HP 3000

SYMBOL TRACE

Computer
Museum

03000-90015A

November 1972

©Copyright, 1972, by HEWLETT-PACKARD COMPANY, 11000 Wolfe Road, Cupertino, California. All rights reserved.
Printed in the U.S.A.

List of Effective Pages

Pages	Effective Date
Title	Nov. 1972
Copyright	Nov. 1972
iii	Nov. 1972
v to vi	Nov. 1972
1-1 to 1-3	Nov. 1972
2-1 to 2-27	Nov. 1972
3-1 to 3-5	Nov. 1972
A-1 to A-2	Nov. 1972
B-1 to B-2	Nov. 1972

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Printing History

Part No.	Date	Update Package	Date
03000-90015A	Nov. 1972		

PREFACE

HP 3000 Symbol Trace (03000-90015) describes the HP 3000 Diagnostic Aid for User Programs (TRACE/3000).

TRACE/3000, a subsystem of MPE/3000 (Multiprogramming Executive Operating System), helps users find and correct errors in their FORTRAN/3000 and SPL/3000 source programs.

- Section I of this manual describes the components and uses of TRACE/3000.
- Section II explains the forms and uses of the TRACE/3000 files, tables, and paragraphs.
- Section III details how to use TRACE/3000 in batch and interactive modes.

The reader should be familiar with the FORTRAN/3000 and/or SPL/3000 programming languages and MPE/3000. For further information on these subjects see

HP 3000 FORTRAN (03000-90007)

HP 3000 Systems Programming Language (03000-90002)

HP 3000 Multiprogramming Executive Operating System (03000-90005)

CONTENTS

Computer Museum

FIGURE

SECTION I
Introduction

TRACE/3000 is an MPE/3000 (Multiprogramming Executive Operating System) subsystem designed to aid the user in finding program logic errors in SPL/3000 and FORTRAN/3000 programs. TRACE/3000 helps the user follow the path of execution, computation of values and manipulation of data in his programs by printing information about program identifiers during program execution.

USING TRACE/3000

TRACE/3000 involves several stages of user activity:

- User source program preparation
- Source program compilation
- User program execution

The user prepares his source program by punching his source language statements on cards (for example). To this, the user places $TRACE records (in front of the source statements) and PRINT and HALT sentences cards (behind the source deck.) $TRACE records inform the compiler which program identifiers the user desires TRACE/3000 to monitor, while PRINT and HALT sentences tell TRACE/3000 when TRACE/3000 is to print information about program identifiers (e.g., when a variable value exceeds a certain limit, or a certain procedure is called). compilers ?

After preparing the source deck, the user compilers the source program. During compilation, the compiler reads the $TRACE records and inserts calls to the TRACE/3000 subprogram into the object code generated by the compiler whenever a program identifier mentioned in a $TRACE card appears in the proper position in the source program.

After compilation, the user loads the program into memory and starts user program execution. Before execution begins, TRACE/3000 reads the PRINT and HALT sentences from the job input device, and stores them in a Print/Halt table. During user program execution, TRACE/3000 searches the Print/Halt table whenever a program identifier appears that was mentioned in a $TRACE record. If the current value or usage of the program identifier satisfies the PRINT or HALT sentence, TRACE/3000 prints information about the identifier to inform the user of the condition.

TRACE/3000 MODES OF OPERATION

TRACE/3000 can be used in either an MPE/3000 SESSION (Interactive mode) or in a JOB (Batch mode). Executing the user program in interactive mode (during an MPE/3000 Session) allows the following features:

- The user can enter interactive commands from the SESSION terminal (see Section II).
- Control returns to the interactive (SESSION) terminal after a HALT sentence is executed.
- The user can regain control from the user program to the interactive terminal by pressing the TRACE key (control Y).

Executing the user program in BATCH mode (during an MPE/3000 Job) allows the following features:

- Execution terminates and the user program is flushed from the MPE/3000 system if a HALT sentence is executed.
- The PRINT/Halt table cannot be modified once TRACE/3000 reads the PRINT and HALT sentences into the Print/Halt table.

TRACE/3000 PROGRAM IDENTIFIERS

TRACE/3000 recognizes four types of program identifiers: variables, arrays, labels and routines. Table 1-1 shows the correspondence between TRACE/3000 identifier types and FORTRAN/3000 or SPL/3000 identifier types.

Table 1-1. TRACE/3000 Identifiers

TRACE/3000 Identifier	FORTRAN/3000	SPL/3000
variable	simple variable	simple variable
		@pointer
array	array	array, pointer
label	statement label	label
routine	function subprogram subroutine subprogram statement function	procedure subroutine

STRUCTURE POINTS

Structure points are points of passage into and out of a program unit. TRACE/3000 recognizes four such structure points:

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

Every program unit has at least one Enter point and one Exit point. Call and Return points come in pairs, and there is no limit to the number of pairs which can exist in a program unit. Call occurs just prior to a routine reference or call; Return occurs immediately after the routine reference or call.

SECTION II
TRACE/3000 Commands and Language

$TRACE CONTROL RECORDS

In order to monitor a source program identifier, the user creates a $TRACE control record bearing the identifier name and the program unit to which the identifier belongs. More than one record can contain the same program unit name. A program unit is defined as either the main program unit or any routine operating with the main program.

The $TRACE control record takes the form:

$TRACE program unit name; identifier,identifier, . . . ,identifier

program unit name	is the name of the main program or routine for which the record is created. TRACE/3000 also recognizes MAIN′ (in FORTRAN) or OB′ (in SPL) to mean the main program unit if the user did not otherwise name it. The name can also be omitted (null), in which case the compiler assumes the main program unit.
identifier	is a specific identifier name (variable, array, label, routine) or $DELETE found in the program unit appearing on the $TRACE record. Only those identifiers contained in the program unit can appear on a record for that program unit (otherwise, the compiler e omits an error message).

During compilation the source language compiler inserts calls to TRACE/3000 to monitor all labels and structure points during execution of any program unit mentioned in a $TRACE record.[1] The user can insert the general term $DELETE as the first identifier in the identifier list to suppress this feature, and if desired, he can then insert the names of specific labels or routines into the identifier list. The compiler inserts calls to TRACE/3000 into the user program to monitor only the specific labels and routines mentioned in the $TRACE card for the program unit. During execution of the user program, TRACE/3000 prints information about structure points only if $FORM appears in a PRINT/HALT sentence mentioning the appropriate program unit. No calls to TRACE/3000 to monitor structure points are made by the compiler unless the program unit name appears as the program unit in a $TRACE record. See "PRINT and HALT Paragraphs" and "PRINT and HALT Sentence Results", in this section.

[1] FORTRAN/3000 does not report structure points for FORTRAN intrinsic functions and some library routines. The user does not know when a call or return from such a routine occurs.

In a $TRACE card record, $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 null (omitted), the semi-colon (;) is separated from $TRACE by only one space. 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, LABEL1,   LABEL2, PROG15
```

TRACE/3000 FILES AND TABLES

TRACE/3000 creates a Batch File, Interactive File and Print/Halt Table during TRACE/3000 execution. The Batch File and Print/Halt Table are always used, while the Interactive File is accessed only if TRACE/3000 is running in interactive mode. The file/table structures and uses are described below.

TRACE/3000 Batch File

The TRACE/3000 Batch File exists for any program unit monitored by TRACE/3000 regardless of the mode in which the programs are running (batch or interactive). The Batch File consists of a $TRACESTART record to open the file, an optional body of PRINT or HALT paragraphs (described in the "PRINT and HALT Paragraphs" section) and a $TRACEEND record which closes the Batch File. The user physically locates the Batch File as the first data on the job input device.

Computer Museum

The form of a $TRACESTART record is

$TRACESTART [ABORT]

The number of computer words necessary for the Print/Halt Table depends upon the number of PRINT/HALT sentences in the Batch File and the complexity of the sentences. Consult Table 3-1 to determine the desired Print/Halt Table size. The MPE/3000 operating system allocates 200 computer words for the table.

The optional parameter ABORT signifies that TRACE/3000 is to terminate the user program if the Print/Halt Table overflows or if TRACE/3000 discovers an error in a PRINT/HALT paragraph. If the ABORT parameter is not used, then TRACE/3000 ignores the sentence in error (and the rest of the paragraph) or the sentences overflowing the Print/Halt Table and continues executing.

PRINT/HALT Paragraphs are the body of the Batch File. The Batch File can contain as many PRINT/HALT Paragraphs as the size of the Print/Halt Table will allow. If the Batch File contains more paragraphs than the Print/Halt Table can hold, TRACE/3000 prints an OVERFLOW message and takes the action indicated by the use of the ABORT parameter in the $TRACESTART record. Each line of a paragraph exists on a separate record and starts in column 1 of the record. The first paragraph in the file must contain a program unit name or an error results (see "Paragraphs").

The Batch File is closed by a $TRACEEND record. The word $TRACEEND starts in column 1 of the record. Failure to observe the proper Batch File opening and closing record form causes TRACE/3000 to terminate execution of the user program, and print the message

```
BAD TRACE FILE
```

Interactive File

The Interactive File accepts PRINT, HALT, SET, DROP, and CHECK paragraphs. TRACE/3000 opens the Interactive File only if the user program is running in interactive mode (during an MPE/3000 Session).

TRACE/3000 activates the Interactive File under two conditions:

- Immediately before starting execution of the user program
- After the user program satisfies a HALT sentence

In the first case above, TRACE indicates that the Interactive File has been activated by typing the message

```
MODE=
```

After the user answers the questions correctly, TRACE/3000 types an asterisk in column 1 of the line. The user can now enter paragraphs. In the other case, TRACE indicates that the Interactive File has been accessed by typing an asterisk in column 1 of a line on the user keyboard device. The user then enters PRINT, HALT, DROP, SET, or CHECK paragraphs (see "Paragraphs" in this section). After entering paragraphs, the user returns control to the user program and deactivates the Interactive File by typing a GO command (see "GO Command" in this section).

PRINT/HALT TABLE

The Print/Halt Table consists of PRINT and HALT sentences entered by the user into the Batch or Interactive Files. TRACE/3000 consolidates all PRINT and HALT sentences from both files into the Print/Halt Table. During user program execution, TRACE/3000 searches the Print/Halt Table for any PRINT/HALT sentences applying to program identifiers appearing in $TRACE records. If the user program satisfies a PRINT or HALT sentence, TRACE/3000 takes the appropriate action (see "PRINT and HALT Sentence Results" in this section).

PRINT and HALT sentences can be added or deleted from the Print/Halt Table once user program execution begins, whenever the Interactive File is accessed (only if the user program is running in interactive mode). The user can then enter PRINT, HALT or DROP paragraphs to modify the Print/Halt Table (see "Paragraphs" in this section).

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 the other paragraphs can be entered into the Interactive File only. Paragraphs entered into the Batch File differ slightly from paragraphs entered into the Interactive File, but all paragraphs are structured in the same way.

Each paragraph starts with a record/line 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 "Program Unit Name in Paragraphs" in this section). For the Batch File, the paragraph type starts in column 1 of the record, and ends (after the optional program unit name) with a string of blanks through column 80 of the record. For the Interactive File, TRACE/3000 prints an asterisk (*) in column 1 of the line to indicate readiness to accept paragraphs. The user starts the paragraph type in column 2 of the line and terminates the line with a carriage return.

The paragraph sentences follow the paragraph type on the next record/line. Each sentence starts in column 1 of the record/line. Sentences in the Batch File end with a string of blanks through column 80 of the record, while sentences on the Interactive File end with a carriage return. The user can enter as many sentences as he wishes, subject to the size of the Print/Halt Table (for PRINT or 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 or by the $TRACEEND record (which closes the Batch File). If the paragraph end is a blank record, the user can insert another PRINT or HALT 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 signify readiness to accept another paragraph.

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 names mentioned in $TRACE records in front of the user program units can be used in paragraphs. If the main user program unit has no name, TRACE/3000 assigns the name MAIN′ (for FORTRAN/3000 programs) or OB′ (for SPL/3000 programs).

The opening paragraph of the Batch File and the Interactive File must contain a program unit name. If no program unit name appears in an opening paragraph type, TRACE/3000 treats the paragraph as having incorrect form.

In each paragraph following the first one, the user 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 the user does 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 Interactive and Batch File need have a program unit name. All other paragraphs can omit the program unit name. Those paragraphs will apply to the same program unit as the first paragraph in the Batch or Interactive File.

PRINT and HALT Paragraphs

PRINT and HALT paragraphs are used to enter PRINT and HALT sentences into the Print/Halt Table through the Batch or Interactive File. PRINT and HALT sentences inform the user about program data and logic flow during program execution. The form of a PRINT/HALT paragraph is

$$\left.\begin{array}{l}\textit{PRINT}\\ \textit{HALT}\end{array}\right\}\ \textit{program unit name}$$

sentence

sentence

.

.

.

null record terminator

program unit name	is the name of the program unit to which all the sentences in the paragraph apply. 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 program unit in the immediately preceding paragraph.
null record terminator	is a blank record (all 80 columns blank) for the Batch File, or a carriage return in column 1 of a line for the Interactive File.
sentence	consists of an identifier name (including the generic identifiers $FORM and $LABEL) optionally followed by one or more *condition clauses* allowed for the identifier type. The identifier name used in the sentence must appear in a $TRACE control record having the same program unit name that is assigned to the PRINT/HALT sentence. $LABEL (if unmodified by *condition clauses*) gives a complete listing of the labels encountered during execution of the program unit (unless $DELETE was specified in a $TRACE record for the program unit). $FORM (if unmodified by *condition clauses*) gives a complete listing of structure points encountered during execution of the program unit (unless $DELETE was specified in a $TRACE record for the program unit). The form of a *sentence* is

array C1 C2 C3 C4

or

variable C2 C3 C4

or

label C3 C4

or

$LABEL C3 C4

or

routine P1 C3 C4

or

$FORM P1 C3 C4

C1, C2, C3, and C4 are *condition clauses* which specify conditions related to the identifier in the *sentence. Condition clauses,* if used, must appear in the order shown above. The clauses are separated from the *identifier* and from each other by one or more blanks, although no spaces can exist within individual *condition clauses.*

C1 — SUBSCRIPT VALUE CONDITION. Condition clause C1 is used only with array-type identifiers. (Identifier type is determined by its source program definition.) The form is

(* *relop integer primary*)

relop stands for 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.

Below are examples of C1-type condition clauses used in PRINT/HALT paragraphs: The first paragraph appears as in a Batch File; the second paragraph appears as in an Interactive File.

```
PRINT PROGRAM1
ARRAY1     (*=6)
DUSTY   (*<MAXIMUM)
SORT     (*>=34)
```
blank record

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

The asterisk (*) in the condition clause stands for the actual subscript expression in the user source program. During program execution, TRACE/3000 substitutes the value of the array subscript for the asterisk, thus forming an arithmetic expression that can be evaluated. If the arithmetic expression is true, the condition clause is true.

C2 – IDENTIFIER VALUE CONDITION.[1] Condition clause C2 is used with array and variable-type identifiers to test the value of a data element. The clause is separated from condition clause C1 by one or more blanks, or from the identifier by one or more blanks if C1 is not used. The form for C2 is

relop primary

relop	has the same definition as in C1.
primary	is either a variable or a constant.

[1]The FORTRAN/3000 compiler does not insert calls to TRACE/3000 into the user program object code after instructions to read values for array elements from a peripheral device. Consequently, the user does not know that array elements have changed unless the element was changed individually.

The following examples (in Batch File) show how condition clause C2 joins with the array element, variable, or array, and condition clause C1 to form an arithmetic expression. If the arithmetic expression is true, then condition clause C2 is true.

```
PRINT
ARR   (*=10)    =MAXIMUM
HOLD    >MINIMUM
ARR (*=24)      =1000
```
blank record

C3— LABEL CONDITION. Condition clause C3 can be used with all identifier types. C3 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 element of the second label. If the identifier in the identifier clause falls between the two labels, the condition clause is true. If the identifier exists after the second label or before the first label, the condition clause is false. C3 is separated from the preceding condition clause or the identifier itself by one or more blanks. The form for the condition clause is

label term-label term

label term	is either a label-type identifier or an asterisk (*). **-label term* denotes "from the beginning" of the program unit to the second label. *label term-** denotes from the first label to the "end of program unit". The second label must physically follow the first label in the source program.

The following are examples (in Batch File) of C3-type condition clauses used in PRINT and HALT paragraph sentences:

```
PRINT EXON
ARR   (*=10)   =MAX         LABEL1-LABEL2
MAXVAL   *-LABEL2
VALUE   (*=8)    LABEL-*
```
blank record

C4 — USE CONDITION. Condition clause C4 can be used with all identifier types. C4 is separated from the preceding condition clause or from the identifier by one or more blanks. Condition clause C4 is referenced only if all the preceding condition clauses in the sentence (if any) are true. The form of the clause is

@integer primary

integer primary	is either an integer variable or an integer constant. The type is determined by the primary's definition in the user source program.

TRACE/3000 initially stores the constant or evaluates and stores the variable. A value of less than one is stored as one. If *integer primary* has a value of n, condition clause C4 is false for the first n—1 times it is referenced. (Remember that the clause is not referenced unless all previous condition clauses in the sentence are true.) The nth time the 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 at the outset.

The following are examples (in Batch File) of C4-type condition clauses used in PRINT and HALT paragraph sentences:

```
HALT PROG4
VALUE  @2
ARR    (*<24)    =1000  LABEL1-LABEL2    @44
NUMBER    >MAXIMUM     @USE
```
blank record

P1 — ROUTINE PARAMETER CLAUSE. P1 consists of a pair of empty parentheses "( )" separated from the routine name or $FORM by one or more blanks. P1 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 (see "Structure Points" in section I). Some examples (in Batch File) of P1-type clauses used in PRINT and HALT paragraph sentences are

```
PRINT PROG6
FIXIT ()    LABEL1-LABEL6
CAL         ()       @4
```
blank record

```
HALT
$FORM   ()   *-FINISH    @2
```
blank record

PRINT and HALT Sentence Execution Conditions

A PRINT or HALT sentence executes if, and only if, the following conditions are satisfied by the user program.

1. The program unit assigned to the sentence paragraph is the program unit which called TRACE/3000.
2. The identifier in the sentence matches the identifier described in the TRACE/3000 calling sequence located in the user program object code. $LABEL matches with any label and $FORM matches with any structure point in the program unit.
3. All condition clauses in the sentence are true.

For example, the PRINT sentence below executes only if the call to TRACE/3000 occurred in the main program unit, and if the call referenced an array called ARR whose subscript must be less than 100 and whose value must be equal to 60. The call to TRACE/3000 must also exist between the program labels LABEL1 and LABEL2. All the former conditions must have been true three times previously, thus making the use condition true and the entire PRINT statement true.

```
PRINT MAIN'
ARR  (*<1ØØ)  =6Ø   LABEL1-LABEL2  #4
```

A PRINT or HALT sentence can exist without any condition clauses at all. The PRINT statement below executes only if the call to TRACE/3000 occurred in the program named MYPROG and the identifier encountered in the user program (VALUE) matches the identifier in the sentence.

```
PRINT MYPROG
VALUE
```

PRINT and HALT Sentence Results

When a PRINT or HALT sentence is evaluated as true, a message to the user is printed on a list device. The form of that message depends upon the type of identifier referenced in the sentence, as follows.

Type	Printout
variable	*variable = value*
array	*array (value of array element subscript) = value*
label	*name of label*
routine	*CALL routine name* *RETURN routine name*
$FORM	*ENTER program unit name* *CALL routine name* *RETURN routine name* *EXIT program unit name*

If the identifier is a variable, the variable name is printed out along with the value of the variable. For example:

```
MAXVAL=240
MYVAL=60
XRAY=2
```

If the identifier is an array, the array name and subscript of the array element referenced in the calling sequence to TRACE/3000 are printed out along with the value of the array element. For example:

```
ARR( 16)=400
MYARRAY( 100)=500
YURARRAY( 1)= 1
```

If the identifier is a label, then TRACE/3000 merely prints out the name of the label. For example:

```
LABEL1
MYLABEL
START
DONE
```

If the identifier is a routine, then the printout format is

CALL routine name

RETURN routine name

TRACE/3000 types *CALL routine name* when the call to the routine is executed, and types *RETURN routine name* when control is passed back to user program. If the PRINT sentence contains a P1 condition clause, e.g., MYROUTINE ( ), the printout format will include a list of the parameter values at the time of the routine call. For example:

```
CALL SWITCHIT(13,4,4,7)
RETURN SWITCHIT
```

Computer Museum

If the routine is a function (that is, a value associated with the routine's name is returned after routine execution), the *RETURN* form includes the value of the function. For example:

```
CALL FIXNUM(3,88)
RETURN FIXNUM=560
```

If $FORM appears in a PRINT/HALT sentence, TRACE/3000 prints out the structure points encountered during execution of the program unit mentioned in the PRINT/HALT paragraph (unless $DELETE was used in the $TRACE record). When execution enters into the program unit, TRACE/3000 types

ENTER program unit name[1]

When execution exits the program unit, TRACE/3000 types

EXIT program unit name

During execution through the program unit, TRACE/3000 also types out structure points for any user-defined routines whether or not the routine is mentioned in a $TRACE record. When the routine is entered, TRACE/3000 types

CALL routine name

and when control returns to the calling program unit, TRACE/3000 types

RETURN routine name

[1] TRACE/3000 acts as if the main program unit is always entered, so the structure point *ENTER main program unit* is never printed.

TRACE/3000 lists structure points of only those routines that are user-defined, and omits the name of the routine if the routine name was not used as a program unit name in a $TRACE record. MPE/3000 system intrinsics and other program units not solely defined by the user do not have their structure points listed by TRACE/3000.

For example, if a program unit named MAINPROG calls a routine named SUBPROG, and both program units are named in a $TRACE record and in a PRINT/HALT sentence containing a $FORM clause, TRACE/3000 prints the structure points as follows:

```
ENTER MAINPROG
CALL SUBPROG
ENTER SUBPROG
EXIT SUBPROG
RETURN SUBPROG
EXIT MAINPROGG
```

If SUBPROG was not mentioned in a PRINT/HALT sentence containing the $FORM clause, then TRACE/3000 prints the structure points as follows:

```
ENTER MAINPROG
CALL SUBPROG
RETURN SUBPROG
EXIT MAINPROG
```

Structure points are always printed to the left of all other TRACE/3000 printout to readily distinguish structure points from other program elements.

DROP Paragraph

The DROP paragraph deletes PRINT and HALT sentences from the Print/Halt Table. Unlike PRINT and HALT paragraphs, DROP paragraphs are entered by the user through the Interactive File only. The Interactive File is opened and accessed only if the user program executes in interactive mode (MPE/3000 session). The DROP paragraph acts upon 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 within one program unit.

To delete the entire Print/Halt Table, the user enters the DROPALL form of the DROP paragraph. In the examples which follow, TRACE/3000 output is underlined.

```
*DROPALL
```

To delete all sentences pertaining to a specific program unit, the user types (for example):

```
*DROP MAIN'
$ALL
```

To delete sentences pertaining to specific identifiers within a program unit the user types:

```
*DROP  PROG1
SAM
DIX
BOB
```

SAM, DIX and BOB are all identifiers within the program unit specified by *program unit name*. The PRINT or HALT sentences containing those identifiers and associated with the program unit mentioned in the DROP paragraph heading are deleted from the Print/Halt Table.

CHECK Paragraph

The CHECK paragraph confirms correct application of program unit name and identifier name abbreviations as they appear in paragraphs (see Appendix B, "Abbreviations in TRACE/3000 Paragraphs"). CHECK paragraphs are entered by the user through the Interactive File only. To start the CHECK paragraph, the user types:

**CHECK program unit name*

After the user types the paragraph heading optionally followed by a *program unit name*, TRACE/3000 responds with the complete spelling of *program unit name*, thus allowing the user to verify any abbreviation of *program unit name* that he might use. For example,

```
*CHECK L
LARGEPROG
```

If *program unit name* is omitted from the CHECK paragraph, TRACE/3000 assigns *program unit name* from the paragraph immediately preceding the current paragraph in the Interactive File. For example,

```
*CHECK
LARGEPROG
```

In the above example, LARGEPROG was the last used program unit name in the Interactive File, and is now assigned to the current paragraph by TRACE/3000.

Once the user types in the CHECK sentence and TRACE/3000 responds with the full spelling of the *program unit name* assigned to the CHECK paragraph, the user can type identifier name abbreviations for identifiers pertaining to the paragraph *program unit name.* First, the user types the identifier abbreviation immediately followed by an equal sign (=). TRACE/3000 responds by typing the full spelling of the abbreviated word. For example,

```
*CHECK
LARGEPROG
S=SAM
J=JIM
V=VALUE
```

In the above example, SAM, JIM, and VALUE are all identifiers. The user closes the paragraph by typing *return* in column one of the next line. The user can enter as many sentences as he wishes. If the abbreviation typed by the user is not the abbreivation for any identifier within the program unit assigned to the paragraph, TRACE/3000 types the BAD SYNTAX message and closes the paragraph. For example,

```
*CHECK F
FIXIT
S=SAM
Q=
BAD SYNTAX 1
*
```

For further information on abbreviations, see Appendix B, "Abbreviations in TRACE/3000 Paragraphs."

SET Paragraph

The SET paragraph performs two main functions for the TRACE/3000 user:

1. Examines and changes the values of *terms* within the user program units monitored by TRACE/3000.
2. Reports the relative addresses of *terms* according to their position within the user stack area (see *HP 3000 Multiprogramming Executive Operating System*, 03000-90005).

TRACE/3000 recognizes two classes of terms: *variables* and *elements*.

variable is either[1]

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

   BOB
   DIX
   VALUE

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

   ARRAY1
   SORT
   MAXPOINT

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

   @ARRAY1
   @SORT
   @MAXPOINT

element is either

1) An array name or pointer name with subscript. For example,

   SORT(1)
   MAX(44)
   ARRAY1(9)

[1] The FORTRAN/3000 compiler does not insert calls to TRACE/3000 into the user program object code after instructions to read values for array elements. Consequently, the user does not know that array elements have changed value unless the array element was changed by assignment.

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 integer*

Q ± *unsigned octal integer*

S - *unsigned octal integer*

S, Q, and DB stand for stack registers which indicate the start of areas within the user stack. Stack elements must always have an increment, e.g., DB+Q, Q-10. TRACE/3000 treats such *terms* as Q or DB as variables of that name rather than stack elements. All terms used in SET paragraphs must be within the bounds of the user's current stack, meaning that S+*unsigned octal integer* is illegal. (TRACE/3000 confirms that all referenced *terms* are between S and DL.)

A SET paragraph can be entered into the Interactive File at any time the file is accessed. If a SET paragraph is entered into the Interactive File when the file is initially accessed before user program execution, only global or common variables, or DB relative stack elements may be used. If the Interactive File is accessed after the user 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.

Term usage in SET paragraphs further depends whether TRACE/3000 is operated in Normal or Restricted mode (see Section III, "Using TRACE/3000"). 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 any mode. In Normal mode, TRACE/3000 modifies terms of all kinds subject to the terms definition as global, common or program local.

SET paragraphs are entered by the user into the Interactive File only. To enter a SET paragraph, the user types: SET starting in column 2 of the line, followed by an optional program unit name which must be either the main program unit or the currently executing program unit. If the name is omitted, TRACE/3000 assigns the program unit name from the immediately preceding paragraph in the Interactive File. If the SET paragraph is the first paragraph entered into the Interactive File, it must contain a program unit name or TRACE/3000 prints a BAD SYNTAX message and closes the paragraph. In the following examples, items output by TRACE/3000 are underlined. *return* stands for carriage return. To open the paragraph, the user types for example,

```
*SET MAIN'
```

The user can now enter as many SET sentences as desired. To display the value of a term, the user types the term name, followed immediately by typing an equal sign (=). TRACE/3000 then responds by typing the current value of the term. For example,

```
*SET MAIN'
VALUE=36
```

Once TRACE/3000 prints the current value of a term, the user can modify the value of the term by typing a slash and the new value of the term. For example,

```
*SET MAIN'
VALUE=36/34
VALUE=34
```

The user can change the value of a term repeatedly within the same sentence. However, entries are limited to the current line. The user is not allowed to continue entries on the next line. When the user is through typing changes he presses *RETURN*. An example of repeated changes of a term's value is

```
*SET MAIN'
DB-6=44/45/46/47/100/88
DB-6=88
```

Block Listing of Elements

To list a block of related elements (such as several elements of the same array), the user types a comma after the element followed by an unsigned positive decimal integer indicating how many elements TRACE/3000 is to list. Variables (i.e., terms without subscripts or indexes) cannot be listed in block form, since their definition implies only one element in its group. TRACE/3000 prints the elements line by line. The number of elements per line depends upon the element type. In the case of string data, TRACE/3000 types three spaces between elements. If a string element cannot be printed on one line, TRACE/3000 continues it on the next line. An example of array block listing is

```
*SET PROG1
ARRAY(0),10=       -10      1000      55      778      932
                   20         30      40       50       60
```

Block data listing can also be used for stack elements, for example

```
*SET
DB+0,6=     %000001     %000002     %000003
            %000004     %000005     %000006
```

String Data in SET Paragraphs

TRACE/3000 prints string data with quote marks (") around it. A quote mark within a string is listed twice to distinguish it from the quote mark at the beginning and end of the string value. For example, the string value ABC"D is printed by TRACE/3000 as

```
"ABC""D"
```

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

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

When the user changes the value of a string variable, care must be taken to replace the initial value with a string value of equal or lesser length. An attempt to replace a string value with a string value of greater length results in a BAD SYNTAX message and the closing of the SET paragraph. For example,

```
*SET
CHAR="ABCD"/"DEFGH"
BAD SYNTAX 12
*
```

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

```
*SET
CHAR="ABCD"/"XY"
CHAR="XYCD"
```

Using SET for Term Address

The user can enter SET paragraphs into the Interactive File to discover the stack address of terms. First, the user opens the SET paragraph in the normal way. For example,

```
*SET PROGRAM1
```

Next, the user types a hatch mark (#) in column 1 of the line, immediately followed by the term and an equal sign (=). TRACE/3000 responds with the address of the term and a return and linefeed. Unlike SET paragraphs which allow the user to change the value of a term, this form of SET does not allow the user 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 upon the registers used by the MPE/3000 operating system. If the term is a globally defined variable in the user program, TRACE/3000 uses the DB register. If the term is a local variable in a user routine, TRACE/3000 prints the address in terms of the Q or S register. Some examples of SET paragraphs used for addressing are shown below. TRACE/3000 output is underlined.

```
*SET MAIN'
#VALUE= Q+6
#MAX(7)= Q+5,I,X
```

If the term entered by the user is a stack element such as Q+4,S–10, TRACE/3000 responds with the stack address of the term relative to the DB register. For example,

```
*SET
#MINIMUM= Q+4
Q+4= DB+200
```

GO COMMAND

The GO command de-activates the Interactive File and starts execution of the user program. The word GO starts in column two of the line (following the asterisk printed by TRACE/3000) and is optionally followed by a label, separated from GO by one or more blanks. For example, the user might type

```
*GO
```

or

```
*GO   LABEL9
```

Typing the GO command starts execution in the currently executing program unit. If GO is typed before the user program unit has started execution for the first time, execution starts in the user main program unit. If the GO command includes a label, execution resumes in the program unit at that label. (GO *label* is not allowed if TRACE/3000 is operating in restricted mode. See Section III, "Using TRACE/3000.") If no label is included in the GO command, execution resumes at the point in the program unit where execution left off. If no label is included in the initial GO command, execution starts at the main entry point of the main program unit.

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

BAD SYNTAX ERROR MESSAGES

Whenever TRACE/3000 discovers a paragraph with improper form, TRACE/3000 prints 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 offending sentence and either aborts the program (if the ABORT parameter in the $TRACESTART record is used) or reads the next paragraph (if the ABORT parameter is omitted).

If TRACE/3000 detects an error in an Interactive File paragraph, TRACE/3000 closes the paragraph and prompts for a new paragraph by typing an asterisk in column one of the next line. Any line prior to the error line is accepted by TRACE/3000.

SECTION III
Using TRACE/3000

To use TRACE/3000, follow the steps outlined below (Figure 3-1 uses cards as the input medium):

STEP 1: PREPARE THE SOURCE FILE FOR COMPILATION

The user source file consists of four main parts — $TRACE records, user source program statements, Batch File (consisting of PRINT and HALT paragraphs), and data cards (if any). Figure 3-1 shows a complete source file (on cards) ready for loading, compilation, and execution. An :EOD card follows the last user source program unit to signify to the compiler that no more program units follow. $TRACE records are inserted into the source file in one of two ways. Either place the $TRACE records for each program unit immediately in front of the program unit to which the $TRACE records apply, or group all of the $TRACE records together and place them in front of the first program unit.

The Batch File follows the last user program unit. The Batch File is opened by a $TRACESTART card, which may also contain two optional parameters, SIZE and ABORT (see "Batch File" in section II). The Batch File is closed by a $TRACEEND card. PRINT and HALT paragraphs are placed between the TRACESTART and TRACEEND cards. The user can insert as many PRINT and HALT paragraph sentences in the file as desired, subject to the Print/Halt Table size (see "Batch File" in section II). No PRINT or HALT paragraphs need be entered through the Batch File, but the $TRACESTART and $TRACEEND cards must still be present. If a $TRACESTART or $TRACEEND card is missing or misspelled, TRACE/3000 prints out BAD TRACE FILE and terminate the user program. In Batch mode, the user program terminates and is flushed from the system; in Interactive mode, execution passes to TRACE/3000 which opens the Interactive File for the first time (see Section III, "Using TRACE/3000").

STEP 2: COMPILING THE SOURCE PROGRAM

Once the source file is completely assembled, load the entire source file and invoke the appropriate source language compiler using MPE/3000 System Commands (see *HP 3000 Multiprogramming Executive Operating System, 03000-90005*). The operating system reads the entire source file and stores it in memory.

The compiler reads the $TRACE and source program cards and generates object code with the appropriate calling sequences to the TRACE/3000 subprogram. (The :EOD card delimits the end of the source program for the compiler.)

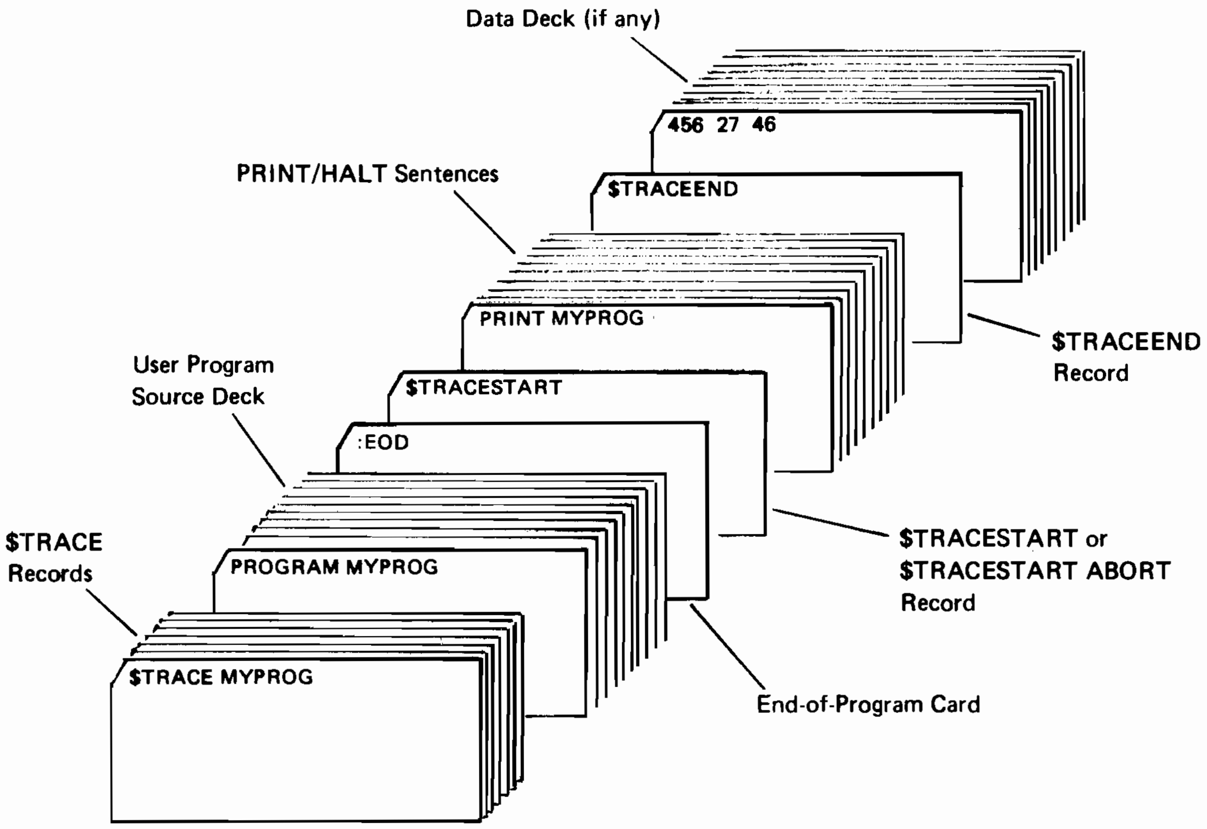

Figure 3-1. TRACE/3000 Source File

STEP 3: LOADING OBJECT CODE

After compilation, the object code generated by the compiler is loaded into memory (in response to the appropriate MPE/3000 system command), and control passes to the TRACE/3000 program. TRACE/3000 reads the Batch File stored in the computer memory. If the Batch File $TRACESTART record does not include the ABORT parameter, TRACE/3000 ignores any PRINT or HALT paragraph sentences of improper form and prints a BAD SYNTAX message. If the ABORT parameter is included in the $TRACESTART record, then TRACE/3000 terminates the entire job when an improper PRINT or HALT paragraph sentence is discovered. In either case, TRACE/3000 prints the offending sentence along with the BAD SYNTAX message.

The Print/Halt Table length is fixed at 200 computer words unless the user specifies otherwise by using the SIZE parameter in the $TRACESTART record at the beginning of the Batch File. The number of PRINT or HALT paragraph sentences that can be entered depends upon the complexity of the sentences. (Consult Table 2-1 "Calculating PRINT/HALT Sentence Size.") If the Print/Halt Table overflows, TRACE/3000 prints OVERFLOW and ignores the sentence that caused the overflow and any following sentences (the offending sentence is printed out). TRACE/3000 executes using the diminished Print/Halt Table.

Computer Museum

STEP 4A: PROGRAM EXECUTION IN BATCH MODE

In batch mode operation, control passes to the object program and the user program starts execution. Every time an identifier mentioned in a $TRACE record is properly referenced in the object program, TRACE/3000 searches the PRINT/HALT table. If a PRINT statement is found to be true, TRACE/3000 prints out information about the identifier. If a HALT sentence is true, the program halts after identifier information is printed and the job is completely terminated. When TRACE/3000 terminates execution, it prints BYE TRACE.

STEP 4B: PROGRAM EXECUTION IN INTERACTIVE MODE

In interactive mode operation, control passes to TRACE/3000, which then opens the Interactive File.

TRACE/3000 asks for the TRACE/3000 mode of operation by typing

```
MODE=
```

The user has the option of running TRACE/3000 under normal or restricted mode. Normal mode gives the user full use of all the TRACE/3000 facilities, while Restricted mode places two constraints on TRACE/3000 use. These constraints are:

1) The user cannot use a label when typing a GO command.

2) The user can modify the value of variables only when entering SET paragraphs into the Interactive File. (This means that elements such as DB+6 cannot be modified, although they can be listed.)

To operate in Normal mode, the user types N; to operate in Restricted mode, the user types R. Either answer is followed by a carriage return.

TRACE/3000 then types the question

```
BATCH FILE =
```

If the user has no Batch File, he presses the return key. Otherwise, he types the name of the file containing the Batch File (followed by pressing return).

TRACE/3000 is now ready to accept paragraphs and types an asterisk in column 1 of the line on the keyboard device. The user can enter as many PRINT, HALT, CHECK, DROP, or SET paragraphs as desired. To start execution of the user program units, the user types a GO command. Program execution begins.

Whenever TRACE/3000 encounters a true HALT sentence, control passes to the user keyboard device after TRACE/3000 activates the Interactive File. The user can then type in paragraphs. To regain control during user program execution, the user presses the TRACE key. TRACE/3000 activates the Interactive File and returns control to the keyboard device.

When TRACE/3000 terminates execution it prints

```
BYE TRACE
```

Table 3-1. Calculating PRINT/HALT Sentence Size

PRINT and HALT sentences are entered into the PRINT/HALT Table by TRACE/3000. The length of each entry depends on the condition clauses in the sentence that builds the entry.

The base word length of any PRINT/HALT paragraph sentence is:	3
Condition clause C1 adds:	1
Condition clause C3 adds:	2
Condition clause C4 adds:	2
Condition clause C2 adds:	1 if the clause contains a variable, or adds as many words as it takes to store the constant if the clause contains a constant. (String variables add one word for every two characters in the string plus one additional half-word (byte) containing the string length.

APPENDIX A
Equivalent Constant Types in Paragraphs

When the user is entering PRINT, HALT, or SET paragraphs, TRACE/3000 allows some constants of different type to substitute for constants of an expected type. For example,

```
*SET MAIN*
SAM= 200/300.25
SAM= 300
```

Computer Museum

In the above example, SAM has an integer value, but the user changed the integer value by entering a real constant, 300.25. TRACE/3000 accepts a real constant in place of an 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 paragraph 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 input 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 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)

APPENDIX B
Abbreviations in TRACE 3000 Paragraphs

ABBREVIATIONS IN TRACE/3000 PARAGRAPHS

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

After compilation of the user source program, the compiler passes a list of all program unit names and identifier names mentioned in the $TRACE cards 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, and so on. Whenever TRACE/3000 encounters a name in a PRINT/HALT sentence or interactive command, it searches the name dictionary. The name chosen is the first full name encountered that is equal to the abbreviation, or that contains the abbreviation as the first letters of its full name.

EXAMPLES:

1. The following $TRACE records are read during the compilation of a source program:

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

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

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

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 TRACE/3000 encounters which uses the abbrevation, M, as the first letter of the full name) and that I stands for IVAR1. TRACE/3000 chooses IVAR1 over IVAR2 or IVAR3 because IVAR1 appears before the two latter names in the name dictionary. TRACE/3000 assumes that the PRINT paragraph means:

```
PRINT MYPROG
IVAR1=6
```

Using the same name dictionary as in example 1, TRACE/3000 responds as follows (TRACE/3000 output is underlined);

```
*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 named B in the name dictionary shown above.