the arrays NAMEIN and NAMEOUT and the simple variable STOPNOW as the items to be monitored by TRACE/3000. The second $TRACE control statement specifies REVERSE as the program unit; and the arrays IN and OUT and the simple variables I, J, and K as the items to be monitored.

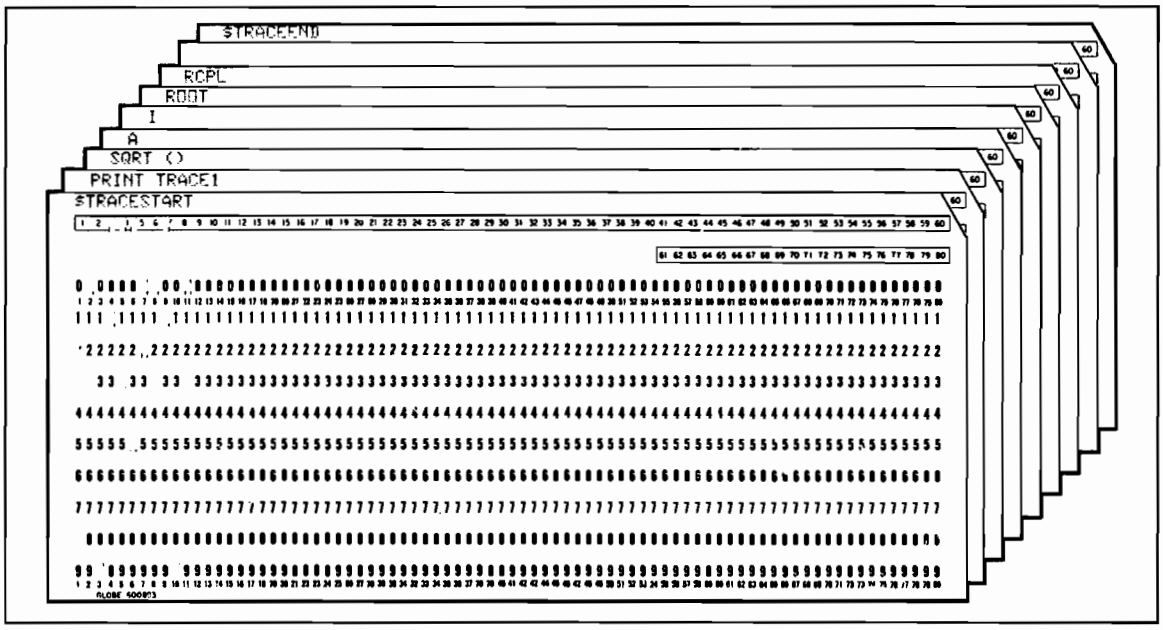


Figure 3-1. BATCH File Punched on Cards

```
:EDITOR

HP32201A.4.01 EDIT/3000   THU, JUL 10, 1975,  3:57 PM
/S SHORT
/ADD
     1      $TRACESTART
     2      PRINT TRACE3
     3      $FORM
     4      return
     5      HALT
     6      40
     7      return
     8      PRINT REVERSE
     9      $FORM
    10      return
    11      $TRACEEND
    12      ... Y
/KEEP BATCH3
/E
CLEAR? Y

 END OF SUBSYSTEM
```

Figure 3-2. BATCH File Prepared Using EDIT/3000

```
 1      $CONTROL USLINIT
 2      $TRACE TRACE3:NAMEIN,NAMEOUT,STOPNOW
 3      $TRACE REVERSE:IN,OUT,I,J,K
 4            PROGRAM TRACE3
 5       100  FORMAT(T10,"NAME",T30,"LAST NAME FIRST"//)
 6       200  FORMAT(20A1)
 7       300  FORMAT(T7,20A1,T32,20A1)
 8            CHARACTER NAMEIN(20),NAMEOUT(20)
 9            INTEGER STOPNOW
10            STOPNOW=0
11       10   WRITE(6,100)
12       20   READ(20,200,END=40)NAMEIN
13       30   CALL REVERSE(NAMEIN,NAMEOUT)
14            WRITE(6,300)NAMEIN,NAMEOUT
15            GOTO 20
16       40   CONTINUE
17            IF(STOPNOW.NE.0)STOP
18            REWIND 20
19            GOTO 10
20       50   STOP
21            END
22            SUBROUTINE REVERSE(IN,OUT)
23            CHARACTER IN(20),OUT(20)
24      C
25      C FIND END OF FIRST NAME
26      C
27       10   DO 20 I=1,20
28            IF(IN(I).EQ." ")GOTO 30
29       20   CONTINUE
30       30   J=I+1
31      C
32      C WRITE LAST NAME INTO OUT
33      C
34            K=0
35            DO 40 I=J,20
36            K=K+1
37            IF(IN(I).EQ." ")GOTO 50
38       40   OUT(K)=IN(I)
39       50   OUT(K)=","
40            K=K+1
41            OUT(K)=" "
42      C
43      C WRITE FIRST NAME INTO OUT
44      C
45            DO 60 I=1,20
46            K=K+1
47            IF(IN(I).EQ." ")GOTO 70
48       60   OUT(K)=IN(I)
49      C
50      C FILL REMAINDER OF OUT WITH BLANKS
51      C
52       70   DO 80 I=K,20
53       80   OUT(I)=" "
54            RETURN
55            END
```

Figure 3-3. FORTRAN/3000 Source Program TRACE3

Be sure to identify, in a $TRACE control statement, all items which are to be monitored by TRACE/3000. Failure to do so will result in a BAD SYNTAX error message when any such items are entered into the BATCH or INTERACTIVE files during an interactive session; or will cause the program to abort if such items are included in the BATCH file during a batch job.

$TRACE control statements are inserted into a source program in one of two ways: Either place the $TRACE control statements for each program unit immediately in front of the program unit to which the $TRACE statements apply, or group all $TRACE statements and place them in front of the first program unit.

3-3. USING TRACE/3000 IN AN INTERACTIVE SESSION

Once a source program is coded, it can be compiled and prepared into a program file by using the appropriate compiler command (:SPLPREP for SPL/3000 programs and :FORTPREP for FORTRAN/3000 programs) or it can be compiled, prepared, and executed using the :SPLGO or :FORTGO commands. See the *MPE/3000 Operating System Reference Manual* for descriptions of the foregoing commands.

Figure 3-4 shows the FORTRAN/3000 source program TRACE3 compiled and prepared, then saved under program file XMPL3.

The source program is compiled and prepared using the

```
:FORTPREP TRACE3
```

command, then saved under program file name XMPL3 with the

```
:SAVE $OLDPASS, XMPL3
```

command.

Figure 3-5 illustrates running the compiled and prepared program using the :RUN command and BATCH file BATCH3.

The :FILE commamd

```
:FILE FTN20=NAMES,OLD
```

is used to equate the old file NAMES to FORTRAN logical unit number 20 (FTN20) so that this file can be accessed by the program.

The :RUN XMPL3 command starts program execution, and since the source program contained $TRACE control statements, TRACE/3000 is accessed. TRACE/3000 displays

```
HELLO TRACE
```

then displays

```
BATCHFILE=
```

BATCH file BATCH3 (see paragraph 3-1) is entered, then TRACE/3000 displays

```
MODE=
```

activating the INTERACTIVE file and asking for the operational mode. N is entered for NORMAL and TRACE/3000 prompts for input to the INTERACTIVE file by displaying an asterisk. The GO command de-activates the INTERACTIVE file and starts program execution.

```
:BUILD XMPL3;CODE=PROG
:FORTPREP TRACE3,XMPL3

PAGE 0001    HP32102A.01.4


00001000  $CONTROL USLINIT
00002000  $TRACE TRACE3;NAMEIN,NAMEOUT,STOPNOW
00003000  $TRACE REVERSE;IN,OUT,I,J,K
00004000        PROGRAM TRACE3
00005000   100  FORMAT(T10,"NAME",T30,"LAST NAME FIRST"//)
00006000   200  FORMAT(20A1)
00007000   300  FORMAT(T7,20A1,T32,20A1)
00008000        CHARACTER NAMEIN(20),NAMEOUT(20)
00009000        INTEGER STOPNOW
00010000        STOPNOW=0
00011000    10  WRITE(6,100)
00012000    20  READ(20,200,END=40)NAMEIN
00013000    30  CALL REVERSE(NAMEIN,NAMEOUT)
00014000        WRITE(6,300)NAMEIN,NAMEOUT
00015000        GOTO 20
00016000    40  CONTINUE
00017000        IF(STOPNOW.NE.0)STOP
00018000        REWIND 20
00019000        GOTO 10
00020000    50  STOP
00021000        END


**** NO ERRORS,  NO WARNINGS;   PROGRAM UNIT COMPILED  ****
COMPILATION TIME    1.531 SECONDS    ELAPSED TIME    87.699 SECONDS
00022000        SUBROUTINE REVERSE(IN,OUT)
00023000        CHARACTER IN(20),OUT(20)
00024000  C
00025000  C FIND END OF FIRST NAME
00026000  C
00027000    10  DO 20 I=1,20
00028000        IF(IN(I).EQ." ")GOTO 30
00029000    20  CONTINUE
00030000    30  J=I+1
00031000  C
00032000  C WRITE LAST NAME INTO OUT
00033000  C
00034000        K=0
00035000        DO 40 I=J,20
00036000        K=K+1
00037000        IF(IN(I).EQ." ")GOTO 50
00038000    40  OUT(K)=IN(I)
00039000    50  OUT(K)=","
00040000        K=K+1
00041000        OUT(K)=" "
00042000  C
00043000  C WRITE FIRST NAME INTO OUT
00044000  C
00045000        DO 60 I=1,20
00046000        K=K+1
00047000        IF(IN(I).EQ." ")GOTO 70
00048000    60  OUT(K)=IN(I)
00049000  C
00050000  C FILL REMAINDER OF OUT WITH BLANKS
00051000  C
00052000    70  DO 80 I=K,20
00053000    80  OUT(I)=" "
00054000        RETURN
00055000        END


**** NO ERRORS,  NO WARNINGS;   PROGRAM UNIT COMPILED  ****
COMPILATION TIME    1.779 SECONDS    ELAPSED TIME   124.484 SECONDS
TOTAL COMPILATION TIME  0:00:04
TOTAL ELAPSED TIME      0:03:46

 END OF COMPILE

 END OF PREPARE
```

Figure 3-4. Compiling and Preparing Source Program TRACE3

```
:FILE FTN20=NAMES,OLD
:RUN XMPL3

HELLO TRACE  HP32222A.02.1
BATCHFILE=BATCH3


MODE=N

*GO

        NAME                    LAST NAME FIRST


CALL REVERSE
ENTER REVERSE
EXIT REVERSE
RETURN REVERSE
     JOHN BIGTOWN               BIGTOWN, JOHN
CALL REVERSE
ENTER REVERSE
EXIT REVERSE
RETURN REVERSE
     LOIS ANYONE                ANYONE, LOIS
CALL REVERSE
ENTER REVERSE
EXIT REVERSE
RETURN REVERSE
     ALI BABA                   BABA, ALI
CALL REVERSE
ENTER REVERSE
EXIT REVERSE
RETURN REVERSE
     JAMES DOE                  DOE, JAMES
CALL REVERSE
ENTER REVERSE
EXIT REVERSE
RETURN REVERSE
     JOHN DOUGHE                DOUGHE, JOHN
CALL REVERSE
ENTER REVERSE
EXIT REVERSE
RETURN REVERSE
     MARY MEEK                  MEEK, MARY
CALL REVERSE
ENTER REVERSE
EXIT REVERSE
RETURN REVERSE
     SPACE MANN                 MANN, SPACE
CALL REVERSE
ENTER REVERSE
EXIT REVERSE
RETURN REVERSE
     KING ARTHUR                ARTHUR, KING
CALL REVERSE
ENTER REVERSE
EXIT REVERSE
RETURN REVERSE
     KARISSA GRANDTR            GRANDTR, KARISSA
CALL REVERSE
ENTER REVERSE
EXIT REVERSE
RETURN REVERSE
     JENNA GRANDTR              GRANDTR, JENNA
CALL REVERSE
ENTER REVERSE
EXIT REVERSE
RETURN REVERSE
     SWASH BUCKLER              BUCKLER, SWASH
CALL REVERSE
ENTER REVERSE
EXIT REVERSE
RETURN REVERSE
     KNEE BUCKLER               BUCKLER, KNEE
40
TRACE3
```

Figure 3-5. Running Program XMPL3 Using BATCH File BATCH3

The PRINT paragraphs in the BATCH file (see figure 3-2) cause TRACE/3000 to monitor the program structure points in program units TRACE3 and REVERSE and

```
CALL REVERSE
ENTER REVERSE
EXIT REVERSE
RETURN REVERSE
```

is displayed each time subroutine REVERSE is called, executed, and exited. The program halts at statement 40 in the main program (TRACE3).

In figure 3-6, the INTERACTIVE file is used to add two new PRINT paragraphs as follows:

```
*PRINT TRACE3
$LABEL

*PRINT REVERSE
$LABEL
```

Computer Museum

When the GO command is entered, TRACE/3000 monitors and displays all statement labels encountered when the program units TRACE3 and REVERSE are executed. The PRINT and HALT paragraphs from the BATCH file are still effective, thus TRACE/3000 also displays the structure points during execution of the two program units, then halts at statement 40 in program unit TRACE3.

In figure 3-7, the DROPALL command deletes all PRINT and HALT paragraphs from the PRINT/HALT table.

The new PRINT and HALT paragraphs

```
*PRINT TRACE3
REVERSE ( )
```

and

```
*HALT
40
```

are entered. The GO command causes TRACE/3000 to display calls to and returns from subroutine REVERSE, and to display the values of the parameters passed to REVERSE. Again, the program halts at statement 40.

The DROP command in figure 3-7 is used to drop the sentence REVERSE from the PRINT/HALT table and the new PRINT paragraph

```
*PRINT REVERSE
K >9
```

is entered. This time TRACE/3000 displays all values of K greater than 9 because of the >9 identifier value condition clause.

In figure 3-8, the DROP command drops the sentence K from the PRINT/HALT table. A new PRINT paragraph

```
*PRINT
OUT
```

causes TRACE/3000 to display the values of array OUT.

```
*PRINT TRACE3
$LABEL

*PRINT REVERSE
$LABEL

*GO

10
        NAME                LAST NAME FIRST

20
30
CALL REVERSE
ENTER REVERSE
10
20
20
20
20
30
40
40
40
40
40
50
60
60
60
70
80
80
80
80
80
80
EXIT REVERSE
RETURN REVERSE
     JOHN BIGTOWN           BIGTOWN, JOHN
20
30
CALL REVERSE
ENTER REVERSE
10
20
20
20
20
30
40
40
40
40
40
50
60
60
60
70
80
80
80
80
80
80
80
EXIT REVERSE
RETURN REVERSE
     LOIS ANYONE            ANYONE, LOIS
20
30
CALL REVERSE
ENTER REVERSE
10
20
20
20
20
30
40
40
40
40
40
50
60
60
60
70
80
80
80
80
80
80
80
EXIT REVERSE
RETURN REVERSE
     ALI BABA               BABA, ALI
20
30
CALL REVERSE
ENTER REVERSE
10
20
20
20
20
30
40
40
40
40
40
50
60
60
60
70
80
80
80
80
80
80
80
EXIT REVERSE
RETURN REVERSE
     JAMES DOE              DOE, JAMES
```

Figure 3-6. Running Program XMPL3 Using the $LABEL Sentence (Sheet 1 of 3)

```
20                                      20
30                                      30
CALL REVERSE                            CALL REVERSE
ENTER REVERSE                           ENTER REVERSE
10                                      10
20                                      20
20                                      20
20                                      20
20                                      20
30                                      20
40                                      30
40                                      40
40                                      40
40                                      40
40                                      40
40                                      50
50                                      60
60                                      60
60                                      60
60                                      60
60                                      60
70                                      70
80                                      80
80                                      80
80                                      80
80                                      80
80                                      80
80                                      80
80                                      80
EXIT REVERSE                            EXIT REVERSE
RETURN REVERSE                          RETURN REVERSE
     JOHN DOUGHE          DOUGHE, JOHN          SPACE MANN          MANN, SPACE
20                                      20
30                                      30
CALL REVERSE                            CALL REVERSE
ENTER REVERSE                           ENTER REVERSE
10                                      10
20                                      20
20                                      20
20                                      20
30                                      20
40                                      30
40                                      40
40                                      40
40                                      40
50                                      40
60                                      40
60                                      40
60                                      50
60                                      60
70                                      60
80                                      60
80                                      60
80                                      70
80                                      80
80                                      80
80                                      80
80                                      80
80                                      80
EXIT REVERSE                            80
RETURN REVERSE                          EXIT REVERSE
     MARY MEEK            MEEK, MARY    RETURN REVERSE
                                               KING ARTHUR          ARTHUR, KING
```

Figure 3-6. Running Program XMPL3 Using the $LABEL Sentence (Sheet 2 of 3)

```
20
30
CALL REVERSE
ENTER REVERSE
10
20
20
20
20
20
20
20
30
40
40
40
40
40
40
40
50
60
60
60
60
60
70
80
80
80
80
EXIT REVERSE
RETURN REVERSE
     KARISSA GRANDTR        GRANDTR, KARISSA
20
30
CALL REVERSE
ENTER REVERSE
10
20
20
20
20
20
30
40
40
40
40
40
40
40
50
60
60
60
60
70
80
80
80
80
EXIT REVERSE
RETURN REVERSE
     JENNA GRANDTR          GRANDTR, JENNA
20
30
CALL REVERSE
ENTER REVERSE
10
20
20
20
20
20
20
30
40
40
40
40
40
40
40
50
60
60
60
60
70
80
80
80
80
EXIT REVERSE
RETURN REVERSE
     SWASH BUCKLER          BUCKLER, SWASH
20
30
CALL REVERSE
ENTER REVERSE
10
20
20
20
30
40
40
40
40
40
40
40
50
60
60
60
60
70
80
80
80
80
EXIT REVERSE
RETURN REVERSE
     KNEE BUCKLER           BUCKLER, KNEE
20
40
TRACE3
```

Figure 3-6. Running Program XMPL3 Using the $LABEL Sentence (Sheet 3 of 3)

```
*DROPALL

*PRINT TRACE3
REVERSE ()

*HALT
40

*GO

        NAME              LAST NAME FIRST

CALL REVERSE("J","B")
RETURN REVERSE
     JOHN BIGTOWN          BIGTOWN, JOHN
CALL REVERSE("L","B")
RETURN REVERSE
     LOIS ANYONE           ANYONE, LOIS
CALL REVERSE("A","A")
RETURN REVERSE
     ALI BABA              BABA, ALI
CALL REVERSE("J","B")
RETURN REVERSE
     JAMES DOE             DOE, JAMES
CALL REVERSE("J","D")
RETURN REVERSE
     JOHN DOUGHE           DOUGHE, JOHN
CALL REVERSE("M","D")
RETURN REVERSE
     MARY MEEK             MEEK, MARY
CALL REVERSE("S","M")
RETURN REVERSE
     SPACE MANN            MANN, SPACE
CALL REVERSE("K","M")
RETURN REVERSE
     KING ARTHUR           ARTHUR, KING
CALL REVERSE("K","A")
RETURN REVERSE
     KARISSA GRANDTR       GRANDTR, KARISSA
CALL REVERSE("J","G")
RETURN REVERSE
     JENNA GRANDTR         GRANDTR, JENNA
CALL REVERSE("S","G")
RETURN REVERSE
     SWASH BUCKLER         BUCKLER, SWASH
CALL REVERSE("K","B")
RETURN REVERSE
     KNEE BUCKLER          BUCKLER, KNEE
40
TRACE3
```

```
*DROP
REVERSE

*PRINT REVERSE
K >9

*GO

        NAME              LAST NAME FIRST

K=    10
K=    11
K=    12
K=    13
K=    14
     JOHN BIGTOWN          BIGTOWN, JOHN
K=    10
K=    11
K=    12
K=    13
     LOIS ANYONE           ANYONE, LOIS
K=    10
     ALI BABA              BABA, ALI
K=    10
K=    11
     JAMES DOE             DOE, JAMES
K=    10
K=    11
K=    12
K=    13
     JOHN DOUGHE           DOUGHE, JOHN
K=    10
K=    11
     MARY MEEK             MEEK, MARY
K=    10
K=    11
K=    12
     SPACE MANN            MANN, SPACE
K=    10
K=    11
K=    12
K=    13
     KING ARTHUR           ARTHUR, KING
K=    10
K=    11
K=    12
K=    13
K=    14
K=    15
K=    16
K=    17
     KARISSA GRANDTR       GRANDTR, KARISSA
K=    10
K=    11
K=    12
K=    13
K=    14
K=    15
     JENNA GRANDTR         GRANDTR, JENNA
K=    10
K=    11
K=    12
K=    13
K=    14
K=    15
     SWASH BUCKLER         BUCKLER, SWASH
K=    10
K=    11
K=    12
K=    13
K=    14
     KNEE BUCKLER          BUCKLER, KNEE
40
TRACE3
```

Figure 3-7. Running Program XMPL3 Using the Routine Parameter Condition and Identifier Value Condition Clauses

```
*DROP
K

*PRINT
OUT


*GO
        NAME                LAST NAME FIRST

OUT(1)="B"
OUT(2)="I"
OUT(3)="G"
OUT(4)="T"
OUT(5)="O"
OUT(6)="W"
OUT(7)="N"
OUT(8)=","
OUT(9)=" "
OUT(10)="J"
OUT(11)="O"
OUT(12)="H"
OUT(13)="N"
OUT(14)=" "
OUT(15)=" "
OUT(16)=" "
OUT(17)=" "
OUT(18)=" "
OUT(19)=" "
OUT(20)=" "
     JOHN BIGTOWN           BIGTOWN, JOHN
OUT(1)="A"
OUT(2)="N"
OUT(3)="Y"
OUT(4)="O"
OUT(5)="N"
OUT(6)="E"
OUT(7)=","
OUT(8)=" "
OUT(9)="L"
OUT(10)="O"
OUT(11)="I"
OUT(12)="S"
OUT(13)=" "
OUT(14)=" "
OUT(15)=" "
OUT(16)=" "
OUT(17)=" "
OUT(18)=" "
OUT(19)=" "
OUT(20)=" "
     LOIS ANYONE            ANYONE, LOIS
OUT(1)="B"
OUT(2)="A"
OUT(3)="B"
OUT(4)="A"
OUT(5)=","
OUT(6)=" "
OUT(7)="A"
OUT(8)="L"
OUT(9)="I"
OUT(10)=" "
OUT(11)=" "
OUT(12)=" "
OUT(13)=" "
OUT(14)=" "
OUT(15)=" "
OUT(16)=" "
OUT(17)=" "
OUT(18)=" "
OUT(19)=" "
OUT(20)=" "
     ALI BABA               BABA, ALI
OUT(1)="D"
OUT(2)="O"
OUT(3)="E"
OUT(4)=","
OUT(5)=" "
OUT(6)="J"
OUT(7)="A"
OUT(8)="M"
OUT(9)="E"
OUT(10)="S"
OUT(11)=" "
OUT(12)=" "
OUT(13)=" "
OUT(14)=" "
OUT(15)=" "
OUT(16)=" "
OUT(17)=" "
OUT(18)=" "
OUT(19)=" "
OUT(20)=" "
     JAMES DOE              DOE, JAMES
OUT(1)="D"
OUT(2)="O"
OUT(3)="U"
OUT(4)="G"
OUT(5)="H"
OUT(6)="E"
OUT(7)=","
OUT(8)=" "
OUT(9)="J"
OUT(10)="O"
OUT(11)="H"
OUT(12)="N"
OUT(13)=" "
OUT(14)=" "
OUT(15)=" "
OUT(16)=" "
OUT(17)=" "
OUT(18)=" "
OUT(19)=" "
OUT(20)=" "
     JOHN DOUGHE            DOUGHE, JOHN
OUT(1)="M"
OUT(2)="E"
OUT(3)="E"
OUT(4)="K"
OUT(5)=","
OUT(6)=" "
OUT(7)="M"
OUT(8)="A"
OUT(9)="R"
OUT(10)="Y"
OUT(11)=" "
OUT(12)=" "
OUT(13)=" "
OUT(14)=" "
OUT(15)=" "
OUT(16)=" "
OUT(17)=" "
OUT(18)=" "
OUT(19)=" "
OUT(20)=" "
     MARY MEEK              MEEK, MARY
```

Figure 3-8. Using TRACE/3000 to Display the Values of Array OUT (Sheet 1 of 2)

```
OUT(1)="M"
OUT(2)="A"
OUT(3)="N"
OUT(4)="N"
OUT(5)=","
OUT(6)=" "
OUT(7)="S"
OUT(8)="P"
OUT(9)="A"
OUT(10)="C"
OUT(11)="E"
OUT(12)=" "
OUT(13)=" "
OUT(14)=" "
OUT(15)=" "
OUT(16)=" "
OUT(17)=" "
OUT(18)=" "
OUT(19)=" "
OUT(20)=" "
     SPACE MANN            MANN, SPACE
OUT(1)="A"
OUT(2)="R"
OUT(3)="T"
OUT(4)="H"
OUT(5)="U"
OUT(6)="R"
OUT(7)=","
OUT(8)=" "
OUT(9)="K"
OUT(10)="I"
OUT(11)="N"
OUT(12)="G"
OUT(13)=" "
OUT(14)=" "
OUT(15)=" "
OUT(16)=" "
OUT(17)=" "
OUT(18)=" "
OUT(19)=" "
OUT(20)=" "
     KING ARTHUR           ARTHUR, KING
OUT(1)="G"
OUT(2)="R"
OUT(3)="A"
OUT(4)="N"
OUT(5)="D"
OUT(6)="T"
OUT(7)="R"
OUT(8)=","
OUT(9)=" "
OUT(10)="K"
OUT(11)="A"
OUT(12)="R"
OUT(13)="I"
OUT(14)="S"
OUT(15)="S"
OUT(16)="A"
OUT(17)=" "
OUT(18)=" "
OUT(19)=" "
OUT(20)=" "
     KARISSA GRANDTR       GRANDTR, KARISSA
OUT(1)="G"
OUT(2)="R"
OUT(3)="A"
OUT(4)="N"
OUT(5)="D"
OUT(6)="T"
OUT(7)="R"
OUT(8)=","
OUT(9)=" "
OUT(10)="J"
OUT(11)="E"
OUT(12)="N"
OUT(13)="N"
OUT(14)="A"
OUT(15)=" "
OUT(16)=" "
OUT(17)=" "
OUT(18)=" "
OUT(19)=" "
OUT(20)=" "
     JENNA GRANDTR         GRANDTR, JENNA
OUT(1)="B"
OUT(2)="U"
OUT(3)="C"
OUT(4)="K"
OUT(5)="L"
OUT(6)="E"
OUT(7)="R"
OUT(8)=","
OUT(9)=" "
OUT(10)="S"
OUT(11)="W"
OUT(12)="A"
OUT(13)="S"
OUT(14)="H"
OUT(15)=" "
OUT(16)=" "
OUT(17)=" "
OUT(18)=" "
OUT(19)=" "
OUT(20)=" "
     SWASH BUCKLER         BUCKLER, SWASH
OUT(1)="B"
OUT(2)="U"
OUT(3)="C"
OUT(4)="K"
OUT(5)="L"
OUT(6)="E"
OUT(7)="R"
OUT(8)=","
OUT(9)=" "
OUT(10)="K"
OUT(11)="N"
OUT(12)="E"
OUT(13)="E"
OUT(14)=" "
OUT(15)=" "
OUT(16)=" "
OUT(17)=" "
OUT(18)=" "
OUT(19)=" "
OUT(20)=" "
     KNEE BUCKLER          BUCKLER, KNEE
40
TRACE3

*DROP
OUT

*SET TRACE3
STOPNOW=      0/1
STOPNOW=      1
#STOPNOW=Q+3
Q+3=%000001

*GO

END OF PROGRAM
```

Figure 3-8. Using TRACE/3000 to Display the Values of Array OUT (Sheet 2 of 2)

Finally, a SET paragraph is used to change the value of STOPNOW from 0 to 1. Also, to demonstrate using the SET command to locate stack addresses, the address of STOPNOW is determined by entering

#STOPNOW=

TRACE/3000 displays Q+3. Then, after Q+3= is entered, TRACE/3000 displays the contents of location Q+3 (which is %000001, the value of STOPNOW).

3-4. USING TRACE/3000 IN A BATCH JOB

Figure 3-9 shows a FORTRAN/3000 source program and a BATCH file, both of which are punched on cards and arranged in the correct order to run in batch mode.

The first two cards are a :JOB command card and a :FORTGO command card. The :JOB command initiates a batch job; the :FORTGO command compiles, prepares, and executes a FORTRAN/3000 source program. See the *MPE/3000 Operating System Reference Manual* for descriptions of these commands.

The $TRACE control statement (card three) informs TRACE/3000 to monitor *identifiers* A, I, SQRT, ROOT, and RCPL in *program unit* TRACE1. (Note that since this program example consists of only one program unit, the *program unit* parameter could have been omitted.) The :EOD command card informs the compiler that there are no more lines of code in this program.

The BATCH file, which must follow the :EOD card, is begun with a $TRACESTART statement and terminated with a $TRACEEND statement. A blank record separates the last sentence (RCPL) in the PRINT paragraph from the $TRACEEND card.

The :EOJ command terminates the job.

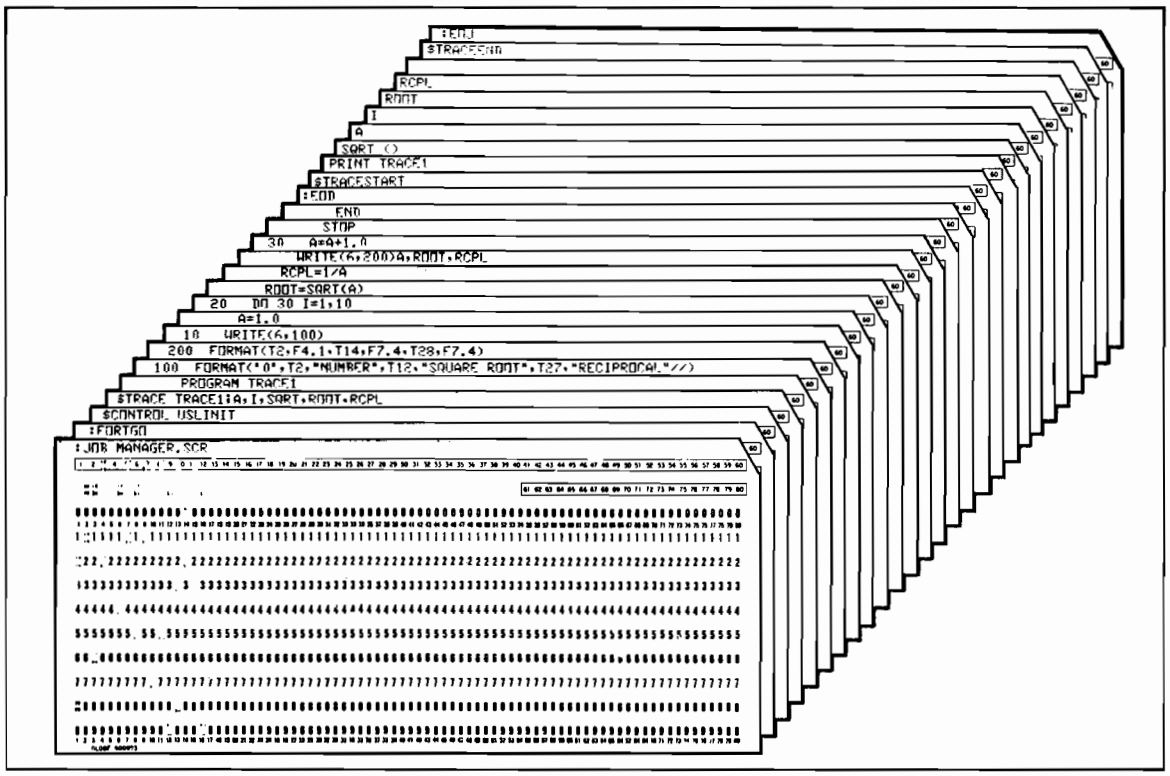


Figure 3-9. FORTRAN/3000 Source Program and BATCH File Punched on Cards

Figure 3-10 shows the output (printed on a line printer) resulting when the card deck is run.

```
:JOB    MANAGER.SCR, PUB
 PRI= DS;  INPRI= 13;  TIME= ?
 JOB NUMBER = #J1
 THU, JUL 10, 1975, 11:33 AM
 HP32000C.00.50

:FORTGO

PAGE 0001   HEWLETT-PACKARD 32102A.01.4   FORTRAN/3000   THU, JUL 10, 1975, 11:33 AM


           $CONTROL USLINIT
           $TRACE TRACE1;A,I,SQRT,ROOT,RCPL
                 PROGRAM TRACE1
            100  FORMAT('0',T2,"NUMBER",T12,"SQUARE ROOT",T27,"RECIPROCAL"//)
            200  FORMAT(T2,F4.1,T14,F7.4,T28,F7.4)
            10   WRITE(6,100)
                 A=1.0
            20   DO 30 I=1,10
                 ROOT=SQRT(A)
                 RCPL=1/A
                 WRITE(6,200)A,ROOT,RCPL
            30   A=A+1.0
                 STOP
                 END


****  NO ERRORS,  NO WARNINGS;   PROGRAM UNIT COMPILED  ****
COMPILATION TIME    0.956 SECONDS    ELAPSED TIME     3.571 SECONDS
TOTAL COMPILATION TIME   0:00:02
TOTAL ELAPSED TIME       0:00:04

END OF COMPILE

END OF PREPARE

HELLO TRACE   HP32222A.02.1

NUMBER     SQUARE ROOT     RECIPROCAL


A= 1.000000
I=      1
CALL SQRT( 1.000000     )
RETURN SQRT= 1.000000
ROOT= 1.000000
RCPL= 1.000000
 1.0          1.0000          1.0000
A= 2.000000
I=      2
CALL SQRT( 2.000000     )
RETURN SQRT= 1.414214
ROOT= 1.414214
RCPL= .5000000
 2.0          1.4142           .5000
A= 3.000000
I=      3
CALL SQRT( 3.000000     )
RETURN SQRT= 1.732051
ROOT= 1.732051
RCPL= .3333333
 3.0          1.7321           .3333
A= 4.000000
I=      4
CALL SQRT( 4.000000     )
RETURN SQRT= 2.000000
ROOT= 2.000000
RCPL= .2500000
 4.0          2.0000           .2500
```

Figure 3-10. Using TRACE/3000 During a Batch Job (Sheet 1 of 2)

```
A= 5.000000
I=      5
CALL SQRT( 5.000000     )
RETURN SQRT= 2.236068
ROOT= 2.236068
RCPL= .2000000
 5.0          2.2361          .2000
A= 6.000000
I=      6
CALL SQRT( 6.000000     )
RETURN SQRT= 2.449490
ROOT= 2.449490
RCPL= .1666667
 6.0          2.4495          .1667
A= 7.000000
I=      7
CALL SQRT( 7.000000     )
RETURN SQRT= 2.645751
ROOT= 2.645751
RCPL= .1428571
 7.0          2.6458          .1429
A= 8.000000
I=      8
CALL SQRT( 8.000000     )
RETURN SQRT= 2.828427
ROOT= 2.828427
RCPL= .1250000
 8.0          2.8284          .1250
A= 9.000000
I=      9
CALL SQRT( 9.000000     )
RETURN SQRT= 3.000000
ROOT= 3.000000
RCPL= .1111111
 9.0          3.0000          .1111
A= 10.00000
I=     10
CALL SQRT( 10.00000     )
RETURN SQRT= 3.162278
ROOT= 3.162278
RCPL= .9999999E-01
10.0          3.1623          .1000
A= 11.00000
 END OF PROGRAM
:EOJ

CPU (SEC) = 15
ELAPSED (MIN) = 2
THU, JUL 10, 1975, 11:34 AM
END OF JOB
```

Figure 3-10. Using TRACE/3000 During a Batch Job (Sheet 2 of 2)

3-5. TRACE/3000 LIST FILE, TRCLIST

TRACE/3000 output will be sent to the standard output file (a terminal in an interactive session or a line printer in a batch job) unless the TRACE/3000 list file, TRCLIST, is equated to another file with a :FILE equation.

For example, to list the output on the line printer, instead of the terminal, in an interactive session, enter:

```
:FILE TRCLIST;DEV=LP
:RUN PROG
```

3-6. PRINT/HALT TABLE SIZE

All PRINT and HALT sentences from both the BATCH and INTERACTIVE files are entered into the PRINT/HALT table by TRACE/3000. As the number of sentences is varied (by changing the INTERACTIVE file during an interactive session), TRACE/3000 changes the PRINT/HALT table accordingly.

The size of the PRINT/HALT table depends on the types of condition clauses used in PRINT and HALT sentences. Table 3-1 shows the sizes for the different condition clauses. TRACE/3000 sets the maximum size of the PRINT/HALT table at 200 computer words.

Table 3-1. Calculating PRINT/HALT Sentence Size

Note: The base word length of any PRINT/HALT sentence is 3 words.	
CONDITION CLAUSE TYPE	**SIZE (COMPUTER WORDS)**
Subscript Value Condition Clause	3
Label Condition Clause	1
Use Condition Clause	2
Identifier Value Condition Clause	1 (If the clause contains a variable, or as many words as necessary if the clause contains a constant, depending on the constant type.)

EQUIVALENT CONSTANT TYPES IN TRACE/3000 PARAGRAPHS

APPENDIX A

When you enter PRINT, HALT, or SET paragraphs, TRACE/3000 allows some constants of different types to substitute for constants of an expected type. For example,

```
*SET MAIN'
ITEM=  200/300.25
ITEM=  300
```

In the above example, ITEM is an integer variable, but the user changed the integer value by entering a real constant (300.25). TRACE/3000 accepts the real constant in place of the integer constant, but truncates the real constant to form an integer.

Table A-1 indicates which constant types are acceptable replacements for other constant types. The table also indicates any restrictions on the relational operators that can be used in PRINT and HALT sentences for constants of various types. In the table, a short string is a string of one or two characters only.

TRACE/3000 prints a BAD SYNTAX message:

1. If the constant type is not equivalent to the expected type.
2. If the relational operator is illegal for the constant type.
3. If an overflow occurs in the PRINT/HALT table in the process of converting a constant to the proper type.

Table A-1. Equivalent Constant Types

TYPE OF CONSTANT EXPECTED BY TRACE/3000	ALLOWABLE RELATIONAL OPERATORS	TYPE OF CONSTANT ALLOWED BY TRACE/3000
LOGICAL,INTEGER,SHORT STRING	=	LOGICAL,INTEGER,SHORT STRING
INTEGER,(SPL) DOUBLE INTEGER, REAL,DOUBLE PRECISION (REAL)	all	INTEGER,(SPL) DOUBLE INTEGER, REAL,DOUBLE PRECISION (REAL)
COMPLEX	=	COMPLEX
STRING	=	STRING (of length not greater than the length of the expected string)

ABBREVIATIONS IN TRACE/3000 PARAGRAPHS

APPENDIX B

Abbreviated program unit names and identifier names can be used in PRINT, HALT, CHECK, DROP, and SET paragraphs ($TRACE control statements, however, must contain the full program unit and identifier names). The abbreviations consist of at least the first letter of the full name, followed by as many characters as necessary to identify the desired full name (subject to the rules contained in the next paragraph).

After compilation of the source program, the compiler passes a list of all program unit names and identifier names in the $TRACE control statements to TRACE/3000.[1] TRACE/3000 stores these names in a name dictionary, in alphabetical order. This means that the name A appears before AA, and AA appears before AAA in the list. Whenever TRACE/3000 encounters a name in a sentence, it searches the name dictionary. The name chosen is the first name encountered in the dictionary that has the same first letter as the first letter in the abbreviation, or that has the same first *letters* if more than one letter is used in the abbreviation.

EXAMPLE:

The following $TRACE control statements are read during compilation of a source program:

```
$TRACEMYPROG;$DELETE,IVAR1,IVAR2,IVAR3,FIXIT
$TRACE FIXIT;$DELETE,AJAX,B,BB
```

The compiler passes the list of names to TRACE/3000. The names are stored in the name dictionary as follows:

```
AJAX
B
BB
FIXIT
IVAR1
IVAR2
IVAR3
MYPROG
```

If TRACE/3000 encounters the PRINT paragraph

```
PRINT M
I=  6
```

it searches the name dictionary and decides that M stands for MYPROG (MYPROG is the first name in the name dictionary that has the same first letter as the first letter of the abbreviation), and that I stands for IVAR1 (IVAR1 is the first name in the name dictionary that has the

---

[1]The compiler also includes all program unit labels and user-defined routine names (for structure points) unless $DELETE is the first identifier in the $TRACE control record identifier list.

same first letter as the first letter of the abbreviation). Thus, TRACE/3000 assumes that the PRINT paragraph means

```
PRINT MYPROG
IVAR1=   6
```

Using the above name dictionary, TRACE/3000 responds as shown below for the following paragraph:

```
*CHECK F
FIXIT
B=B
```

"B" is the full name of a variable and cannot be abbreviated; BB cannot be abbreviated since TRACE/3000 always takes B to stand for the identifier B in the name dictionary shown above.

INDEX

INDEX