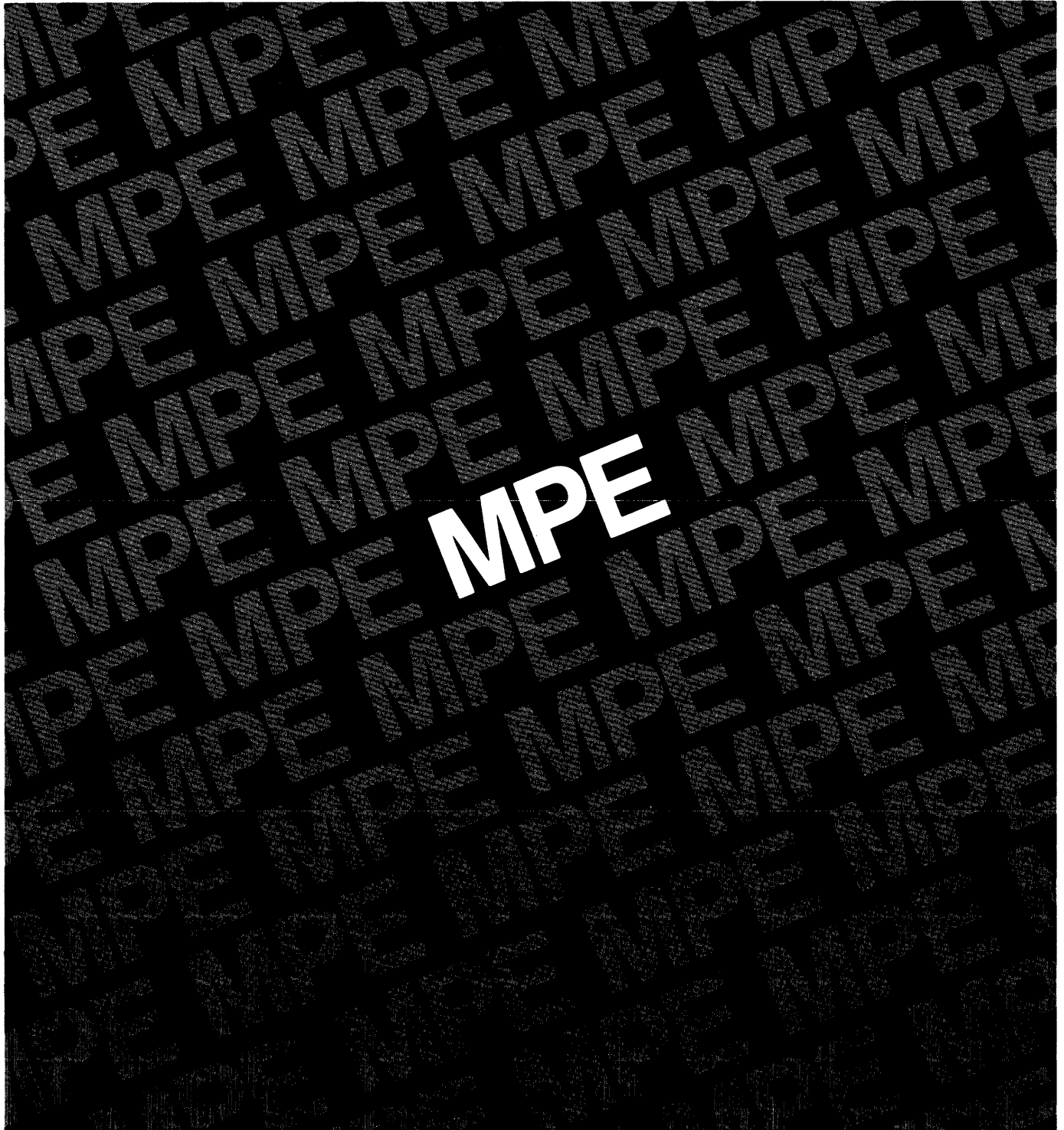


MPE Debug/Stack Dump reference manual



HP 3000 Computer Systems

MPE Debug/Stack Dump Reference Manual



5303 STEVENS CREEK BLVD., SANTA CLARA, CALIFORNIA 95050

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.

LIST OF EFFECTIVE PAGES

The List of Effective Pages gives the date of the current edition and of any pages changed in updates to that edition. Within the manual, any page changed since the last edition is indicated by printing the date the changes were made on the bottom of the page. Changes are marked with a vertical bar in the margin. If an update is incorporated when an edition is reprinted, these bars are removed but the dates remain. No information is incorporated into a reprinting unless it appears as a prior update.

Pages	Effective Date
Title	Jun 1976
ii	Jun 1977
iii to iv	Jun 1977
v to x	Jun 1976
1-1 to 1-3	Jun 1976
2-1 to 2-2	Jun 1976
2-3	Sep 1976
2-4 to 2-15	Jun 1976
2-16	Sep 1976
2-17 to 2-31	Jun 1976
2-32	Sep 1976
2-33	Jun 1976
2-34 to 2-35	Sep 1976
2-36	Jun 1976
3-1 to 3-3	Jun 1976
3-4 to 3-5	Sep 1976
3-6 to 3-26	Jun 1976
3-27	Sep 1976
3-28 to 3-31	Jun 1976
4-1 to 4-2	Jun 1977
4-3	Sep 1976
4-4 to 4-11	Jun 1976
5-1 to 5-2	Jun 1976
A-1 to A-4	Jun 1976
I-1 to I-4	Jun 1976

PRINTING HISTORY

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The date on the title page and back cover of the manual changes only when a new edition is published. When an edition is reprinted, all the prior updates to the edition are incorporated. No information is incorporated into a reprinting unless it appears as a prior update. The edition does not change.

The software product part number printed alongside the date indicates the version and update level of the software product at the time the manual edition or update was issued. Many product updates and fixes do not require manual changes, and conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one to one correspondence between product updates and manual updates.

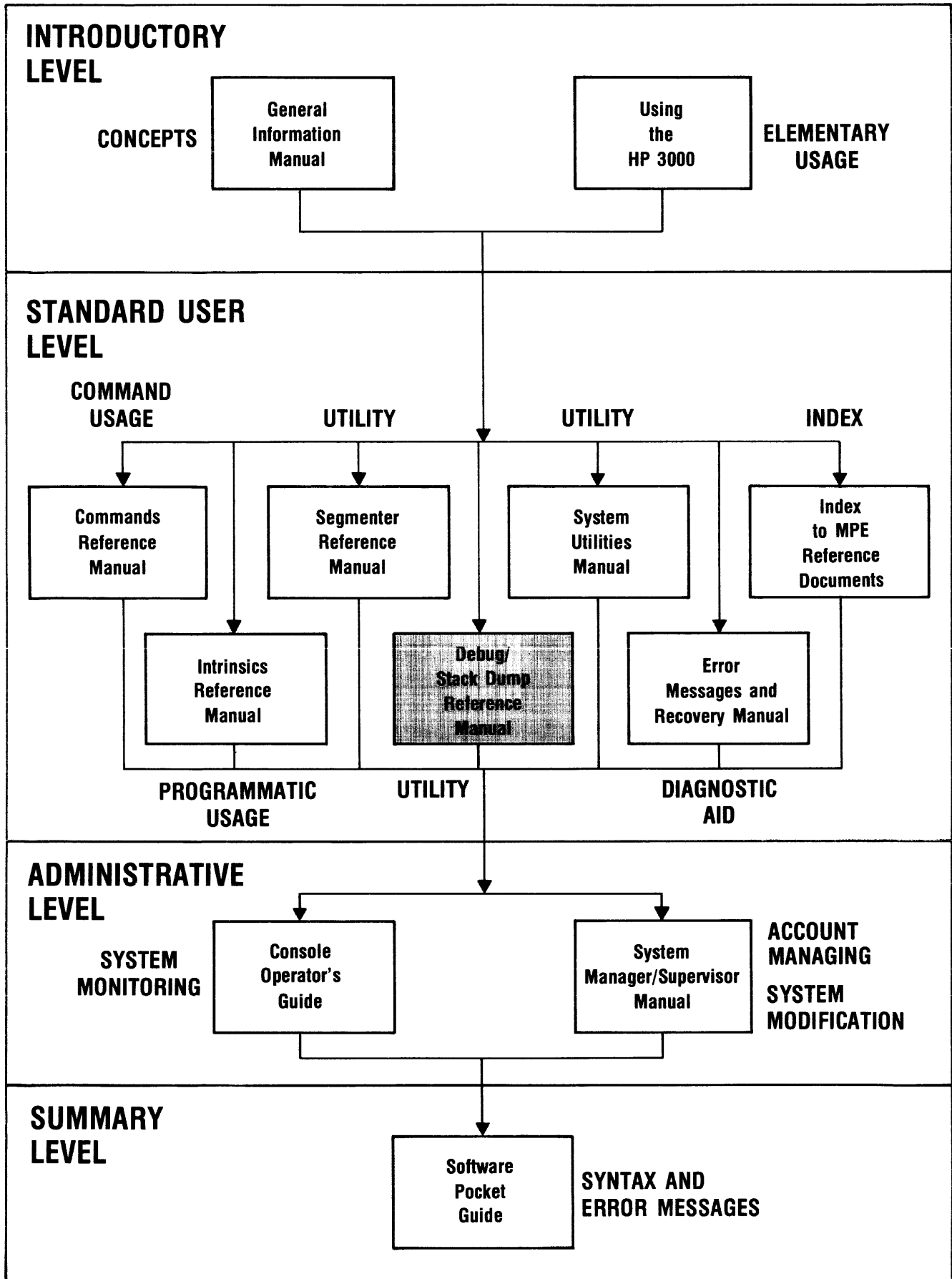
First Edition	Jun 1976
Second Edition	Sep 1976
Update No. 1	Jun 1977
Update Incorporated	Oct 1978

This manual describes two facilities that can be used to locate and correct errors in programs run under control of the MPE operating system. The two facilities are DEBUG and Stack Dump. The manual is part of a set of manuals that describe operation of the Multiprogramming Executive Operating System (MPE) for the HP 3000 Series II computer system. The manual plan on the next page illustrates the relation between this manual (shaded box) and others in the set.

This manual is organized as follows:

- Section I Contains an introduction to the DEBUG and Stack Dump facilities.
- Section II Describes the syntax of the DEBUG commands and the MPE commands and intrinsics used by Stack Dump. The DEBUG commands are listed in alphabetic order followed by the Stack Dump commands and intrinsics; each command and intrinsic is described formally in this section.
- Section III Describes how to use the DEBUG facility, first for users with standard capability followed by an expanded description for users with privileged mode capability.
- Section IV Describes how to use the Stack Dump facility, first in case of abnormal program termination, and then unconditionally.
- Section V Describes the DEBUG error messages and the action to be taken when each message is received.
- Appendix A Illustrates the output from program DECOMP, a useful tool for DEBUG users.

MANUAL PLAN



CONVENTIONS USED IN THIS MANUAL

NOTATION	DESCRIPTION
[]	An element inside brackets is <i>optional</i> . Several elements stacked inside a pair of brackets means the user may select any one or none of these elements. Example: $\begin{bmatrix} A \\ B \end{bmatrix}$ user may select A or B or neither
{ }	When several elements are stacked within braces the user must select one of these elements. Example: $\left\{ \begin{matrix} A \\ B \\ C \end{matrix} \right\}$ user must select A or B or C.
italics	Lowercase italics denote a parameter which must be replaced by a user-supplied variable. Example: CALL <i>name</i> <i>name</i> one to 15 alphanumeric characters.
underlining	Dialogue: Where it is necessary to distinguish user input from computer output, the input is underlined. Example: NEW NAME? <u>ALPHA1</u>
superscript C	Control characters are indicated by a superscript C Example: Y ^C
<i>return</i>	<i>return</i> in italics indicates a carriage return
<i>linefeed</i>	<i>linefeed</i> in italics indicates a linefeed
...	A horizontal ellipsis indicates that a previous bracketed element may be repeated, or that elements have been omitted.

CONTENTS

Section I	Page
INTRODUCTION TO DEBUG AND STACKDUMP	
DEBUG	1-1
Stack Dump Facility	1-2

Section II	Page
COMMAND AND INTRINSIC SPECIFICATIONS	
Commands	2-1
DEBUG Commands	2-2
MPE Commands	2-2
Intrinsics	2-3
DEBUG Specifications	2-4
A Command	2-5
B Command	2-6
C Command	2-9
D Command	2-11
DR Command	2-13
DV Command	2-15
E Command	2-16
F Command	2-17
L Command	2-18
M Command	2-19
MR Command	2-21
R Command	2-23
T Command	2-25
U Command	2-26
= Command	2-27
\$ Command	2-28
DEBUG Intrinsic	2-29
Stack Dump Specifications	2-30
RESETDUMP Intrinsic	2-31
:RESETDUMP Command	2-32
SETDUMP Intrinsic	2-33
:SETDUMP Command	2-34
STACKDUMP INTRINSIC	2-35

Section III	Page
HOW TO USE DEBUG	
Preparing to Use DEBUG	3-1
Using DEBUG With Standard Capability	3-5
What is a Breakpoint?	3-6
How to Establish Breakpoints	3-6
Repeated Breakpoints	3-7
Displaying Breakpoints	3-7

Clearing Breakpoints	3-8
Resuming Execution	3-9
Switching Display to Line Printer	3-9
How to Display and Modify Values	3-11
Displaying Values	3-12
Modifying Values	3-16
Displaying and Modifying Registers	3-16
Displaying Code	3-18
Displaying Stack Markers	3-20
How to Use the DEBUG Intrinsic	3-21
Running With the DEBUG Intrinsic	3-21
Using DEBUG With Privileged Mode Capability	3-26
Setting Breakpoints in Privileged Mode	3-28
Switching Display to Line Printer in Privileged Mode	3-28
Displaying Code, Data, and Disc Segments in Privileged Mode	3-29
Modifying Data in Privileged Mode	3-29
Display or Modify Registers in Privileged Mode	3-30
Display Registers	3-30
Modify Registers	3-30
Freezing/Unfreezing Segments in Privileged Mode	3-30

Section IV	Page
HOW TO USE THE STACK DUMP FACILITY	
Stack Trace and Analysis	4-1
Using SETDUMP and RESETDUMP	4-2
Interactive Use of SETDUMP	4-3
Using :SETDUMP Command Interactively	4-3
Using SETDUMP Intrinsic Interactively	4-5
Using SETDUMP in Batch Mode	4-6
Using :SETDUMP Command in a Job	4-6
Using SETDUMP Intrinsic in a Job	4-6
Terminating With RESETDUMP	4-7
Using STACKDUMP	4-8
How to Call STACKDUMP	4-8
Analyzing the Stack Dump	4-10

Section V	Page
DEBUG ERROR MESSAGES	5-1

Appendix A	Page
USING DECOMP	A-1

ILLUSTRATIONS

Title	Page	Title	Page
Stack Dump Modes and <i>selec</i> Array Format	2-36	PMAP Listing for FORTRAN Program With	
FORTRAN Compilation Showing Symbol and		DEBUG Intrinsic	3-24
Label Maps	3-2	DEBUG Example From Program With	
Result of Preparation With PMAP	3-4	DEBUG Intrinsic	3-25
DEBUG Example	3-10	Sample Stack Trace and Analysis	4-1
Layout of Items in Data Stack	3-15	Sample FORTRAN Program With Error	4-4
Display Code Locations Example	3-19	Sample FORTRAN Program	
Sample FORTRAN Program With Call		Using STACKDUMP	4-9
to DEBUG	3-22	Sample Stack Dump	4-11
		DECOMP Listing	A-2

TABLES

Title	Page	Title	Page
DEBUG Command Summary	2-4	Privileged Mode Command Capabilities	3-27
Stack Dump Facility Summary	2-30	DEBUG Error Messages	5-2

INTRODUCTION TO DEBUG AND STACKDUMP

SECTION

I

DEBUG

DEBUG is an intrinsic procedure which provides both non-privileged and privileged users with an interactive debugging facility to enable checkout of their operating environments.

Non-privileged users are bounded by software protection checks to their private code segment domains and (stack) data space. For privileged users the only bounds checking performed is that for stack overflow (where the S register contains an address greater than that in the Z register).

WARNING

The normal checks and limitations that apply to the standard users in MPE are bypassed in privileged mode. It is possible for a privileged mode program to destroy system integrity, including the MPE operating system software itself. Hewlett-Packard cannot be responsible for system integrity when programs written by users operate in privileged mode.

With DEBUG, it is possible to:

- Establish one or more breakpoints in a program. The program will execute until a breakpoint is reached, then stop and pass control to the user.
- Display and/or modify the contents of memory locations relative to the data stack bases DB, DL, Q, and S.
- Display the contents of memory locations relative to the code bases PB, P, or PL.
- Display and/or modify the contents of registers.
- Trace and display stack markers.
- Calculate the value of expressions in order to determine the correct values for variables at a given point in a program.
- Redirect displays to a list device.
- In addition, privileged users can:

Display and/or modify absolute data segments

Display code segments and disc sectors

Freeze (and subsequently unfreeze) code or data segments in memory.

DEBUG can be invoked by a direct external call to the DEBUG intrinsic, or by specifying the DEBUG parameter in the :PREPRUN or :RUN commands or the CREATE intrinsic.

NOTE

See the *MPE Commands Reference Manual* for a discussion of the :PREPRUN and :RUN commands and the *MPE Intrinsics Reference Manual* for a discussion of the CREATE intrinsic.

Once the DEBUG facility is accessed, control is passed to the user at the interactive terminal so the user can establish breakpoints in his program. The program executes until a breakpoint is reached, then DEBUG is called again and control is again passed to the user.

It is important to note that when DEBUG is called, the scope of access is determined solely by the capability (non-privileged or privileged) of the *user* and not the calling program.

Upon entry to DEBUG, validity checks ensure that the user is in an interactive session and has read/write access to the program file. If not, control is returned immediately to the user. Hence, a DEBUG intrinsic call in a program running in batch mode is essentially regarded as a null statement and DEBUG is not executed.

Four temporary registers, known as registers 1, 2, 3, and 4, are provided by DEBUG for the convenience of the user. Each register is set to zero upon entry to DEBUG. These registers can be displayed or modified in the same way as normal system registers and thus may be used to store values that are used repeatedly in debugging a program.

See Section III for a description of how to use DEBUG.

STACK DUMP FACILITY

The stack dump facility is composed of two complementary, independent features:

- A callable intrinsic (STACKDUMP) that enables any program to selectively dump any part of the stack to the standard list device or other file. This feature is a debugging aid inasmuch as it is a simple method to dynamically monitor any or all variables of a program.
- A mechanism (abort stack analysis) that can be enabled or disabled by commands or intrinsics. This facility causes the following special actions when the program aborts:

For a batch job, parts or all of the stack are dumped on the standard list device.

For an interactive session, an automatic call to DEBUG is generated.

The abort stack analysis mechanism can be enabled, in a batch job or an interactive session, in three ways:

1. With the `:SETDUMP` command. (The `:RESETDUMP` command disables the mechanism.)
2. With the `SETDUMP` intrinsic. (The `RESETDUMP` intrinsic disables the mechanism.)
3. By specifying the `DEBUG` parameter of the `CREATE` intrinsic.

The stack dump facility provides the following features:

- Makes it simple for a user to obtain information necessary to determine the cause of an abort.
- Allows the information to be formatted in a way that is clear to read.
- Provides a means of limiting the amount of output in those cases where the entire stack does not need to be dumped.

See Section IV for a description of how to use the stack dump facility.

COMMAND AND INTRINSIC SPECIFICATIONS

SECTION

II

Specifications for the DEBUG commands are presented in this section in alphabetical order. The DEBUG commands are followed by specification of the DEBUG intrinsic that can be used to request the DEBUG facility when it is not requested by the DEBUG parameter in the RUN or PREPRUN commands.

The STACKDUMP facility uses a set of intrinsics to request a stack dump during normal program execution or only when the program terminates abnormally. The abort stack dump can also be requested or disabled with a pair of MPE commands that parallel the stackdump intrinsics. The specification of these commands and intrinsics are presented in alphabetic order following the DEBUG commands.

COMMANDS

The command specifications contain the following information:

- The command name.
- The type of command (DEBUG command or MPE command). This information is shown in italics directly under the command name. If the command can be used only in privileged mode, this information is shown also.
- A brief summary of the command's function.
- Syntax. The command syntax is highlighted by being shown in a shaded box.
- Parameter definitions, including meaning, constraints, and defaults.
- Examples.
- Text discussion. (The page in this manual where usage of the command is discussed.)
- Privileged mode capabilities, if any, for the various commands are enclosed in a box, as for example,

Privileged Mode

All ST bits can be changed

DEBUG COMMANDS

DEBUG commands consist of the following elements:

- A question mark prompt, displayed by DEBUG.
- The command name, consisting of a one- or two-letter identifier. The command name must follow the prompt with no intervening blanks.
- Parameters, if any, follow the command name. Blank characters may be used anywhere after the command name, but are not required. For some commands, however, the first parameter must be preceded by a comma. See the SYNTAX and EXAMPLES parts of the command specifications for details about specifying parameters.
- Parameters that are numeric values (*segment*, *offset*, etc.) can be specified as an expression that, when evaluated, results in a single precision number.

An expression can include any of the operators: +, -, /, or *. The operands in an expression are assumed to be octal numbers unless indicated otherwise as follows:

- # prefix to decimal value; e.g., #10 is decimal value 10
- % prefix to octal value; e.g., 10 or %10 is octal value 10
- " " enclose ASCII characters for which an octal equivalent is indicated; e.g., "A" indicates octal equivalent 101.
- ' ' enclose memory location of which the contents are indicated; e.g., 'DB+ 4' is the contents of location DB+ 4.
- \$ prefix indicating contents of a register; e.g., \$P is the current value of P.
- !(a:b) extract bit field where *a* is the starting bit and *b* is the number of bits; e.g., \$P!(8:8) is bits 8 through 15 of the contents of register P.

- Parameters used as an offset to a memory location base can be followed by a colon (:) to indicate indirect addressing.

MPE COMMANDS

MPE commands consist of the following elements:

- A colon, required in all cases as an MPE command identifier. In an interactive session, MPE displays the colon on the terminal when it is ready to accept a command. In a batch job, you must enter the colon in the first column of the command record.
- The command name, which must follow immediately after the colon. MPE prohibits embedded blanks within the command name, or between the colon and the name. A blank signifies the end of the command name.

- Parameters, if any, follow the command name. You must separate the parameter list from the command name by one or more blanks. When several parameters are used in a list, they must be separated by commas (delimiters). Any delimiter can be surrounded by any number of blanks; however, blanks may not be embedded within parameters. The end of the parameter list is indicated by a carriage return in a session or the end of the record in a job. *Positional* parameters have significance due to their position in the parameter list. For example, in the following instance

```
:COMMAND parameter1, parameter2, parameter3
```

parameter1 must always be specified before *parameter2* or *parameter3*

If an optional positional parameter is omitted from the parameter list, commas are used to denote this as illustrated:

```
:COMMAND parameter1,,parameter3      (From middle of list)
:COMMAND ,parameter2,parameter3      (From beginning of list)
:COMMAND parameter1                   (From end of list. Commas are not required.)
```

INTRINSICS

The intrinsic specifications contain the following information:

- The intrinsic name. The word *intrinsic*, in italics, directly under the intrinsic name identifies it as an intrinsic.
- A brief summary of the function of the intrinsic.
- The complete intrinsic call description, highlighted by being enclosed in a box. The format is:

LV SETDUMP(<i>flags</i>);

Required parameters, such as *flags*, are shown in **bold face italics**.

Optional parameters are shown in *regular italics*. Superscripts (see LV above) are used to denote the types of parameters and whether they must be passed by *value*, instead of by *reference* (the default case). See Section I of the *MPE Intrinsic Reference Manual* for a discussion of passing parameters by value and by reference.

Superscripts have the following meanings:

BA	Byte array
DA	Double array
I	Integer
L	Logical
LV	Logical by value
O-V	Optional variable

In addition to the superscripts shown over the parameters, the superscript O-V is shown following some parameters to denote *option variable*. Option variable means that the intrinsic contains *optional* parameters.

DEBUG SPECIFICATIONS

The specifications for the DEBUG commands are listed in the following pages in alphabetic order. They are followed by the DEBUG intrinsic specification.

A summary of the commands and their purpose is given in table 2-1.

Table 2-1. DEBUG Command Summary

COMMAND	MODE	PURPOSE
A	*	Establish breakpoint mode as private or system.
B		Set breakpoints in executing program.
C		Clear established breakpoints.
D		Display code or data stored in memory.
DR		Display register contents.
DV	*	Display disc sectors.
E		Delete values from top of stack or terminate program.
F	*	Freeze code or data segment in memory.
L		Switch display to list device.
M		Modify data stored in memory.
MR		Modify register contents.
R		Resume program execution and, optionally, set a new breakpoint.
T		Trace and display stack markers.
U	*	Unfreeze frozen code or data.
=		Calculate value of expression.
\$		Modify value of single register.

An asterisk (*) in the Mode column indicates that this command can be used only in privileged mode.

Switches between private and system breakpoint modes.

SYNTAX

AS	<i>Switch to system breakpoint mode.</i>
AP	<i>Switch to private breakpoint mode.</i>

NOTE

System breakpoints are global; any program in the system will break at those breakpoints. In private breakpoint mode, only the program in which the breakpoints are established will break.

TEXT DISCUSSION

Page 3-28.

B

DEBUG command

Establishes one or more breakpoints within a program.

SYNTAX

$$B \left[\begin{matrix} [G] \\ [P] \end{matrix} \right] \textit{segment.} \textit{offset} \left[: \left[@ \right] \left[\textit{count} \right] \right], \dots$$

or

B@

NOTE

B@ causes all breakpoints belonging to the current process to be displayed.

Privileged Mode

To set a breakpoint to debug system segmented library, use prefix S as follows:

B[S *segment.*] *offset* [:[@][*count*]]

To set a breakpoint in absolute code segment (CST), use prefix A as follows:

B[A *segment.*] *offset* [:[@][*count*]]

PARAMETERS

- G or P To establish breakpoints in the group or account segmented library, use the prefix G for group or P for account before the *segment* parameter. If omitted, breakpoints are established in the currently executing program. (Optional parameter.)
- segment* The logical code segment to contain the breakpoint. If omitted, the currently executing segment contains the breakpoint. (Optional parameter.)
- offset* The relative offset of the breakpoint from the start of the segment. (Required parameter.) A register may be specified if preceded by \$.
- :@ Makes breakpoint permanent until program terminates or breakpoint cleared by explicit C command. If omitted, and if *count* is omitted, the breakpoint is cleared (deleted) after first execution. (Optional parameter.)

B 20:@ Break each time location 20 is executed.

:count An expression specifying when the breakpoint is executed. When *count* is reached, the breakpoint is executed and, if @ is not specified, deleted. (Optional parameter.) If omitted, breakpoint is executed the first time it is reached.

B 20:3 Break the third time location 20 is executed.

When @ and count are combined, the breakpoint is conditionally permanent, that is, the breakpoint will break, but not delete after *count* executions.

B 20:@3 Break every third time location 20 is executed.

EXAMPLES

?B 4.55, 20:@, A45.44:20, G1.22:@12, \$P+4

Breakpoints are set at:

Location 55 of program segment 4.

Location 20 of current program segment (permanent — break every time).

Location 44 of absolute code segment 45 (conditional — break and delete after 20th time). (Requires privileged mode.)

Location 22 of group library segment 1 (permanent and conditional — break every 12th time).

Location defined by contents of register P+4.

NOTE

An optional form of the B command — B@, causes all breakpoints belonging to the current process to be displayed.

Privileged Mode

B@ displays all system-owned breakpoints in AS mode; all user-owned breakpoints in AP mode.

The display format for each breakpoint is

$$\text{LCST} = \begin{bmatrix} \text{P} \\ \text{G} \end{bmatrix} \text{ lsn, P = pc, CST = asn, [@] t/u}$$

where:

P = Account Public System Library.

G = Group System Library.

- lsn* = Logical Code Segment Number.
- pc* = Program Counter.
- asn* = Actual Code Segment Index. (Privileged Mode only.)
- @ = Permanent breakpoint. (No @ indicates temporary breakpoint.)
- t* = Total number of executions allowed by conditional breakpoint.
- u* = Total number of times breakpoint actually executed.

TEXT DISCUSSION

Page 3-6; privileged mode 3-28.

Clears one or more breakpoints.

SYNTAX

$C \left[\begin{array}{c} [G] \\ [P] \end{array} \right] \text{segment.} \text{offset}, \dots$ <p>or</p> $C@$
--

NOTE

C@ clears all breakpoints in a program.

Privileged Mode

Includes all the above plus the following:

$$C \left[\begin{array}{c} [S] \\ [A] \end{array} \right] \text{segment.} \text{offset}, \dots$$

where S indicates breakpoints in the system segment library

A indicates breakpoints in absolute code segment (CST).

C@ clears all system-owned breakpoints if operating in AS mode or all user-owned breakpoints if operating in AP mode.

PARAMETERS

- | | |
|----------------|---|
| G or P | Breakpoints to be cleared are in group segmented library (G) or in account segmented library (P). If omitted, breakpoints in currently executing program are cleared. (Optional parameter.) |
| <i>segment</i> | The logical program segment containing the breakpoint. If omitted, the currently executing segment applies. (Optional parameter.) |
| <i>offset</i> | The relative offset of the breakpoint from the start of the segment. (Required parameter unless the form C@ is used.) |

EXAMPLES

?C50, 2.33, P4.77

Breakpoints are cleared at:

Location 50 in current program segment.

Location 33 in program segment 2.

Location 77 in public library segment 4.

TEXT DISCUSSION

Page 3-8.

Displays memory contents of a specified number of locations relative to a given code base or data base.

SYNTAX

D [*dispbase*] [*offset*] [,*count*] [,*mode*]

PARAMETERS

dispbase One of the following stack (data) relative display bases:

DB, DL, Q, S

or one of the following code relative display bases:

PB, P, PL

If *dispbase* is omitted, DB is specified by default. (Optional parameter.)

Privileged Mode

Includes all of the above plus the following:

- A = Absolute Relative (base = location 0).
- SY = System Global Relative (base = system base).
- CO = Code Segment Relative (base = base of segment).
- DA = Data Segment Relative (base = base of segment).
- DX = Current Absolute DB Relative (base = absolute DB).
- EA = Extended Absolute Address (base = bank specified).

The bank number in EA mode follows EA; for example:

D EA2+ 10 Displays one word at location 10 of bank 2.

For CO and DA, the offset immediately follows the mnemonic (CO or DA) unless it is an expression involving a calculation when it is enclosed in parentheses; for example:

D DA22+ 6,6 Displays 6 words starting at location 6 of data segment 22.

D CO(4+ 6),3 Displays first 3 words of segment 12 (octal).

offset The offset relative to the display base which specifies the starting memory location of the area to be displayed. It is written in the following format:

[+] *expression* [: [+] *expression*]

If no sign is given, positive offset is assumed. If *expression* is followed by a colon, indirect addressing is indicated. The indirect addressing is relative to the specified *dispbse* or, if omitted, to DB for stack relative display or PB for code relative display bases.

The first *expression* can be followed by an additional offset *expression*. For example:

D 6:+ 2 Display contents of address found by adding two words to address stored in DB+ 6.

Multiple levels of indirect are permitted. For example:

D 6:+ Q+ 3: Display contents of address found by adding address stored in Q+ 3 to address stored in DB+ 6.

If *offset* is omitted, location 0 is specified by default. (Optional parameter.)

count An expression which defines the number of memory locations to be displayed. If omitted, *count* is assigned a default value of 1. (Optional parameter.)

mode A one-character specifier to indicate the representation mode for output values: O for octal, I for decimal, or A for ASCII. If omitted, default mode is octal. (Optional parameter.)

EXAMPLES

```
?D5
DB+5            047516
?D Q-5
Q-5            177777
?D Q-5:
DB+177777      000020
?D Q+1:,A
DB+0            MA
?D Q+1:,4,A
DB+0            MAINPROG
```

Location DB + 5. DB is assumed when dispbase omitted.

Location Q - 5.

Location pointed to by Q - 5.

One word in ASCII at location pointed to by Q+1.

Four ASCII words at location pointed to by Q+1.

TEXT DISCUSSION

Page 3-11; privileged mode 3-29.

Displays contents of the X, ST, DL, Q, S, Z, P, 1, 2, 3, and 4 registers.

SYNTAX

`DR [,register] [,register] , . . .`

PARAMETERS

register The register whose contents are to be displayed. This can be ST, X, DL, Q, S, Z, P, or temporary registers 1, 2, 3, and 4. If omitted, all these registers plus LCST (the index to the logical code segment) are displayed. (Optional parameter.)

NOTE

If LCST is displayed and the breakpoint is set in a group segmented library, the segment number is preceded by G; if set in a public library, it is preceded by P; if in the current user program, the segment number has a blank prefix.

Privileged Mode

Privileged mode, if displaying all registers (*register* parameter omitted), also includes the following values:

PCB = Process Control Block Index.
CST = Absolute Code Segment Index.
STAK = Stack Segment Index.
DST = Extra Data Segment Index.
DX = Current value of DB register, if in absolute mode.
EA = Current bank number, if in absolute mode.

In addition, the segment number displayed as LCST is preceded by S for a system library segment.

EXAMPLES

```
?DR
ST=60301, X=0, DL=177602, Q=26, S=26, Z=1462, P=166
1=0, 2=0, 3=0, 4=0, LCST=0
```

DR with no parameters specified displays all registers.

```
?DR, ST, Z
ST=60301, Z=1462
?DR, 1, 2
1=0, 2=0
?DR, Q
Q=26
```

Display registers ST and Z.

Display temporary registers 1 and 2.

Display register Q.

TEXT DISCUSSION

Page 3-16; privileged mode 3-30.

Displays a requested number of sectors of virtual memory or disc.

SYNTAX

`DV [ldev] + startsector [,count] [,mode]`

PARAMETERS

- ldev* The logical device number of the disc to be displayed. If omitted, the system disc is specified by default. (Optional parameter.)
- startsector* An expression signifying the starting sector address to be displayed. If the sector address requires more than 16 bits, it must be entered in the following form:

 (*high-order bits:low-order bits*) (Required parameter.)
- count* An expression indicating the number of sectors to be displayed. If omitted, one sector is displayed. (Optional parameter.)
- mode* One character to indicate representation mode of output values: O for octal, I for decimal, or A for ASCII. If omitted, octal output is assumed (Optional parameter.)

EXAMPLES

?DV2 + 2230,5

Displays 5 sectors from logical device number 2 starting at sector number 2230.

?DV (012345:3)

Displays 1 sector from the system disc starting at sector number 612345 where the low-order bits are 012345 (octal) and the high-order bits are 3 (octal).

TEXT DISCUSSION

E

DEBUG command

Exits from *DEBUG* and resumes program execution; optionally, deletes values from top of stack. Can also be used to terminate program.

SYNTAX

<p style="text-align: center;">E [<i>parameternum</i>]</p> <p style="text-align: center;">or</p> <p style="text-align: center;">E@</p>
--

NOTE

E@ terminates the program.

PARAMETERS

parameternum An expression which specifies the number of values to be deleted from the stack counting back from the top-of-stack, S. This expression cannot be greater than 255. If omitted, and if @ is omitted, control returns to the instruction following a direct call to *DEBUG*, or to the instruction which generated a break to *DEBUG*. (Optional parameter.)

EXAMPLES

?E2

Delete 2 values (S-0 and S-1) from the stack and resume program execution.

TEXT DISCUSSION

Page 3-9; 4-5

F

DEBUG command
Privileged mode only

Freezes a code or data segment in main memory so that it is not swapped out.

SYNTAX

$F \left\{ \begin{array}{l} \text{CO} \\ \text{DA} \end{array} \right\} \textit{segnum}$
--

PARAMETERS

CO Indicates code segment. CO or DA must be specified.

DA Indicates data segment. DA or CO must be specified.

segnum An expression indicating the segment number to be frozen. (Required parameter.)

EXAMPLES

?FDA22 Freezes data segment 22.

TEXT DISCUSSION

Page 3-30.

L

DEBUG command

Switches all display output to a file or, if in privileged mode, to a file or a specific logical device, instead of to the interactive standard list device. The file or logical device must be equated to a line printer.

SYNTAX

<p>L [<i>filereference</i>]</p> <p>or</p> <p>L0</p>

NOTE

L0 closes an open file and switches display back to interactive device.

<p>Privileged Mode</p> <p>L [<i>ldev</i>]</p>

PARAMETERS

filereference The name (and optional group and account) of the file to which all output is to be directed. If omitted, the file (assuming one is open) is closed and output again is displayed on the interactive device. Omitting the *filereference* parameter has the same effect as specifying L0. (Optional parameter.)

<p>Privileged Mode</p> <p><i>ldev</i> The logical device number of a device to which the output is to be directed. If omitted, the display is switched back to the interactive device. (Optional parameter.)</p>
--

EXAMPLES

?L PRINTER

Directs output to a file named PRINTER.

?L

Closes file PRINTER and directs further output to interactive device.

TEXT DISCUSSION

Page 3-9; privileged mode 3-28.

Modifies the contents of a specified number of memory locations relative to a given base in the user's stack.

SYNTAX

M [*modbase*] [*offset*] [,*count*] [,*mode*]

PARAMETERS

modbase One of the following stack relative modification bases:

DB, DL, Q, S

If *modbase* is omitted, DB is assumed. (Optional parameter.)

Privileged Mode

Other values allowed for *modbase*:

A	=	Absolute Relative (base = location 0).
SY	=	System Global Relative (base = system base).
DA	=	Data Segment Relative (base = base of segment).
DX	=	Current Absolute DB Relative (base = absolute DB).
EA	=	Extended Absolute Address (base = bank specified).

offset The offset relative to the modify base which specifies the starting memory location of the area to be modified. It is written in the following format:

[+] *expression* [: [[+ *expression*]]

If no sign is given, positive offset is assumed. If *expression* is followed by a colon, indirect addressing is indicated. The indirect addressing is relative to the specified *modbase* or, if omitted, to DB for stack relative or PB for code relative modify bases.

The first *expression* can be followed by an additional offset *expression*. For example:

M 6:+ 2	Modify contents of address found by adding two words to address stored in DB+ 6.
---------	--

Multiple levels of indirect are permitted. For example:

M 6:+ Q+ 3: Modify contents of address found by adding address stored in Q+ 3 to address stored in DB+ 6.

If *offset* is omitted, location 0 is specified by default. (Optional parameter.)

count An expression which defines the number of memory locations to be modified. If omitted, 1 is specified by default. (Optional parameter.)

mode A one-character specifier to indicate the representation mode for output values: O for octal, I for decimal, or A for ASCII. If omitted, default is octal. (Optional parameter.)

The M command causes the current contents of each specified location to be displayed. If mode is not specified, the display is in octal. Following the display, DEBUG issues a :=. You can then enter a new value. This value is octal unless preceded by # (decimal), by \$ (register contents), surrounded by quotes (ASCII), surrounded by apostrophes (location contents). If an invalid value is entered, DEBUG repeats the prompt := until you enter an acceptable value, press *return* to retain the current value, or enter "." to terminate the command.

EXAMPLES

```
?M
DB+0            046501 :=046502
?M5,4
DB+5            047516 :=047505
DB+6            042440 :=041440
DB+7            020040 :=020044
DB+10           046516 :=047516
```

Modify stack location DB + 0 (default location when no parameters are specified).

Modify four locations starting at stack DB + 5.

Privileged Mode

?MSY54	Modify relative location 54 in system global area.
?MDA26+5:,2	Modify two cells in data segment 26 starting at the location pointed to by location 5 in the segment.

TEXT DISCUSSION

Page 3-16; privileged mode 3-29.

Modifies contents of ST, X, DL, Q, S, Z, P, 1, 2, 3, and 4 registers.

SYNTAX

MR [*,register*] [*,register*] , . . .

PARAMETERS

register The register to be modified. Omitting this parameter indicates all registers. New values then are requested, one at a time, in the form:

register = *currentcontents* :=

to which you respond with an expression to specify the desired new contents, terminated by a carriage return.

If the input is invalid, := is repeated so you can enter valid characters. A carriage return alone (null value input) preserves the current contents.

An ASCII period (".") terminates any further requests for modification. (Optional parameter.)

NOTE

The following restrictions apply to the MR command:

1. The register contents must be such that

$$DL \leq 0 \leq Q \leq S \leq Z$$

2. Only bits 2 through 7 of the ST register can be changed.

— Privileged Mode —

All ST bits can be changed.

EXAMPLES

```
?MR
ST=60301:=2
X=0:=444
DL=177602:=500
      :=177601
Q=26:= return
S=26:=27
Z=1462:= return
P=166:=167
1=0:= return
2=0:= return
3=0:=47
4=0:=124
```

MR command with no parameters. (All registers listed.)

Change to 2.

Change to 444.

Illegal entry.

Good value.

No change.

Change to 27.

No change.

Change to 167.

No change.

No change.

Change to 47.

Change to 124.

TEXT DISCUSSION

Page 3-16; privileged mode 3-30.

Resumes program execution and optionally establishes another breakpoint.

SYNTAX

$$R[[segment.] offset [: [@] count]]$$

NOTE

To resume and set a group or account breakpoint, use prefix G for group or P for account before the *segment* parameter, as follows:

$$R[[\left. \begin{array}{c} G \\ P \end{array} \right\} segment.] offset [: [@] count]]$$

Privileged Mode

To resume and set a breakpoint for system segmented library, use prefix S before the *segment* parameter, as follows:

$$R [[S segment.] offset [: [@] count]]$$

PARAMETERS

- | | |
|----------------|---|
| <i>segment</i> | The logical code segment to contain the breakpoint. If omitted, the current segment applies. (Optional parameter.) |
| <i>offset</i> | The relative offset of the breakpoint from the start of the segment. If omitted, no breakpoint is set. (Optional parameter.) |
| :@ | Makes breakpoint permanent until program terminates or breakpoint cleared by explicit C command. If omitted, and if <i>count</i> is omitted, the breakpoint is cleared (deleted) after first execution. (Optional parameter.) |

R 20:@ Resume and then break each time location 20 is executed.

count An expression specifying when the breakpoint is executed. When *count* is reached, the breakpoint is executed and, if @ is not specified, deleted. (Optional parameter.) If omitted, breakpoint is executed the first time it is reached.

R 20:3 Resume and then break the third time 20 is executed.

When @ and *count* are combined, the breakpoint is conditionally permanent, that is, the breakpoint will break, but not delete after *count* executions.

R 20:@3 Resume and then break every third time 20 is executed.

NOTE

If the R command is used with no parameters specified, execution resumes without establishment of a new breakpoint. Control is returned to the instruction following a direct call to DEBUG, or to the instruction which generated a break to DEBUG.

EXAMPLES

?R 110

Resume, set a breakpoint at location 110 of the current segment, and run until this breakpoint is encountered.

TEXT DISCUSSION

Page 3-9; privileged mode 3-28.

Traces and displays stack markers.

SYNTAX

T

————— Privileged Mode —————

In privileged mode, T also displays absolute code segment index (CST).

NOTE

Listing format for each marker is:

$$Q = dq, LCST = \begin{bmatrix} P \\ G \\ S \end{bmatrix} lsn, P = pc, CST = asn$$

where:

- dq* = Displacement from current Q.
- P* = Account public segmented library.
- G* = Group segmented library.
- S* = System segmented library. If P, G, and S are omitted, current program.
- lsn* = Logical CST number.
- pc* = P (relative) address.
- asn* = Absolute (actual) CST (privileged mode only).

EXAMPLES

```
?T
Q-0      ,LCST=0      ,P=266
Q-23     ,LCST=S132 ,P=0
```

TEXT DISCUSSION

U

DEBUG command

Privileged Mode Only

Unfreezes a frozen code or data segment.

SYNTAX

$$U \left\{ \begin{array}{l} CO \\ DA \end{array} \right\} segnum$$

PARAMETERS

CO Indicates code segment. CO or DA must be specified.

DA Indicates data segment. DA or CO must be specified.

segnum An expression indicating the segment number to be unfrozen. (Required parameter.)

EXAMPLES

?U C0123 Unfreezes code segment 123.

TEXT DISCUSSION

Page 3-30.

Calculates expression.

SYNTAX

= *expression* [,*mode*]

PARAMETERS

- expression* The expression to be evaluated. The expression can include octal values (default), decimal values (must be preceded by #), ASCII values (must be enclosed in quotes, as "A"), memory content (location must be enclosed in apostrophes, as 'Q+2'), and register content (must be preceded by \$, as \$X). The types of values may be mixed in an expression. (Required parameter.)
- mode* The representation mode for the result. O specifies octal, I decimal, and A ASCII. If omitted, the result is represented as an octal value. (Optional parameter.)

EXAMPLES

?=4+(5*2-3/(1+1))
=15

Answer is 15 (octal).

?=#4+(55*#12)-"A",I
=+479

Answer is +479 (decimal).

?=\$Q
=41

Answer is current value of Q-register (41 octal)

?='DB+26'
=23420

Answer is current contents of DB+26 (23420 octal)

TEXT DISCUSSION

Page 3-17.

\$

DEBUG command

Modifies single register value.

SYNTAX

\$ register := expression

PARAMETERS

register The register to be modified; may be ST, X, DL, Q, S, Z, P, or temporary registers 1, 2, 3, and 4. (Required parameter.)

:=expression An expression signifying the new value. The expression must be preceded by :=. Unless specified otherwise, as described for the = command, all values in the expression are considered octal. (Required parameter.)

EXAMPLES

```
?$2:=44
?DR,2
2=44
?DR,S
S=26
?$$S:=$S+2
?DR,S
S=30
```

Set register 2 to the octal value 44.
Verify.

Display value of S register.

Increment S register by 2.

Verify

TEXT DISCUSSION

Page 3-17.

Invokes the DEBUG facility in order to set breakpoints, and modify or display data stored in memory.

DEBUG;

CONDITION CODES

The condition code remains unchanged.

TEXT DISCUSSION

Page 3-21.

STACK DUMP SPECIFICATIONS

The specifications for the intrinsics and MPE commands that comprise the stack dump facility are listed in alphabetic order in the following pages. Table 2-2 summarizes the commands and intrinsics and their functions.

Table 2-2. Stack Dump Facility Summary

COMMAND	INTRINSIC	PURPOSE
	RESETDUMP	Programmatically disables stack dump in case of program abort.
:RESETDUMP		Disables stack dump in case of program abort.
	SETDUMP	Programmatically enables a stack dump in case of program abort.
:SETDUMP		Enables stack dump in case of program abort.
	STACKDUMP	Dumps selected portion of stack to file or list device.

RESETDUMP

Intrinsic

Programmatically disables the abort stack analysis facility for the calling process but has no effect on any of the processes of the family.

```
RESETDUMP;
```

CONDITION CODES

CCE Request granted.

CCG Abort stack analysis facility was disabled prior to the RESETDUMP intrinsic call and remains disabled.

CCL Not returned by this intrinsic.

TEXT DISCUSSION

Page 4-7.

:RESETDUMP

MPE command

Disables the abort stack analysis facility for the current process only.

SYNTAX

<code>:RESETDUMP</code>

:RESETDUMP can be specified in a session or a job (not during a DEBUG break) to terminate the effects of any previous :SETDUMP command. If no :SETDUMP has been specified, :RESETDUMP is ignored.

TEXT DISCUSSION

Page 4-7.

SETDUMP

Intrinsic

Programmatically enables the abort stack analysis facility for the caller process (and subsequent sons).

LV SETDUMP(<i>flags</i>);

PARAMETERS

flags *logical by value (required)*

A logical word whose bits specify the following:

Bit 15 = If *on*, specifies a DL to Q initial dump.

Bit 14 = If *on*, specifies a Q initial to S dump.

Bit 13 = If *on*, specifies a Q-63 to S dump. This bit is ignored if bit 14 is on.

Bit 12 = If *on*, causes an ASCII dump of the octal content along with the octal values.

A 0 value for *flags* results in the display of registers and stack marker trace only.

CONDITION CODES

CCE Request granted.

CCG Abort stack analysis facility already enabled before SETDUMP intrinsic call. The facility is now set up according to new specifications from this call.

CCL Not returned by this intrinsic.

TEXT DISCUSSION

Page 4-5; 4-6.

:SETDUMP

MPE command

Enables the abort stack analysis facility for the current process and any subsequently created sons.

SYNTAX

```
:SETDUMP [ DB [ ,ST [ ,QS ] ] [ ;ASCII ] ]
```

NOTE

DB, ST, and QS are keyword parameters and can be specified in any order.

PARAMETERS

- | | |
|--------|---|
| DB | Specifies a DL to Q initial dump. (Optional parameter.) |
| ST | Specifies a Q initial to S dump. (Optional parameter.) |
| QS | Specifies a Q-63 to S dump. This parameter is ignored if ST is specified. (Optional parameter.) |
| ;ASCII | Causes an ASCII conversion of the octal content to be dumped along with the octal values. (Optional parameter.) |

NOTE

In an interactive session, all parameters are ignored and the only effect of the :SETDUMP command is to enable the stack dump facility in order for the process to go to DEBUG in case of an abort.

EXAMPLES

```
:SETDUMP DB;ASCII
```

Specifies a DL to Q initial dump and an ASCII conversion along with the octal values.

TEXT DISCUSSION

Page 4-3; 4-6.

Dumps selected parts of stack to a file.

SYNTAX

```
          BA      I  L  DA  O-V
STACKDUMP(filename,idnumber,flags,selec);
or
          BA      I  L  DA  O-V
STACKDUMP'(filename,idnumber,flags,selec);
```

PARAMETERS

filename *byte array (optional)*

Contains the file name of the file where the information is to be dumped. When *filename* contains the formal designator of the file, the file will be opened and closed by the STACKDUMP intrinsic. If the secondary entry point (STACKDUMP') is used to enter this intrinsic, MPE assumes that *filename(0)* contains the *file number* of a file which has been successfully opened prior to the call to STACKDUMP. In this case, the file is not closed before returning to the user program. When a file number is passed via the STACKDUMP' secondary entry point, the record length must be between 32 and 256 words and write access must be allowed to the file.

Default: Dump is sent to the standard list device.

idnumber *integer (optional)*

An integer which is displayed in the header of the dump to identify the printout.

Default: Identifying integer is not displayed.

flags *logical (optional)*

A logical value used to specify the following options:

Bit 15 = 1 Suppress ASCII dump.

Bit 14 = 1 Suppress trace back of stack markers.

Default: Bits 14 and 15 = 00. A corresponding ASCII dump is provided for all values dumped in octal, and a trace back of stack markers is displayed.

selec *double array (optional)*

Specifies which stack areas are to be dumped. The format of the array is shown in figure 2-1. The array has no predetermined length; the first double word containing the values 0/-1 indicates the end of the array. An entry for which the count is 0 is understood to be a "skip" (i.e., go to next double word element in list).

Default: If missing, or if the first double word contains 0/-1 (indicating end of array), then no dump (except for the header) occurs, unless flags bit 14 = 0, in which case the trace back of stack markers is displayed.

CONDITION CODES

- CCE Request granted.
- CCG Request denied. Bounds violation occurred and the dump was not completed. Record size was not between 32 and 256 words.
- CCL Request denied. File system error occurred from opening, writing, or closing of the file. The file error number is returned to *idnumber*. See the *MPE Intrinsic Reference Manual* for a discussion of file errors.

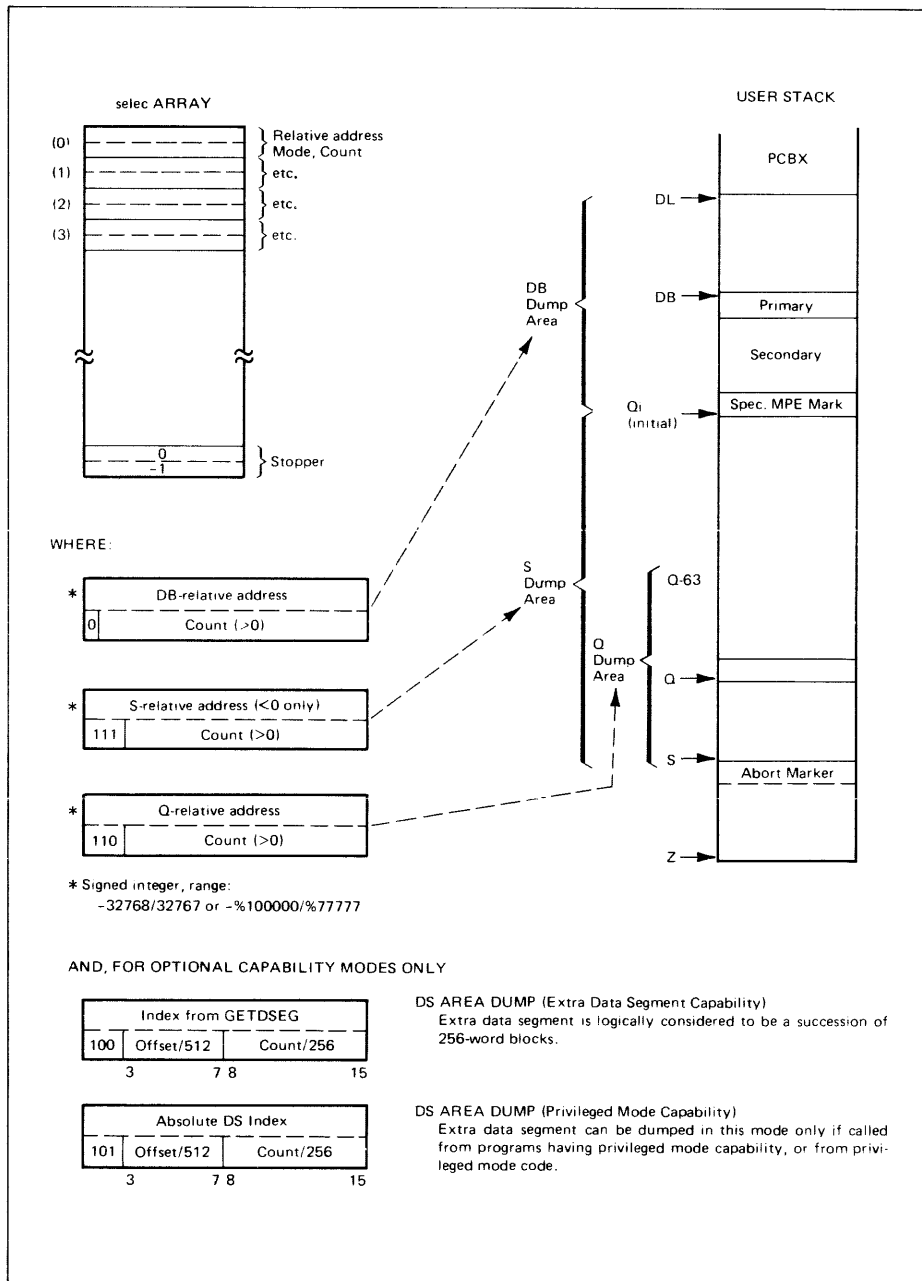


Figure 2-1. Stack Dump Modes and *selec* Array Format

TEXT DISCUSSION

HOW TO USE DEBUG

SECTION

III

DEBUG is accessed by specifying the DEBUG parameter in a RUN or PREPRUN command, or by inserting DEBUG intrinsic calls in your program. The purpose of DEBUG is to establish breakpoints in an executing program so that you can examine or modify current values in memory or the current values of program registers.

PREPARING TO USE DEBUG

In order to use DEBUG, it is important to know the memory locations where your program symbols are stored, the beginning locations of each program unit, and the offset from these locations of each line of code. This information is obtained in several ways depending somewhat on the source language of your program.

A FORTRAN program should be compiled with the MAP, LABEL, and LOCATION parameters in the \$CONTROL command; COBOL and SPL programs should be compiled with the MAP parameter. LABEL provides a label map showing the offset of each labelled statement, and LOCATION provides the offset of every statement as part of the source listing. In an SPL program, this information is provided automatically in your compilation listing. The MAP parameter generates a symbol map that lists all the symbolic names in your program and the location in the data stack where the data is stored for each symbol.

Any program to be debugged should be prepared with the PMAP parameter in order to determine the beginning code location of each program unit and, if more than one segment is used, the segment number.

If you want to use DEBUG but your program has already been compiled and prepared, you can run the DECOMP program to obtain the necessary code locations. For COBOL, which does not have a LOCATION parameter, this is usually necessary. To run DECOMP, enter:

```
:RUN DECOMP.PUB.SYS
```

You will be prompted for the program file name, the starting segment number and the program entry point location. These numbers are listed when you prepare your program with PMAP. (Refer to Appendix A for an example showing execution of DECOMP for the sample FORTRAN program used in this manual.)

For the purposes of this manual, FORTRAN has been chosen as the language to be used in examples. An SPL user will find that DEBUG has fewer complications for his language than it does for FORTRAN and should simply skip over the descriptions of determining data locations that are peculiar to the FORTRAN compiler.

Figure 3-1 shows a simple FORTRAN program used to demonstrate use of the DEBUG commands. The compilation includes the MAP, LABEL, and LOCATION parameters. If you want an octal dump of the program, you can also include the CODE parameter, but this is usually not necessary. Figure 3-2 shows the result of preparing the program using the PMAP parameter in the :PREP command. (Note that PMAP can be included in :FORTPREP if you compile and prepare in one operation.) In both these figures, significant entries are indicated by numbers keyed to the following discussion.

NOTE

All numbers in the PMAP listing except elapsed time and processor time are octal values; all numbers in the symbol map are decimal; in the label map, the labels are decimal, the code offset is octal. DEBUG itself expects octal values unless otherwise indicated.

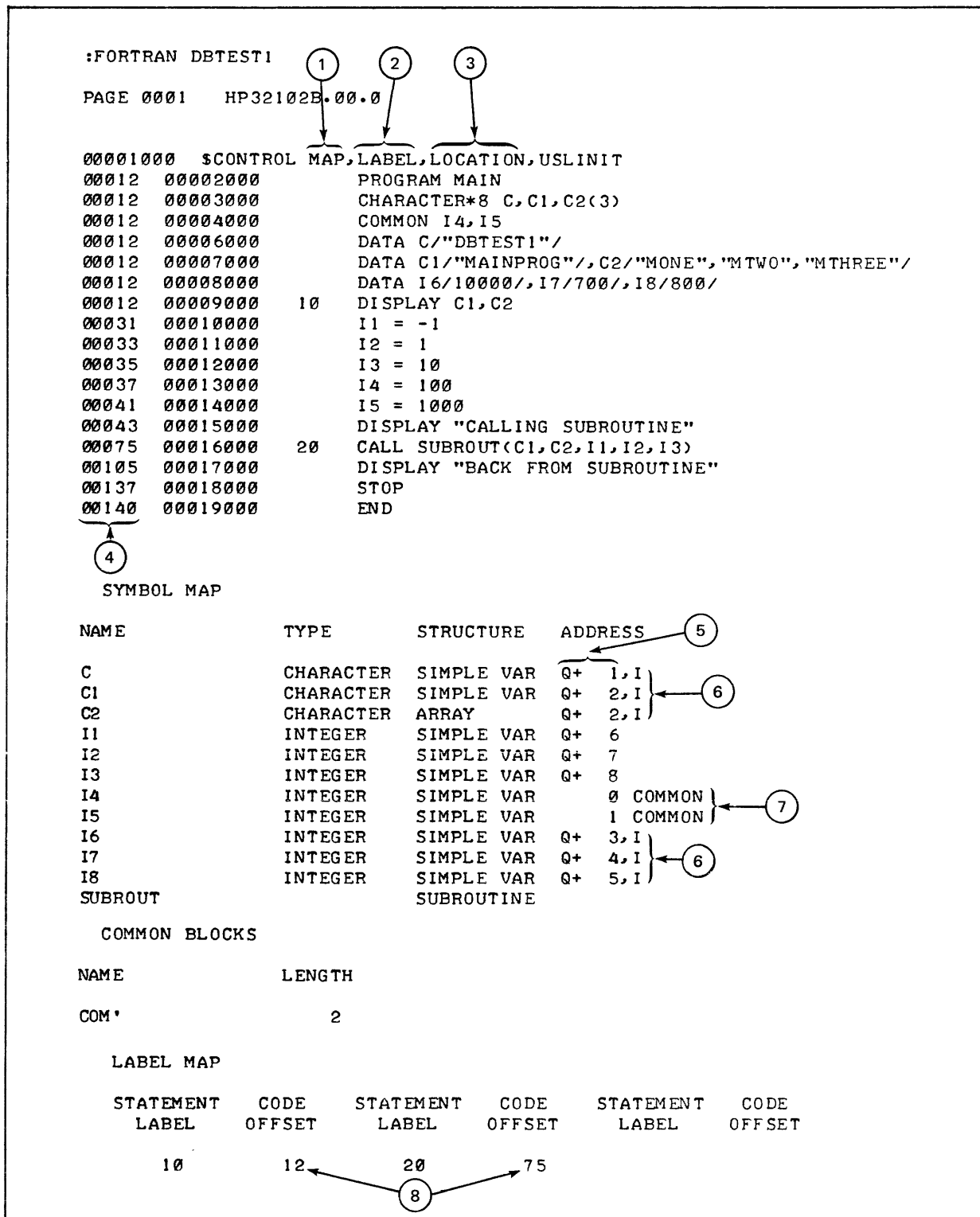
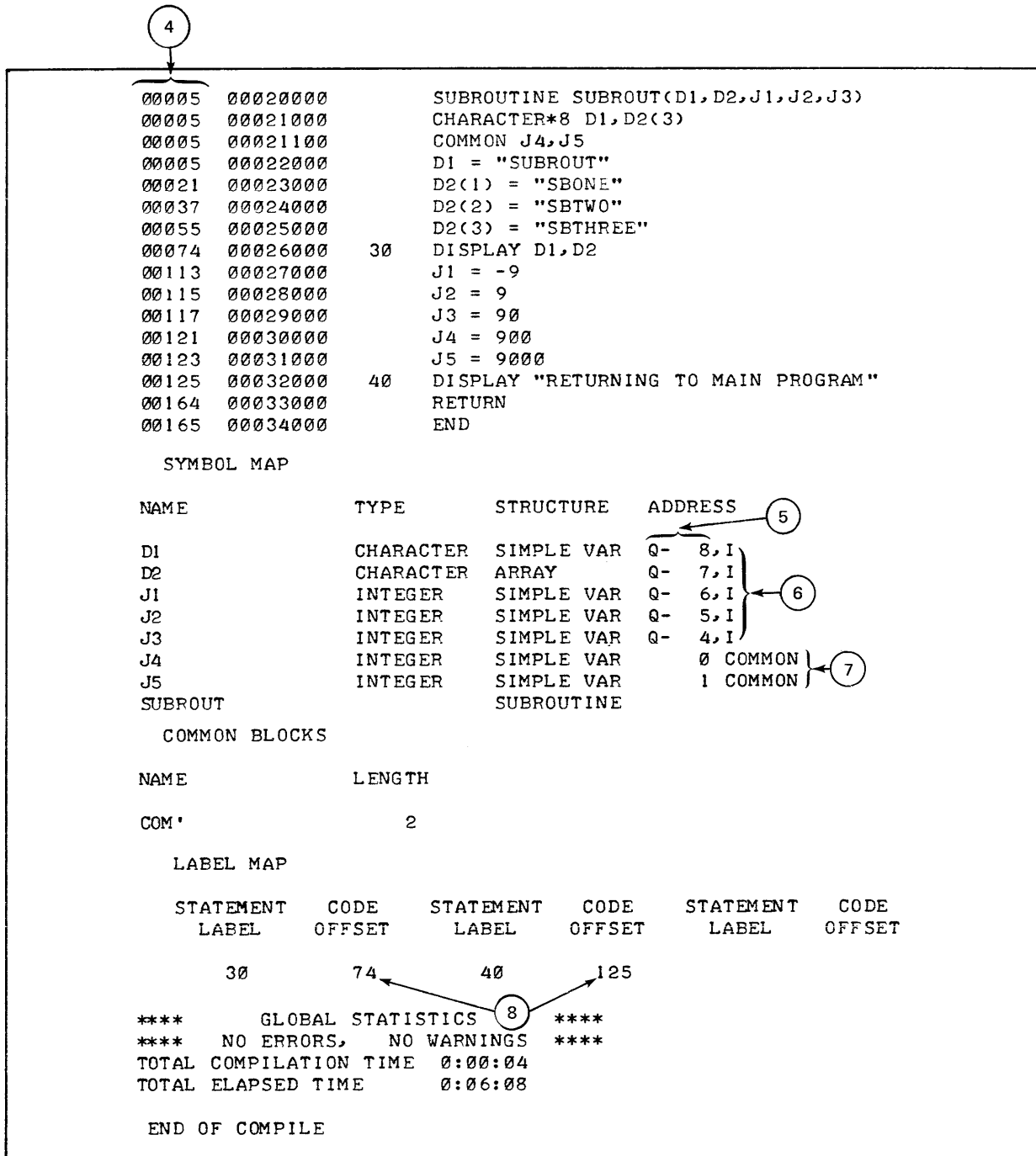


Figure 3-1. FORTRAN Compilation Showing Symbol and Label Maps



The key items in figure 3-1 are:

- 1 MAP parameter included to produce symbol map of each program unit; in this example, one map for program MAIN and one for subroutine SUBROUT.
- 2 LABEL parameter included to produce a label map for each program unit.
- 3 LOCATION parameter included to produce location of each instruction relative to start of code.
- 4 Location of each instruction relative to start of code.

- 5 Memory address relative to Q location in data stack of each symbol except those declared COMMON.
- 6 I indicates the address is indirect; used for all values declared in DATA statements or passed from another program unit.
- 7 Location of symbol relative to start of common.
- 8 Location of each labelled instruction relative to start of code.

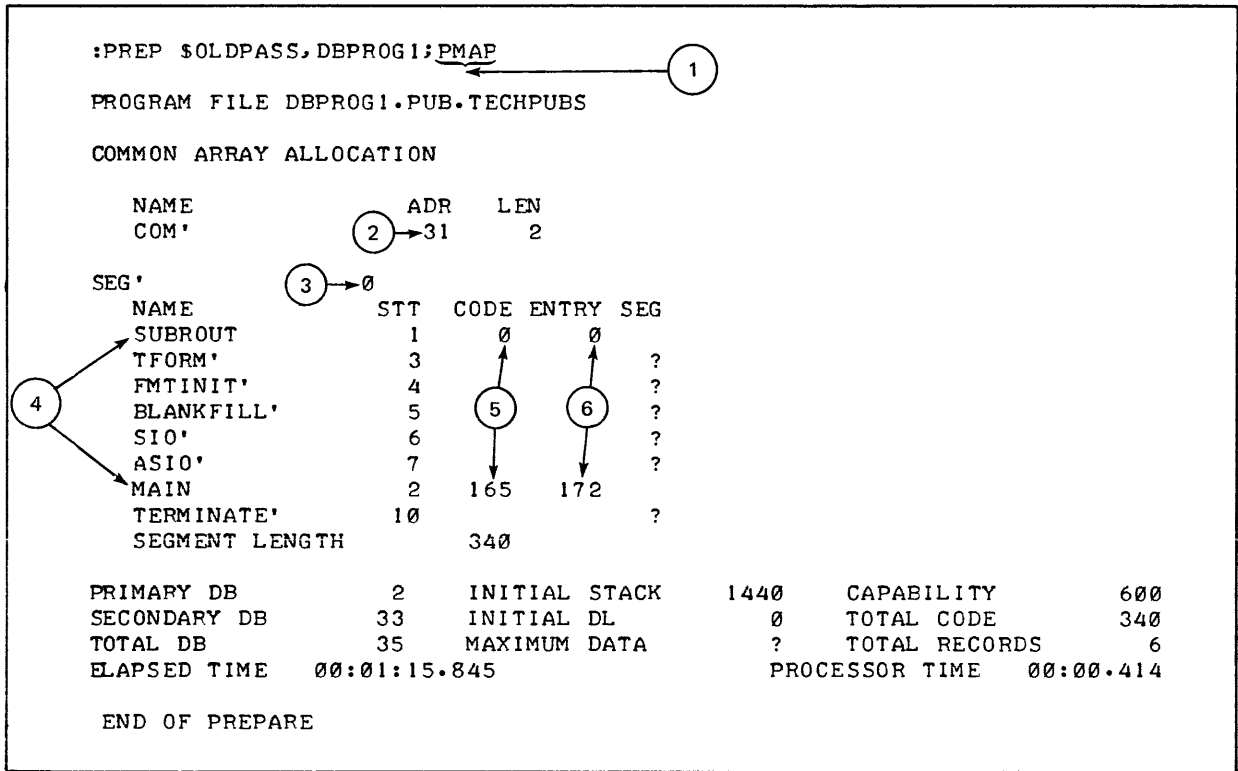


Figure 3-2. Result of Preparation With PMAP

The key items in figure 3-2 are:

- 1 PMAP parameter included to list beginning code locations of each program unit and relative location of common.
- 2 Location of unlabelled common relative to DB. If used in program, labelled common would follow.
- 3 Logical segment number of the program file.
- 4 Program unit entry point; corresponding question mark (?) under heading SEG indicates that the procedure is external to the program.
- 5 Beginning location of code in the segment (start of code).
- 6 Location within this segment of entry point to program unit.

USING DEBUG WITH STANDARD CAPABILITY

Figure 3-3 shows a sample execution of program DBTEST1 (program file DBPROG1) using DEBUG commands to establish breakpoints, display values and registers, and modify values and registers. This run illustrates use of DEBUG commands with the standard capability. (Use of the DEBUG intrinsic is illustrated in figure 3-8. Use of DEBUG with privileged mode capability is discussed later in this section. The stack dump facility is discussed in section IV.)

If you want to debug a compiled and prepared program, you specify the DEBUG parameter in the :RUN command as follows:

```
:RUN program file name; DEBUG
```

The program file prepared in figure 3-2 is named DBPROG1. To run this program in DEBUG mode, enter:

```
:RUN DBPROG1;DEBUG
```

If the program has only been compiled but not prepared, you may include DEBUG in a :PREPRUN command to prepare and execute in DEBUG mode. Since you will probably need a map, include the PMAP parameter as follows:

```
:PREPRUN USL file name; DEBUG;PMAP
```

Or, assuming the program DBTEST1 has just been compiled:

```
:PREPRUN $OLDPASS;DEBUG;PMAP
```

Since both DEBUG and PMAP are keyword parameters, they can be entered in any order after the file name.

In order to use DEBUG, you must have write access to the program and be operating in a session. MPE checks whether you have read/write access to the program file and are in an interactive session. If either condition is not true, control is returned to you.

When a program executed in DEBUG mode starts execution, it breaks before the first instruction so that you can establish your breakpoints. At this initial break, a message is displayed indicating that you are in DEBUG mode and specifying the entry point location relative to the start of code of the first program unit to be executed. (Refer to item 1 in figure 3-3.) The message has the form:

```
*DEBUG* s.nnn
```

where

s is the segment number

nnn is the location of the entry point.

DEBUG then displays the prompt for DEBUG commands, a question mark. In response to this prompt, you may enter any DEBUG command for which you have the capability. Usually, the first command you enter is B or R in order to establish where you want to break during program execution. The B command simply establishes breakpoints or displays the existing breakpoints. R both establishes a breakpoint and resumes program execution. Whenever a break occurs, you must specifically request execution to continue with an R command or, alternatively, the E command.

WHAT IS A BREAKPOINT?

A breakpoint is a pause in program execution during which control is given to the DEBUG program until you specifically request that execution resumes. During this pause, you may display any data registers belonging to your program or any program registers that currently contain program code. Bounds are set so that you cannot access data or code not belonging to your program. Exceptions to this general rule are made if you have the privileged mode capability that allows you to display and modify system areas.

When a breakpoint is reached, the program pauses just prior to the instruction specified in the B or R command. Following the breakpoint that instruction is executed.

At each breakpoint, a message is issued showing the current breakpoint location. The form is:

```
*BREAK* s.nnn
```

where

s is the segment number of the current segment

nnn is the location of the breakpoint relative to the start of code.

HOW TO ESTABLISH BREAKPOINTS

In order to determine the breakpoints, look at the map produced during program preparation by PMAP (figure 3-2). This program has only one unnamed segment listed as SEG' with segment number 0 (see item 3). If you had divided your program into segments, the PMAP listing would show each segment in ascending order by number. The segment number is only necessary in a segmented program and then only if you want to set a breakpoint in a segment other than the currently executing segment.

The offset from the start of the code must, however, always be specified. This offset is relative to the start of code shown for each program unit in the PMAP listing. Refer to item 5 in figure 3-2 for the start of code in the main program (location 165) and in the subroutine (location 0).

For example, suppose a program with five segments. The first entry point is in segment 0 and you want to set a breakpoint in segment 4 at location 200 relative to the start of code in that segment, you enter:

```
?B 4.200+5           where 5 is the start of code, 4 is the segment number and you  
                        want to break 200 (octal) locations past the start of code.
```

Returning to the sample program, suppose you want to break at a labelled statement, you can refer to the label map in figure 3-1 for the offset of each program label from the start of code.

For example, to break at label 10 in the main program, enter:

```
?B 165+12 or ?B 177      (note that the offset is octal)
```

To break at label 30 in the subroutine, enter:

```
?B 74                    (relative to location 0)
```

More than one breakpoint can be specified in one B command:

?B 165+ 12, 165+ 75, 74 (break at labels 10, 20, and 30)

To break at an unlabelled statement, use the location (item 4, figure 3-1) as the offset from the start of code. For example, to break at statement 13 in the main program, enter:

?B 165+37 (37 is the offset to start of code in MAIN)

REPEATED BREAKPOINTS. If your program loops through a breakpoint more than once, you may specify that the breakpoint be set:

- only at the nth occurrence (conditional breakpoint)
- at every occurrence (permanent breakpoint)
- at every nth occurrence. (permanent conditional breakpoint)

For example, to break the third time the program executes the instruction at location 155:

?B 155:3 (break the 3rd time 155 is executed)

Following this break, the breakpoint is cleared and no breaks will occur in subsequent executions of location 155.

To break every time the program executes the instruction at 155, enter:

?B 155:@ or ?B 155:@1 (break each time 155 is executed)

This is a permanent breakpoint, indicated by @, and remains in effect until the program terminates or the breakpoint is specifically cleared.

To break every third time the instruction is executed, enter:

?B 155:@3 (break every third time 155 is executed)

NOTE

When a breakpoint is set, the program pauses in its execution *before* the instruction at the specified breakpoint location is executed.

DISPLAYING BREAKPOINTS. All existing breakpoints are displayed by the following command:

?B@

In response, the system prints the logical segment number (LCST) and the P register location of each existing breakpoint. (Refer to item 3, figure 3-3.)

If a breakpoint is permanent, the @ symbol is displayed following the LCST and P values. The number of times the program loops through such a breakpoint is also displayed followed by a slash zero if the break is executed every time. For example:

```
?B 165+25:@
?B@
LCST=0      ,P=212  ,@1/0
```

If a conditional breakpoint is set, for instance to break every third time the statement at 165+ 25 is executed, this is indicated in the breakpoint display as follows:

```
?B 165+25:@3
?B@
LCST=0      ,P=212  ,@3/0
```

Each time the statement at 212 is executed, the count is increased by one. When the breakpoint is taken, it is reduced to zero once again.

If the breakpoint is not permanent, but a count was specified, the breakpoint is cleared after the break is executed on the *n*th pass through the statement. For instance, if a break on the third pass through location 212 is specified:

```
?B 165+25:3
```

At a break following the second execution of location 212, the breakpoint is displayed as:

```
*BREAK* 0.262
?B@
LCST=0      ,P=212  , 3/2
```

When the breakpoint at 212 is taken, the breakpoint is cleared:

```
*BREAK* 0.212
?B@
?
```

CLEARING BREAKPOINTS. A breakpoint specified for one time only is cleared when execution resumes after the break has occurred. A permanent breakpoint is cleared only when the program terminates.

For example, if the following breakpoints are set:

```
?B 165+ 12, 165+ 75, 74, 125
```

When the break at location 165+ 12 has been taken, that breakpoint is cleared. Similarly with each of the others.

If breakpoints are permanent as specified in the following command:

```
?B 155:@, 175:@ 5
```

The breakpoint at each of these locations is cleared only when the program terminates.

You may, however, clear any breakpoints that remain with the C command. For example, suppose the breakpoint at 165+ 12 has been taken, before resuming execution of the program, you can clear all the remaining breakpoints by entering:

```
?C@
```

If you only want to clear one of the remaining breakpoints, it must be explicitly specified, for instance, as:

```
?C 165+ 75
```

Several breakpoints may be cleared as follows:

```
?C 165+ 75,125
```

If you attempt to clear a breakpoint that has not been set, the command is ignored.

RESUMING EXECUTION. When you have displayed or modified at the current breakpoint, you can enter either R or E in order to resume execution. R with no parameters simply resumes execution at the current breakpoint location. (Refer to item 4, figure 3-3.)

You can resume and also set a new breakpoint with the R command. The breakpoint is specified in exactly the same way as in the B command, except that only one breakpoint can be specified with R. (Refer to item 4, figure 3-3.)

Another way to resume execution is by entering the E command with no parameters. Used in this way it produces the same effect as R with no parameters. (Refer to item 6, figure 3-3.) With parameters E is used to delete values from the top-of-stack, S. This capability is illustrated in section 4 in the discussion of interactive use of :SETDUMP.

E@ can be used to terminate a program executing in DEBUG mode.

Used in this way, it is similar to pressing the BREAK key during normal program execution and then entering the :ABORT command. (Refer to item 7, figure 3-3.)

SWITCHING DISPLAY TO LINE PRINTER

If you are at a CRT terminal and want hard copy of your output, or if your output does not easily fit a teleprinter (as in a memory dump), you can switch your display to a line printer through the L command. In this command, you identify the printer by a file reference. You must, therefore, equate the file reference to a device in a :FILE command before running your program in :DEBUG mode.

For example:

```
:FILE PRINTER;DEV=LP  
:RUN YOURPROG;DEBUG
```

```
*DEBUG* 0.177
```

```
?L PRINTER
```

(opens PRINTER and directs output to line printer)

When output is complete, and before exiting from DEBUG, enter an L command to close the file PRINTER:

```
?L
```

Note that if your output is spooled, it will appear after the line printer is closed.

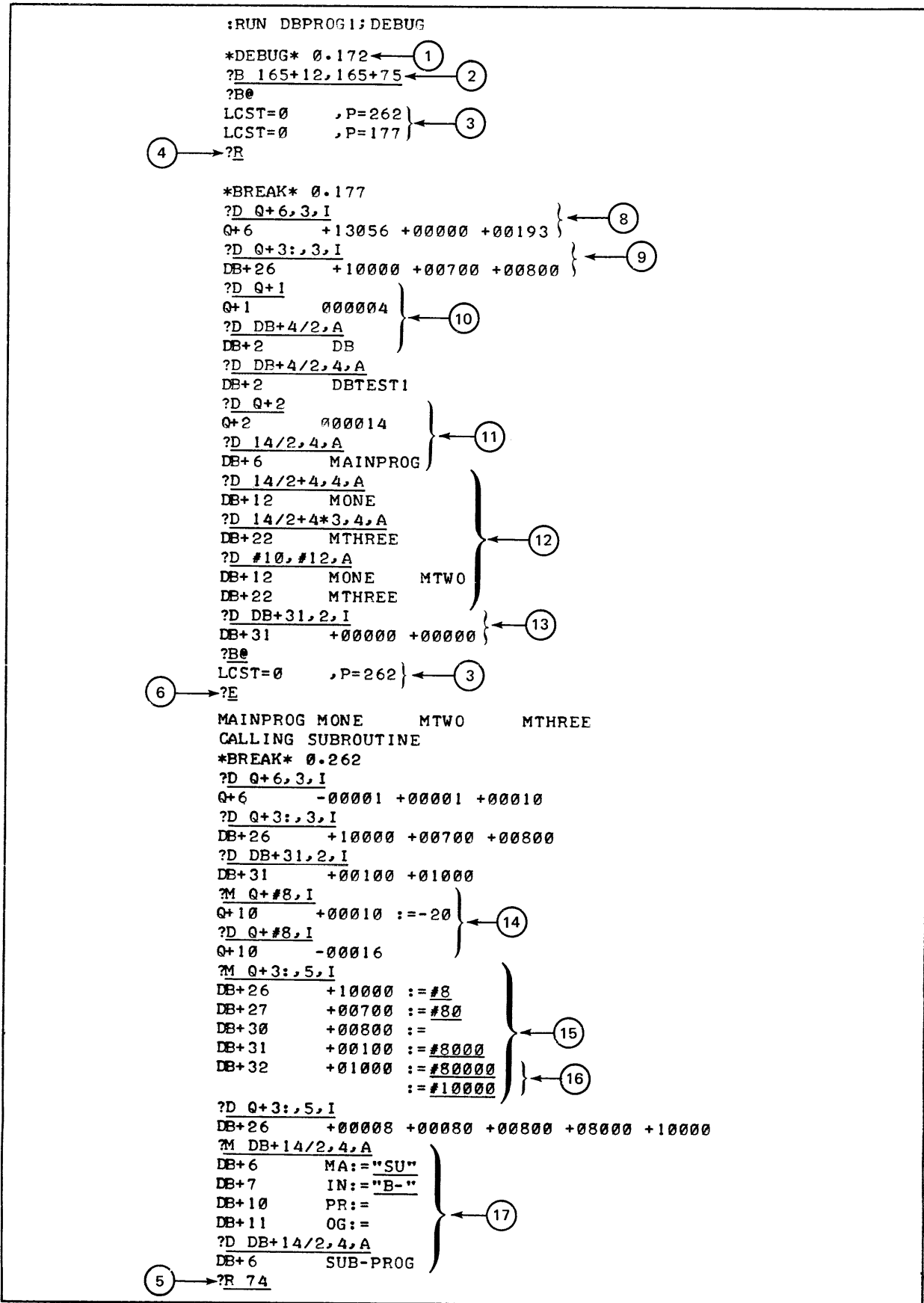


Figure 3-3. DEBUG Example

```

*BREAK* 0.74
?DR,1,2,3,4
1=0,2=0,3=0,4=0 } ← 18
?MR,1
1=0:=077777
?M Q-6:,3,I
DB+47 -00001 :=$1
DB+50 +00001 :=$1
DB+51 +00010 :=$1
?D Q-6:,3,I
DB+47 +32767 +32767 +32767 } ← 19
?S2:=166666 } ← 20
?S3:='DB+51'
?='DB+51'/2
=37777 } ← 21
?=37777,I
=+16383
22 → ?S4:='DB+51'/2
?M DB+26,3,I
DB+26 +00008 :=$2
DB+27 +00080 :=$3
DB+30 +00800 :=$4
?D DB+26,3
DB+26 166666 077777 037777
?D DB+26,3,I
DB+26 -04682 +32767 +16383
23 → ?R 125

SUBROUT SBONE SBTWO SBTHREE

*BREAK* 0.125
?D DB+47,3,I
DB+47 -00009 +00009 +00090
?D DB+31,2,I
DB+31 +00900 +09000
?DR
ST=60301,X=0,DL=177602,Q=62,S=62,Z=1475,P=125 } ← 18
1=0,2=0,3=0,4=0,LCST=0
7 → ?E@

END OF PROGRAM

```

Figure 3-3. DEBUG Example (Continued)

HOW TO DISPLAY AND MODIFY VALUES

The symbol map produced during compilation by the MAP parameter is the key to finding the location of most values. For simple variables to which values are assigned dynamically in the program, the map gives the Q-relative location where the assigned value is to be stored. (Refer to items I1, I2, and I5 in the main program symbol map, figure 3-1.)

If, however, the value has been assigned in a DATA statement, is a string value or array, or has been passed from another program unit via a program call, the Q-relative location in the symbol table is indirect, that is, it contains the address where the value is located. Indirect addresses are followed by the letter I in the symbol map. (Refer to items C, C1, C2, I6, I7, I8, and J1, J2, J3 in the symbol maps, figure 3-1.)

When the variable contains characters, the location is given as a byte rather than a word address. To find the word address, some simple calculation is usually required. (Refer to items C, C1, C2, and D1, D2 in the symbol maps, figure 3-1.)

Data stored in COMMON is specifically indicated in the symbol map. Its location is not given as a Q-relative address but as an offset to the beginning of COMMON. This, in turn, can be found from the PMAP listing (item 2, figure 3-2) where the start of COMMON is shown as a DB-relative address. (Refer to items I4, I5, and J4, J5 in the symbol maps, figure 3-1.) In the example, only unlabelled common is used; any labelled common would be listed immediately following unlabelled common in the maps.

DISPLAYING VALUES. The location of values is usually specified relative to the display bases DB or Q. To display values stored in a location relative to Q, enter:

?D Q+6 (Q is the display base, 6 the offset)

If you want to display more than one value, you must specify a count separated by a comma from the offset. For instance, to display the three values stored in Q+6, Q+7, Q+8, enter:

?D Q+6,3

All values are displayed as octal numbers by DEBUG unless you indicate that you want them to be decimal (I) or ASCII (A). The mode indicator may follow the offset separated by a comma or it may follow the count, also separated by a comma. For instance, to display the value at Q+6 as a decimal number, enter:

?D Q+6,I

Any values for which you know the DB relative location can be displayed without specifying the display base. For instance, to display the contents of DB+31, you can simply enter:

?D 31

So far, the display examples have used octal offsets. If you want to display the value in I3 stored at location Q+8, you must either indicate that the offset is decimal, as:

?D Q+#8

or you must enter the octal equivalent of the decimal number, as:

?D Q+10

Usually, it is simpler to use the indicator # than to convert for yourself. Because the symbol map shows Q-relative locations as decimal values, you must be very careful to use the decimal indicator for these locations. If you enter a decimal number in a DEBUG command without an indicator, you will receive an error message. If, however, the number appears to DEBUG to be octal, but was meant to be decimal, a value may be displayed from the wrong location or you may receive a BOUNDS violation.

Indirect Addressing. To display a value stored in an indirect address, follow the location with a colon (:). For example:

?D Q+3:,3,I

If a count and/or a mode indicator is included, it must be preceded by a comma; the colon does not act as a delimiter. (Refer to item 9, figure 3-3 for an example displaying indirectly addressed values.)

When indirect addresses are used, DEBUG displays the DB-relative location to which the Q-relative address points.

Byte Addressing. The addresses of items containing characters are specified as bytes rather than as words. Thus, if you display the contents of Q+ 1 (the address of character element C), this address is given as the number of bytes relative to the start of DB rather than as the number of words. (Refer to item 10, figure 3-3.) To determine the word address that you will need to display the character element, you must divide this byte address by 2. For example, to locate item C:

```
?D Q+ 1
Q+ 1          000004          (DB+ 4 bytes)
?D DB+ 4/2,A
DB+ 2        DB
```

Another method is to use the form that specifies the contents of a location. This allows you to perform the same function in fewer steps. For example:

```
?D 'Q+ 1'/2,A
DB+ 2        DB
```

Note that only the first two characters of C are displayed. In order to display the entire character element, you must determine the number of words per element. In line 3 of the sample FORTRAN program, CHARACTERS*8 defines the length of each of the succeeding character elements as eight characters. To find the number of words per element, divide by 2. In this case, C, C1, and C2 require four words per element. (If the number of characters per element is an odd number, you must add one character before dividing by two.) This value gives you the *count* so you can display the entire element. For example, to display all of C:

```
?D DB+ 2,4,A          or          ?D 'Q+ 1'/2,4,A
```

The characters in C1 are displayed in a similar manner. (Refer to item 11, figure 3-3.)

Array Addressing. Note that the address of the array C2 is stored in the same location (Q+ 2) as the address of C1. This is because the FORTRAN compiler has stored C1 in the unused zero element of array C2. This is done frequently in order to save space. In order to display the first element of array C2, you must add the number of words per element to the word address calculated from the byte address stored in Q+ 2. For example, to display C2(1), enter:

```
?D DB+ 14/2+ 4,4,A          or          ?D 'Q+ 2'/2+ 4,4,A
```

To display the element C2(3), enter:

```
?D DB+ 14/2+ 4*3,4,A          or          ?D 'Q+ 2'/2+ 4*3,4,A
```

To display the entire array, enter as the *count* the number of words per element multiplied by the total number of elements in the array. In the case of C2 with 3 4-word elements, enter:

```
?D DB+ 14/2+ 4,3*4,A
```

If you wish, you can perform the calculations for *count* and *expression* and enter the results. In performing these calculations, remember that DEBUG, by default, uses octal values and any decimal values must be preceded by the prefix #.

The displays resulting from addressing arrays and array elements are shown in item 12, figure 3-3.

A general formula for finding the word address of the start of an array element is:

$$B/2 + I*W$$

where:

B is the byte address provided by DEBUG

I is the index to the array element, i.e., 3 for C(3)

W is the number of words per element.

Addressing Common. The addresses for items in COMMON are shown in the symbol table as relative to the start of COMMON. For example, items I4 and J4 are at location 0, I5, and J5 at location 1. The DB-relative address of the start of COMMON is given in the PMAP listing generated during program preparation (figure 3-2). In this example, COMMON begins at location 31. Thus to display either I4 or J4, enter:

?D 31,I

To display I5 or J5, enter

?D 32

(Refer to item 13, figure 3-3 for an example of displaying COMMON items.)

Figure 3-4 shows the Q-relative and DB-relative address for the data items used in the sample FORTRAN program.

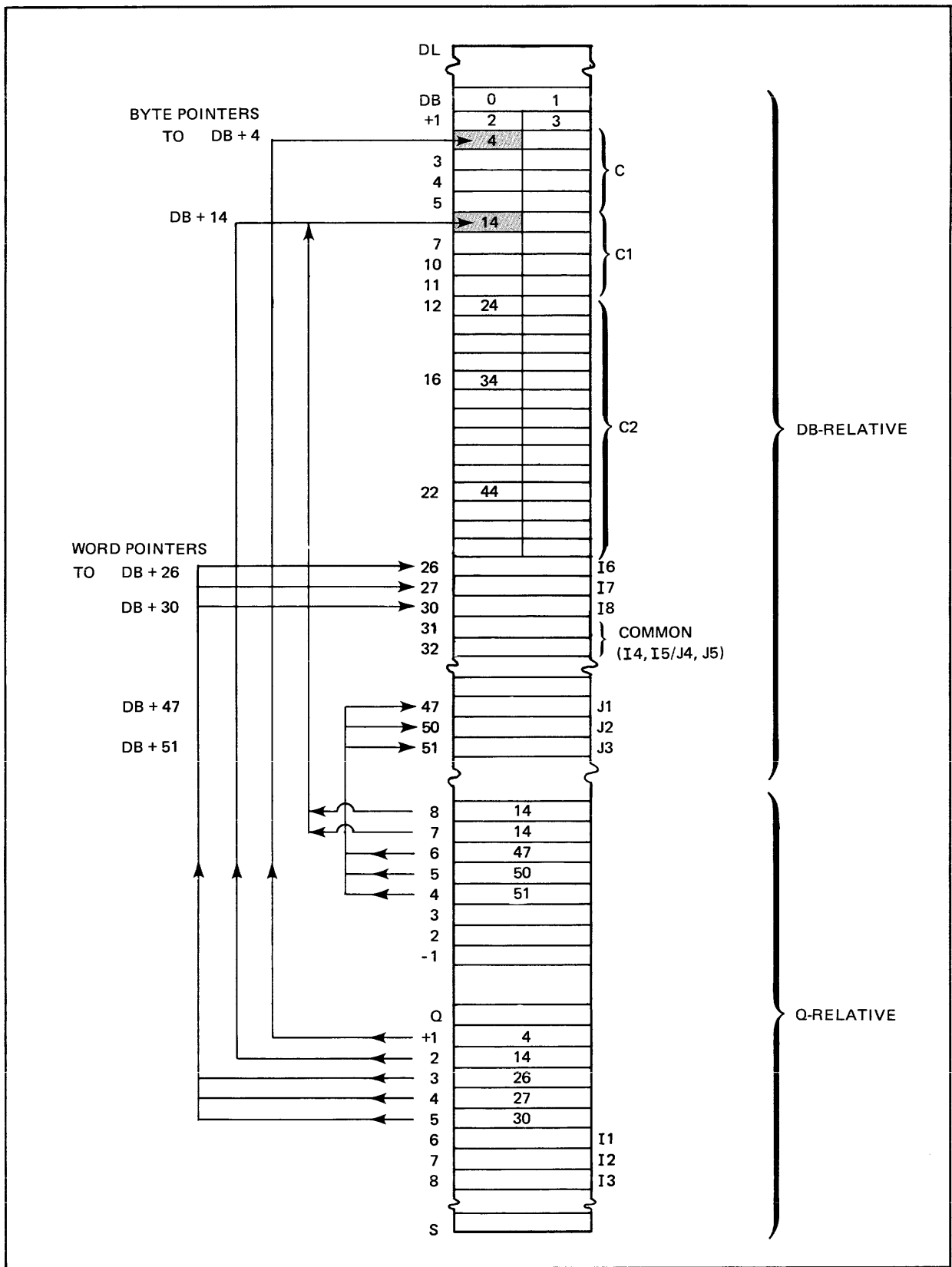


Figure 3-4. Layout of Items in Data Stack

MODIFYING VALUES. Values can be modified with the M command. This command mirrors the D command in its specification so that the methods for finding values to be modified are the same as those described for finding values to be displayed.

When the M command is executed, DEBUG prints the current value of each item to be modified followed by :=. You either enter a new value or press *return* to retain the current value. The values you enter are assumed to be octal unless surrounded by quotes (ASCII) or preceded by # (decimal).

When you specify a mode in the M command, this applies only to the display of current values; it does not apply to the values you enter. For example, if you want to modify the decimal value in Q+ 10, enter:

```
?M Q+ 10,I
```

DEBUG prints the current value as a decimal number and asks for a new value:

```
Q+ 10      + 00010 :=
```

You must enter a value or else press return. If you want to enter a decimal value, precede the number with #, otherwise, an octal value replaces the current value in Q+ 10. (Refer to item 14 in figure 3-3.)

If you want to modify more than one word, you enter a count:

```
?M Q+ 6,3
```

DEBUG responds by printing the first value followed by :=. When you have responded, it prints the second value followed by :=. This process continues until all requested values have been displayed and modified. (Refer to item 15 in figure 3-3.)

If the value you enter is for some reason unacceptable, the request is repeated until you enter an acceptable value. In item 16, figure 3-3, the value entered is too large. When a smaller value is entered, DEBUG accepts it and issues the next prompt.

When character strings are to be modified, the string is displayed two characters (one word) at a time. New character values must be surrounded by quotes. (Refer to item 17, figure 3-3.)

NOTE

Although your program code can be displayed (using DPB, DP, or DPL), it cannot be modified. Any needed modifications should be made to the source code followed by a recompilation.

DISPLAYING AND MODIFYING REGISTERS. A set of machine registers can be displayed and modified with the DR and MR commands. (Refer to the DR description in section II for a list of these registers.) In addition to the standard registers, four temporary registers, numbered from 1 through 4, are provided by DEBUG. These registers are particularly useful during modifications. The temporary registers are reset to zero at the start of each break so they cannot be used to transfer values past a breakpoint. (Refer to items 18 at Break 74 and then in Break 125, figure 3-3.)

Suppose you want to set the value of three data items to 32767 (octal 77777), you can first set a temporary register to that value:

```
?MR,1
1=0:= 77777
```

Then use the register contents as a value for modification:

```
?M DB+ 26,3,I
DB+ 26      + 10000 := $1
DB+ 27      + 00700 := $1
DB+ 30      + 00800 := $1
```

Note that the register must be preceded by a dollar sign (\$) to indicate that it is a register, not a digit. (Refer to item 19, figure 3-3.)

If you enter DR with no parameters, all available registers are displayed. (Refer to item 18 at Break 125, figure 3-3.) You can modify all registers by entering MR with no parameters. The MR description in section II contains such an example.

Using the \$ Command. Values can be assigned to any register using the \$ command. For example, to assign the octal value 077777 to register 1, enter:

```
?$1:= 77777
```

The value can take the form of an expression. Any numbers are assumed to be octal unless preceded by a #, the contents of a location can be specified by enclosing the address in apostrophes (i.e., 'DB+ 26'), and an ASCII character can be specified by enclosing it in quotes. Since each register is one word in length, no more than two characters can be assigned.

In each case, an octal value is assigned. For example, to assign the octal equivalent of the ASCII character A to register 4, enter:

```
?$4:="A"
```

You can then display the register by entering:

```
?DR,4
4= 101          (DEBUG displays octal value of register 4)
```

(Refer to items 20 and 22 in figure 3-3 for examples using the \$ command.)

Using the = Command. The result of any expression can be displayed by using the = command. The expression itself can contain register contents (\$Q), decimal values (#100), or location contents ('DB+ 51'), as well as octal values.

For example, if you enter:

```
?= 30*42
```

DEBUG responds with the result of the calculation preceded by an equals sign:

```
= 1560
```

The result is always displayed in octal unless a *mode* parameter is included. For instance, if you want to display the contents of location Q+26 in decimal, enter:

```
?='Q+26',I
```

DEBUG returns the decimal equivalent of the contents of this location:

```
= -1
```

In the following example, the expression is expanded:

```
?='Q+26'+#20,I    (request contents of Q+26 plus 20 in decimal)  
= +19
```

Or you can find the decimal equivalent of an ASCII character as follows:

```
?="A",I  
= +65
```

(Refer to item 21, figure 3-3, for examples using the = command.)

DISPLAYING CODE. If you want an octal dump of a portion of your program code, you can use the DP, DPB, or DPL commands. (You can also see the octal code generated by a compiler by running the program DECOMP or by including the CODE parameter in the \$CONTROL command when you compile. In both these cases, the entire program is dumped.) DP, DPB, and DPL are variations on the D command that specify a program area to be used as the display base. DP displays code relative to the program counter (P) that marks your current instruction.

DPB displays code relative to the program base (PB). DPL displays code relative to the program limit (PL). The code in the currently executing program unit starts at location PB. The program limit PL marks the end of the segment transfer table (STT) that follows each segment of code. (Refer to figure 3-5 for a diagram of the code layout.)

As with the D command, you can specify offsets, a count, and the mode of the display. Since you will usually want an octal display, the mode is omitted from the examples in figure 3-5.

In item 1, figure 3-5, twenty words of octal code at the current P-counter location are displayed. These words are the translation by the FORTRAN compiler of line 9 in the sample program (DISPLAY C1,C2). The resulting machine instructions start at the entry point of program MAIN, location 172 relative to PB. Thus, you could achieve the same result by entering:

```
?DPB+172,20
```

If you specify an offset in a DP command, it is relative to the P-counter, as:

```
?DP+20          (20 words past P)
```

Suppose the P-counter is set to location 172, then the same result is achieved by entering:

```
?DPB+172+20
```

The resulting display is illustrated in item 2, figure 3-5.

```

END OF PROGRAM
:RUN DBPROG1;DEBUG

```

```

*DEBUG* 0.172
?DP,20
P+0      034405 034404 040403 035004 004000 000707 021002 170015
P+10    031004 041402 021010 003200 031006 021010 021003 041402
?DP+20
P+20    021010
?D PB+172+20
PB+212  021010
?R 74

```

1

```

MAINPROG MONE      MTWO      MTHREE
CALLING SUBROUTINE

```

```

*BREAK* 0.74
?DP
P+0      000707
?DPB+74
PB+74    000707
?DPB
PB+0     035001
?DPL-1
PL-1    000000
?DPL-2
PL-2    000172
?E0

```

END OF PROGRAM

P = 74 (BREAK IN
SUBROUTINE)

P = 172 (BREAK IN
MAIN)

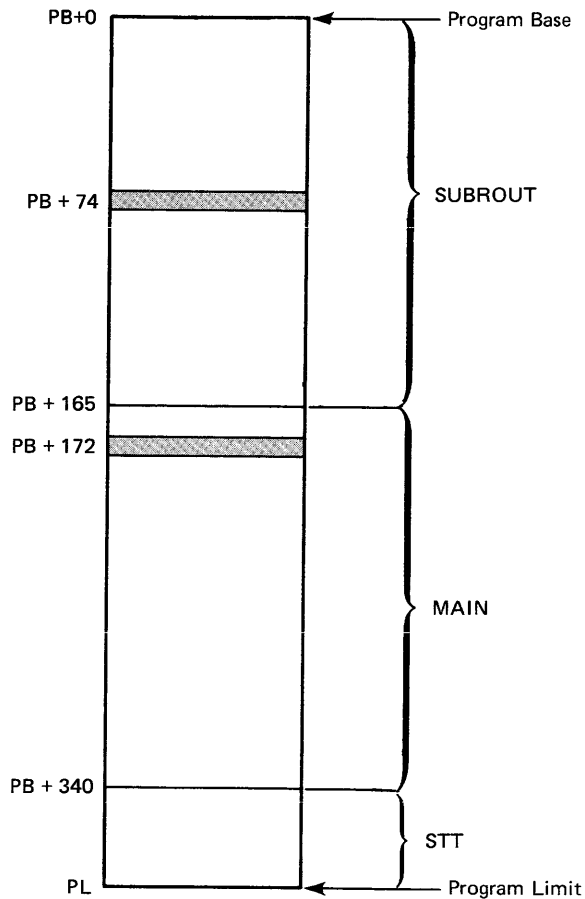


Figure 3-5. Display Code Locations Example

As shown in item 3, figure 3-5, if you want to display the contents of location PB+ 0, enter:

DPB

PL, the program limit, is at the end of the area allocated to your program segment and thus can have only a negative offset. As shown in item 4, figure 3-5, PL-1 contains the value zero. This is the last item in the stack transfer table and points to the entry point to SUBR. PL-2 is displayed as 172, the entry point to program MAIN. (Refer to the numbers under STT in the PMAP listing in figure 3-2. These numbers indicate the value to be subtracted from PL to determine values in the segment transfer table.) Note that the location pointed to by PL always contains the length of the STT in the right byte.

DISPLAYING STACK MARKERS. The T command allows you to trace stack markers through a series of nested subroutines (FORTRAN) or nested procedures (SPL). When T is executed, it displays for each routine, the displacement of Q to the initial Q, the relative P location and the logical segment number (LCST) to which control returns upon exit. The trace is useful particularly when you have a number of nested subroutines and you want to find out which calls which, where they return, and the displacement of Q for each.

For example, if T is entered at a breakpoint in the main program of the sample FORTRAN program used in this manual, the following values are displayed:

```
?T
Q-0      ,LCST=S132,P=0
```

The current displacement of Q from initial Q is zero; the logical code segment to which control returns, should you exit from the program other than through TERMINATE', is system segment library segment 132; the P register is zero, the initial setting for the start of the entire program.

If you enter T in a subroutine, the display shows the stack markers for the currently executing routine at the top of the list, followed in turn by each nested routine back to the main program. In this case, the main program T display follows that of the subroutine. For example:

```
?T
Q-0      ,LCST=0      ,P=272 (SUBROUT marker)
Q-21     ,LCST=S132,P=0 (MAIN marker)
```

In the subroutine, the displacement from the current Q is zero, the logical code segment to which control returns should you exit from the program other than through TERMINATE', is system segment library segment 132; the P register is zero, the initial setting for the entire program. The display for the main program is unchanged except for the displacement of initial Q which is shown as 21 locations less than the current Q.

T traces only the subroutines that have led to execution of the current routine, displaying one line for each. Thus T displays only one line for the main program; and in the sample FORTRAN program, a maximum of two lines are displayed by T. If, however, this subroutine had called another, the T executed in a breakpoint to that routine would display three lines.

HOW TO USE THE DEBUG INTRINSIC

In order to use the DEBUG intrinsic in a FORTRAN program, you should insert in your program declarations, the following statement:

```
SYSTEM INTRINSIC DEBUG
```

This defines the intrinsic to the program and allows you to make calls to DEBUG. Then all you need to do is insert a call to DEBUG at the beginning of your program as the first executable statement. When this call is executed, it places the program in DEBUG mode exactly as if you had run the program with the DEBUG parameter in the :RUN command. *Note that the intrinsic is ignored if the program is executed in a batch job.*

Figure 3-6 shows the compilation listing of the FORTRAN program DBTEST. The program is identical to DBTEST1 used in the previous discussion except for the call to DEBUG and its declaration. (Refer to items 1 and 2.) The new statement increases all subsequent locations by 1 so that the instruction at label 10 is now at code offset 13, and the instruction at label 20 is at code offset 76. Otherwise, the compilation listings for the two programs are identical.

Figure 3-7 shows the PMAP listing for program DBPROG (the prepared program file resulting from preparation of DBTEST). This map is identical to the map for DBPROG1 (see figure 3-2) except that the intrinsic DEBUG requires a new STT entry, and the STT entry for 'TERMINATE' has been increased to compensate.

RUNNING WITH THE DEBUG INTRINSIC

When the call to DEBUG is executed, the program pauses at the next location. This break is equivalent to the break at the location of the call to DEBUG. DEBUG displays the message:

```
*DEBUG* s.nnn      (where s is the segment number; nnn the breakpoint)
```

The DEBUG prompt, a question mark, is then issued and you can respond with any of the DEBUG commands exactly as if you had run the program with the DEBUG parameter, as described earlier in this section.

:FORTRAN DBTEST

PAGE 0001 HP32102B.00.0

```
00001000 $CONTROL MAP,LABEL,LOCATION,USLIMIT
00012 00002000 PROGRAM MAIN
00012 00002100 SYSTEM INTRINSIC DEBUG ← ①
00012 00003000 CHARACTER*8 C,C1,C2(3)
00012 00004000 COMMON I4,I5
00012 00006000 DATA C/"DBTEST1"/
00012 00007000 DATA C1/"MAINPROG"/,C2/"MONE","MTWO","MTHREE"/
00012 00008000 DATA I6/10000/,I7/700/,I8/800/
00012 00008100 CALL DEBUG ← ②
③ → 00013 00009000 10 DISPLAY C1,C2
00032 00010000 I1 = -1
00034 00011000 I2 = 1
00036 00012000 I3 = 10
00040 00013000 I4 = 100
00042 00014000 I5 = 1000
00044 00015000 DISPLAY "CALLING SUBROUTINE"
④ → 00076 00016000 20 CALL SUBROUT(C1,C2,I1,I2,I3)
00106 00017000 DISPLAY "BACK FROM SUBROUTINE"
00140 00018000 STOP
00141 00019000 END
```

SYMBOL MAP

NAME	TYPE	STRUCTURE	ADDRESS
C	CHARACTER	SIMPLE VAR	Q+ 1,I
C1	CHARACTER	SIMPLE VAR	Q+ 2,I
C2	CHARACTER	ARRAY	Q+ 2,I
DEBUG		SUBROUTINE	
I1	INTEGER	SIMPLE VAR	Q+ 6
I2	INTEGER	SIMPLE VAR	Q+ 7
I3	INTEGER	SIMPLE VAR	Q+ 8
I4	INTEGER	SIMPLE VAR	0 COMMON
I5	INTEGER	SIMPLE VAR	1 COMMON
I6	INTEGER	SIMPLE VAR	Q+ 3,I
I7	INTEGER	SIMPLE VAR	Q+ 4,I
I8	INTEGER	SIMPLE VAR	Q+ 5,I
SUBROUT		SUBROUTINE	

COMMON BLOCKS

NAME	LENGTH
COM'	2

LABEL MAP

STATEMENT LABEL	CODE OFFSET	STATEMENT LABEL	CODE OFFSET	STATEMENT LABEL	CODE OFFSET
10	13	20	76		

Figure 3-6. Sample FORTRAN Program With Call to DEBUG

```

00005 00020000      SUBROUTINE SUBROUT(D1,D2,J1,J2,J3)
00005 00021000      CHARACTER*8 D1,D2(3)
00005 00021100      COMMON J4,J5
00005 00022000      D1 = "SUBROUT"
00021 00023000      D2(1) = "SBONE"
00037 00024000      D2(2) = "SBTWO"
00055 00025000      D2(3) = "SBTHREE"
5 → 00074 00026000      30  DISPLAY D1,D2
00113 00027000      J1 = -9
00115 00028000      J2 = 9
00117 00029000      J3 = 90
00121 00030000      J4 = 900
00123 00031000      J5 = 9000
6 → 00125 00032000      40  DISPLAY "RETURNING TO MAIN PROGRAM"
00164 00033000      RETURN
00165 00034000      END

```

SYMBOL MAP

NAME	TYPE	STRUCTURE	ADDRESS
D1	CHARACTER	SIMPLE VAR	0- 8,I
D2	CHARACTER	ARRAY	0- 7,I
J1	INTEGER	SIMPLE VAR	0- 6,I
J2	INTEGER	SIMPLE VAR	0- 5,I
J3	INTEGER	SIMPLE VAR	0- 4,I
J4	INTEGER	SIMPLE VAR	0 COMMON
J5	INTEGER	SIMPLE VAR	1 COMMON
SUBROUT		SUBROUTINE	

COMMON BLOCKS

NAME	LENGTH
COM'	2

LABEL MAP

STATEMENT LABEL	CODE OFFSET	STATEMENT LABEL	CODE OFFSET	STATEMENT LABEL	CODE OFFSET
30	74	40	125		

```

****      GLOBAL STATISTICS      ****
**** NO ERRORS, NO WARNINGS ****
TOTAL COMPILATION TIME 0:00:04
TOTAL ELAPSED TIME      0:06:22

```

END OF COMPILE

Figure 3-6. Sample FORTRAN Program With Call to DEBUG (Continued)


```

:PREP $OLDPASS,DBPROG;PMAP
PROGRAM FILE DBPROG.PUB.TECHPUBS
COMMON ARRAY ALLOCATION

NAME          ADR  LEN
COM'          31   2

SEG'          0
NAME          STT  CODE ENTRY SEG
SUBROUT      1    0    0    ?
TFORM'      3    0    0    ?
FMTINIT'    4    0    0    ?
BLANKFILL'  5    0    0    ?
SIO'        6    0    0    ?
ASIO'       7    0    0    ?
MAIN        2   165  172  ?
DEBUG      10   165  172  ?
TERMINATE'  11   165  172  ?
SEGMENT LENGTH          340

PRIMARY DB      2   INITIAL STACK      1440  CAPABILITY      600
SECONDARY DB   33   INITIAL DL          0    TOTAL CODE      340
TOTAL DB      35   MAXIMUM DATA        ?    TOTAL RECORDS   6
ELAPSED TIME  00:01:02.274              PROCESSOR TIME  00:00.417

END OF PREPARE

```

Figure 3-7. PMAP Listing For FORTRAN Program With DEBUG Intrinsic

You can determine where the first break occurs in your program by adding the location of the first instruction after the DEBUG call (item 3, figure 3-6) to the start of code for the main program (item 1, figure 3-7). In the sample program, the first break will occur at a location determined as follows:

$$13 \text{ (octal)} + 165 \text{ (octal)} = 200 \text{ (octal)}$$

(Refer to figure 3-8 for an illustration of running a program containing a call to DEBUG.) In this example, all subsequent breakpoints are set when the program pauses for the first time because of the call to DEBUG. Since this break is equivalent to a break at label 10, only the breakpoints at labels 20, 30, and 40 are set at this time. Otherwise, the breakpoints are the same as those set in figure 3-3, either initially or with the R command. Figure 3-8 simply uses the D command to display items at each of the four breakpoints. Since there is no difference in the use of DEBUG commands when a program is run with the DEBUG parameter or contains a call to DEBUG internally, no attempt is made to illustrate DEBUG commands in this example. Rather it illustrates the program flow at each of the breakpoints as indicated in the following key to the numbered items.

Key to items in figure 3-8 are:

- 1 Location of first breakpoint at label 10 (165+ 13)
- 2 Set three more breakpoints at labels 20, 30, and 40
- 3 Display I6, I7, and I8 set by DATA statement

- 4 Display 8-character item C set by DATA statement
- 5 Display 8-character item C1 set by DATA statement
- 6 Display 3 elements of array C2 set by DATA statement
- 7 Location of second breakpoint at label 20 (165+ 76)
- 8 Display items I1, I2, I3 set between labels 10 and 20
- 9 Display items I4, and I5 set between labels 10 and 20
- 10 Location of third breakpoint at label 30 in SUBROUT
- 11 Display 8-character item D1 set prior to label 30 in SUBROUT
- 12 Display 3 elements of array D2 set prior to label 30
- 13 Display items J1, J2, and J3 passed from program MAIN
- 14 Display items J4 and J5, items common to I4 and I5 set in MAIN
- 15 Location of last breakpoint at label 40 in SUBROUT
- 16 Display items J1, J2, and J3 set between labels 30 and 40
- 17 Display items J4 and J5 set between labels 30 and 40

```

:RUN DBPROG
*DEBUG* 0.200 ← 1
?B 165+76,74,125 ← 2
?D Q+3,3,I
DB+26      +10000 +00700 +00800 ← 3
?D Q+1
Q+1      000004
?D 4/2,4,A
DB+2      DBTEST1 ← 4
?D Q+2
Q+2      000014
?D 14/2,4,A
DB+6      MAINPROG ← 5
?D 14/2+4,4,A
DB+12     MONE
?D 14/2+4,#12,A
DB+12     MONE      MTWO } ← 6
DB+22     MTHREE
?R
MAINPROG MONE      MTWO      MTHREE
CALLING SUBROUTINE
*BREAK* 0.263 ← 7
?D Q+6,3,I
Q+6      -00001 +00001 +00010 ← 8
?D 31,2,I
DB+31     +00100 +01000 ← 9
?R

```

Figure 3-8. DEBUG Example From Program With DEBUG Intrinsic

```

*BREAK* 0.74 ← (10)
?D 0-7
Q-7      000014
?D 14/2,4,A
DB+6     SUBROUT ← (11)
?D Q-#8
Q-10     000014
?D 14/2+4,#12,A
DB+12    SBONE   SBTWO } ← (12)
DB+22    SBTHREE
?D Q-6:,3,I
DB+47    -00001 +00001 +00010 ← (13)
?D 31,2,I
DB+31    +00100 +01000 ← (14)
?R

SUBROUT  SBONE   SBTWO   SBTHREE

*BREAK* 0.125 ← (15)
?D Q-6:,3,I
DB+47    -00009 +00009 +00090 ← (16)
?D 31,2,I
DB+31    +00900 +09000 ← (17)
?R

RETURNING TO MAIN PROGRAM
BACK FROM SUBROUTINE

END OF PROGRAM

```

Figure 3-8. DEBUG Example From Program With DEBUG Intrinsic (Continued)

USING DEBUG WITH PRIVILEGED MODE CAPABILITY

Additional commands and capabilities are available when you use DEBUG in privileged mode. Mainly, privileged mode allows you to display and modify data up to the stack limit, Z, to display and establish breakpoints in system code as well as in your own program, and to display disc sector contents.

Four additional commands are allowed in privileged mode; also, most commands have enhanced capabilities. (Refer to table 3-1 for a list of the new commands and the extra capabilities for the standard commands.) In addition to the extra DEBUG capabilities, a user with privileged mode can use the MPE :DEBUG command to debug the Command Interpreter. (Refer to the *MPE Commands Reference Manual* for a discussion of this command.)

WARNING

The normal checks and limitations that apply to the standard users in MPE are bypassed in privileged mode. It is possible for a privileged mode program to destroy system integrity, including the MPE operating system software itself. Hewlett-Packard cannot be responsible for system integrity when programs written by users operate in privileged mode.

Use of DEBUG in privileged mode is essentially identical to use with standard capabilities. A minor difference is that the message printed at the initial entry into DEBUG and at each breakpoint includes the word PRIV to indicate that you are operating in privileged mode. For example, at the first entry point, the following message is printed:

DEBUG PRIV 0.170 (Enter DEBUG in privileged mode at location 170.)

At a subsequent breakpoint, the following might be printed:

BREAK PRIV 2.360 (Break in privileged mode in segment 2, location 360.)

The following discussion of using DEBUG in privileged mode gives only a summary of the additional capabilities within the various functional areas.

Table 3-1. Privileged Mode Command Capabilities

PRIVILEGED COMMAND	CAPABILITY
A { S P }	Establish breakpoint mode as private (P) or system global (S).
DV [<i>ldev</i>] + <i>startsector</i> [<i>,count</i>] [<i>,mode</i>]	Display specified number of sectors (<i>count</i>) from specified device (<i>ldev</i>) or system disc, from specified sector number (<i>startsector</i>), in octal, decimal, or ASCII (<i>mode</i>).
F { CO DA } <i>segnum</i>	Freeze code (CO) or data (DA) in the specified segment (<i>segnum</i>) so that it is not swapped.
U { CO DA } <i>segnum</i>	Unfreeze frozen code (CO) or data (DA) in specified segment (<i>segnum</i>).
STANDARD COMMAND	PRIVILEGED CAPABILITY
B [S A] <i>segment.</i> . . .	Segment location is extended to include system segmented library (S) or absolute code segment (A).
C [S A] <i>segment.</i> . . .	Segment location is extended to include system segmented library (S) or absolute code segment (A).
D [A SY CO DA DX EA] . . .	Display base extensions allow display of absolute relative (A), system global relative (SY), code segment relative (CO), data segment relative (DA), current absolute DB relative (DX), and extended absolute address (EA).
DR	When all registers displayed, display includes PCB (process control block index), CST (absolute code segment index), STAK (stack segment index), DST (extra data segment index). Also, if operating in absolute mode, DX (absolute location of DB register) and EA (current block number).
L [<i>ldev</i>]	Switches display to specified logical device number (<i>ldev</i>).
M [A SY DA DX EA] . . .	The modify base (<i>modbase</i>) extensions allow modification of absolute relative (A), system global relative (SY), data segment relative (DA). Also, in absolute mode, current absolute DB relative (DX) and extended absolute address (EA).
MR ,ST	All ST bits, rather than only bits 2-7 can be modified.
R[[S] <i>segment.</i>] . . .	Segment location is extended to include system segmented library (S).
T	Trace is extended to show absolute code segment index (CST).

SETTING BREAKPOINTS IN PRIVILEGED MODE

Two modes are defined for breakpoints:

- P private breakpoint mode (restricted to your own program)
- S system breakpoint mode (includes entire MPE system)

When a system breakpoint is established, it is global; that is, any program will break at that point. When a breakpoint is established in private mode, only the program in which the breakpoint is established breaks.

The A command can be used to specify that subsequent breakpoints are private or global. Use AS to set global breakpoints; use AP to return to private mode, the default. For example:

```
*DEBUG* PRIV 0.270
?AS
?B 5.301           (The first program to execute relative location 301, segment 5,
                    will break.)
    .
    .
*BREAK* SYS.PRIV 5.301 (In system global mode, the mode is indicated by SYS.PRIV in
                    the break message.)
    .
    .
?AP              (Subsequent breakpoints only apply to your own program.)
```

To set breakpoints in the system segmented library, follow the B or R commands by S. For example:

```
?BS 3.160+ 10,160+ 57 (Breakpoints are set at locations 160+ 10 and 160+ 57 of seg-
                    ment 3 of the system segmented library.)

?RS 3.324           (Resume execution and break at location 324 in segment 3 of
                    the system segmented library.)
```

To set a breakpoint in absolute code, follow the B command with A. For example:

```
?BA 52.173         (Breakpoint is set at code segment absolute 52, location 173.)
```

SWITCHING DISPLAY TO LINE PRINTER IN PRIVILEGED MODE

In privileged mode, you can specify a logical device number of the device to which you want the display sent. For example, you can send output to a line printer equated to logical device 6 as follows:

```
?L6
```

When a logical device number rather than a file reference is specified, the line printer is preempted immediately for the output. This means that if other users are printing to the same device, your output can be interspersed with theirs. To avoid this, always check to insure that you are the sole user of a line printer before using this form of the L command.

NOTE

The logical device must always be equated to a line printer.

DISPLAYING CODE, DATA, AND DISC SEGMENTS IN PRIVILEGED MODE

The D command can be followed by a letter indicator in order to establish a display base other than the stack relative bases DB, DL, Q, and S, or the code relative bases PB, P, and PL. The privileged mode bases allow you to display absolute code locations in bank zero or in other banks, and to display data relative to the absolute location of DB. For example, in order to display code starting at an absolute location in bank 2, enter:

?D EA2+ 121144+ 372,20 (Display 16 words at location 372 relative to absolute location 121144 in bank 2.)

DEBUG returns the octal listing listing, 8 words per line as follows:

```
EA2+ 121536 00000 00103 00105 00000 00000 00314 00372 00402
EA2+ 121546 00000 00000 00000 00000 00000 00000 00000 00000
```

The privileged mode user can also display code or data relative to the system base, to the base of any code segment or the base of any data segment. To display at a location relative to the system base, enter:

?D SY 161 (Displays location 161 relative to the system base.)

When displaying relative to the base of a code or data segment, any offset within the segment is separated from the relative segment number by a plus (+). For example:

?D DA 5+ 74,10,I (Displays 8 words of data in decimal starting at location 74 of data segment 5.)

?D CO 301+ 20,20 (Display 16 words of code starting at location 20 of user segment 301.)

?D CO(10+ 6)+ 10, # 64 (Display 64 words of code starting at location 10 of system segment 16; since the segment number is an expression, it must be enclosed in parentheses.)

The contents of any segment of disc can be displayed with the DEBUG command DV. If a disc logical device number is not specified, the system disc (*ldev*= 1) is assumed. The starting sector address must be specified and, if more than one, the number of sectors to be displayed must be specified. Portions of a sector cannot be displayed. For example:

?DV 2+ 32715,3 (Display 3 sectors starting at sector 32715 on logical device 2.)

MODIFYING DATA IN PRIVILEGED MODE

Although DEBUG cannot be used to modify code directly, in privileged mode code can be modified if its absolute address is known. All the privileged mode extensions used with the D command, except CO indicating a code segment, can be used with the M command. You can modify absolute code locations in any bank, and modify data relative to the absolute location of DB. You can also modify data relative to the system base or relative to the base of any data segment. If the data segment number is an expression, it must be enclosed in parentheses. For example:


?M DA(62+ 20)+ 10,5,I (Display 5 words of decimal data at location 10 in data segment 102 (octal).)

DISPLAY OR MODIFY REGISTERS IN PRIVILEGED MODE

The same commands, DR and MR, are used to display or modify registers in privileged mode. The difference is that more registers are displayed in privileged mode and all ST bits, rather than just bits 2 through 7, can be modified.

DISPLAY REGISTERS. To illustrate, enter the command DR:

```
?DR
ST= 60301,X= 0,DL= 177602,Q= 4207,S= 4207,Z= 16013,P= 0
1= 0,2= 0,3= 0,4= 0
PCB= 27,CST= 301,LCST= 0,STAK= 214
```



privileged mode only

where:

PCB is the index to the process control block.
CST is the index to your absolute code segment.
STAK is the index to your stack segment.

If operating in absolute DB mode, two additional registers are displayed:

DX absolute address of DB.
EA current bank number.

If you set breakpoints in the system library (BS command), the logical code segment number (LCST) is preceded by the letter S.

If you are using extra data segments, one more register is displayed:

DST index to current extra data segment.

MODIFY REGISTERS. If you enter the MR command, you will be prompted to change the same registers as if you entered the command in non-privileged mode. The only difference is that you can change bits 0 and 1, and bits 8 through 15 of the ST register.

FREEZING/UNFREEZING SEGMENTS IN PRIVILEGED MODE

During DEBUG operations it is occasionally necessary to freeze your code and data so that swapping does not alter the current locations. This is important particularly when you want to determine absolute locations. The F command is used to freeze code (FCO) or data (FDA). The U command unfreezes code (UCO) or data (UDA).

Suppose you want to freeze your code and data segments, first you can use DR to determine the segment numbers, then freeze those segments:

```
?DR,CST,STAK
CST= 301,STAK= 214
?FCO 301
?FDA 214
```

You can then display the contents of the first 8 words (absolute address 0-7) of these segments as follows:

?DCO 301 ,10

?DDA 214 ,10

You can then unfreeze these segments and continue by entering:

?UCO 301

?UDA 214

HOW TO USE THE STACK DUMP FACILITY

SECTION

IV

The commands and intrinsics that comprise the stack dump facility of DEBUG allow you to dump all or portions of your data stack, provide a trace and analysis of the stack, and in interactive mode, allow you to use DEBUG commands to trace and possibly correct errors.

The STACKDUMP intrinsic allows you to dump the stack at any time and send the dump to a file. The SETDUMP intrinsic or command provides a stack dump or stack marker analysis automatically in case your program terminates abnormally (aborts).

The operation of SETDUMP differs depending on whether the program is executed interactively or is part of a batch job. If the :SETDUMP command is entered during an interactive session or a program containing a SETDUMP intrinsic is executed interactively, the stack is not dumped. When the program aborts, you are given a trace of the stack markers and placed in DEBUG mode so that you can enter any of the DEBUG commands described in Section III. If :SETDUMP is entered as a batch command in a job or if a program containing the SETDUMP intrinsic is executed in a batch job, a trace of the stack markers is provided when the program terminates and, optionally, a dump is provided of all or portions of the stack.

Unless specified otherwise, the STACKDUMP intrinsic provides a trace of the stack markers in addition to the stack dump.

The :RESETDUMP command or RESETDUMP intrinsic can be used to cancel the effect of the :SETDUMP command or SETDUMP intrinsic, respectively.

STACK TRACE AND ANALYSIS

The stack trace and analysis provided automatically by SETDUMP and optionally by STACKDUMP can be very useful. The values provided by the stack trace and analysis are illustrated in figure 4-1.

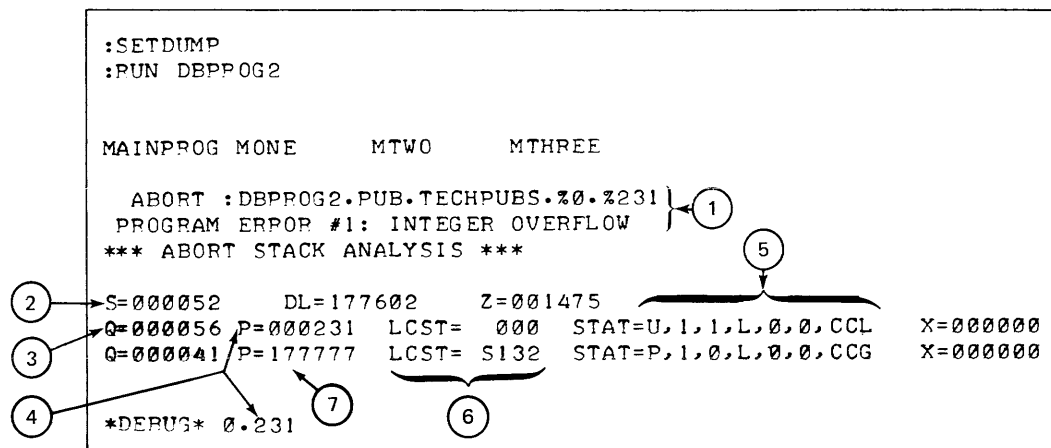


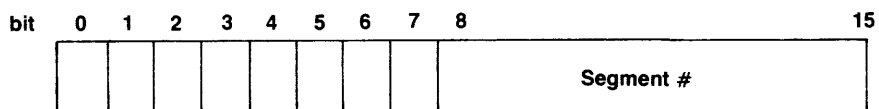
Figure 4-1. Sample Stack Trace and Analysis

The key items in figure 4-1 are:

- 1 Abort message indicating where the abort occurred (sector 0, relative location 231) and the type of error (integer overflow).
- 2 Location of top of stack when the error occurred.
- 3 Location of Q *after* the error and *after* the four-word stack marker has been placed at the top of the stack; note that Q is four words greater than S.
- 4 Value of program counter following abort; P-1 is location of statement that caused the abort.
- 5 Status word at the time of the error; the first line shows the status of the program unit that caused the program to abort.
The second line provides the status of system area to which control normally returns at an abort.
- 6 Logical Code Segment number assigned to the program by the segmenter at the time of PREP. The S preceding the LCST number indicates it is a system code segment.
- 7 Since code segment S132 has not been entered yet, the program counter has no valid value. This is indicated by printing -1, (177777).

This information is not only displayed for your use, but is placed at the top of the stack as the four-word stack marker.

The status word (ST in the display) has the following format:



- | | |
|---|--|
| <p>bit 0 = 1 if program is privileged
0 if program is in user mode</p> <p>bit 1 = 1 if external interrupts are enabled
0 if not</p> <p>bit 2 = 1 if user traps enabled
0 if not</p> <p>bit 3 = 1 if right stack operation pending
0 if left stack operation pending</p> | <p>bit 4 = 1 if overflow bit set (not set if user traps enabled)
0 if not</p> <p>bit 5 = 1 if carry bit set
0 if not</p> <p>bits 6,7 = 01 if CCL
10 if CCE
00 if CCG</p> |
|---|--|

USING SETDUMP AND RESETDUMP

Whether entered as a command or included in your program as a call to the SETDUMP intrinsic, SETDUMP only executes when and if your program aborts.

Entered as a command, :SETDUMP remains active for the life of the session or job. If you want to cancel SETDUMP before these points are reached, you can enter the :RESETDUMP command from your terminal or from within a batch job.

Called as an intrinsic, SETDUMP remains in effect for the rest of the program unless specifically cancelled by a call to the RESETDUMP intrinsic.

As a command or intrinsic, SETDUMP affects not only the current process but also any son processes created after SETDUMP is executed. RESETDUMP, on the other hand, affects only the current process and must be specified separately for any son processes for which SETDUMP enabled the stack analysis facility.

INTERACTIVE USE OF SETDUMP

If you enter the :SETDUMP command before running your program or if the SETDUMP intrinsic is included in a program run interactively, control transfers to DEBUG if the program terminates abnormally. A stack trace analysis is displayed and then the program enters DEBUG mode so that you can enter any of the DEBUG commands described in Section III. Any SETDUMP parameters are ignored.

USING :SETDUMP COMMAND INTERACTIVELY. To illustrate the use of :SETDUMP in interactive mode, suppose you are running program DBPROG2 for the first time. You can precede the :RUN command by a :SETDUMP command and, if the program aborts, you can then use DEBUG to determine the cause and possibly correct it. In the example in figure 4-1, DBPROG2 terminates with an integer overflow.

The status of the aborted program in the example, shows that the user traps are set (bit 2= 1). As a result, the overflow bit (bit 4) is not set even though an overflow occurred. This is a result of the way the interrupt system operates.

Following the stack analysis, the DEBUG prompt is issued. In response, you can perform any of the standard DEBUG operations. For instance, if you simply want to bypass the erroneous statement, you can increment the P register and continue. Or you might want to terminate by entering E@. You can also display the contents of the instruction where the program terminated and try to determine the cause of the error. Since the error in this case is an integer overflow, you could look at the contents of the last three or four words in the stack when the error occurred. Usually, you will also look at your compilation listing. For this sample program, the listing of program MAIN is shown in figure 4-2; the listing for SUBROUT and the PMAP listing from preparation are identical to those shown in figure 3-1 in Section III since only program MAIN has been changed.

:FORTRAN DBTEST2

PAGE 0001 HP32102B.00.0

```
00001000 $CONTROL MAP,LABEL,LOCATION,USLINIT
00012 00002000      PROGRAM MAIN
00012 00003000      CHARACTER*8 C,C1,C2(3)
00012 00004000      COMMON I4,I5
00012 00006000      DATA C/"DBTEST1"/
00012 00007000      DATA C1/"MAINPROG"/,C2/"MONE","MTWO","MTHREE"/
00012 00008000      DATA I6/10000/,I7/700/,I8/800/
00012 00009000      10  DISPLAY C1,C2
00031 00010000      I1 = -1
00033 00011000      I2 = 1
00035 00012000      I3 = 10
00037 00013000      I4 = 100
00041 00014000      I5 = 1000
00043 00014100      I5=32767+I2
00046 00015000      DISPLAY "CALLING SUBROUTINE"
00101 00016000      20  CALL SUBROUT(C1,C2,I1,I2,I3)
00111 00017000      DISPLAY "BACK FROM SUBROUTINE"
00143 00018000      STOP
00144 00019000      END
```

SYMBOL MAP

NAME	TYPE	STRUCTURE	ADDRESS
C	CHARACTER	SIMPLE VAR	Q+ 1,I
C1	CHARACTER	SIMPLE VAR	Q+ 2,I
C2	CHARACTER	ARRAY	Q+ 2,I
I1	INTEGER	SIMPLE VAR	Q+ 6
I2	INTEGER	SIMPLE VAR	Q+ 7
I3	INTEGER	SIMPLE VAR	Q+ 8
I4	INTEGER	SIMPLE VAR	0 COMMON
I5	INTEGER	SIMPLE VAR	1 COMMON
I6	INTEGER	SIMPLE VAR	Q+ 3,I
I7	INTEGER	SIMPLE VAR	Q+ 4,I
I8	INTEGER	SIMPLE VAR	Q+ 5,I
SUBROUT	INTEGER	SUBROUTINE	

COMMON BLOCKS

NAME	LENGTH
COM*	2

LABEL MAP

STATEMENT LABEL	CODE OFFSET	STATEMENT LABEL	CODE OFFSET	STATEMENT LABEL	CODE OFFSET
10	12	20	101		

Figure 4-2. Sample FORTRAN Program With Error

Note item 1 in figure 4-2. This location relative to the start of code location at 165 gives the PB-relative location 230. The program halted with the P register at 231 so this is the instruction containing the error. If you want to continue ignoring this error, enter:

```
?MR,P  
P= 231 := 233  
?R
```

The program continues execution with the statement at location 46 + 165. Assuming there are no further errors, it will run until normal termination.

You could also terminate the program by entering:

```
?E@
```

The program terminates with the message END OF PROGRAM and returns control to the operating system.

If you want to display the contents of the statement causing the error, enter:

```
?DP- 1,3
```

The three octal words of the instruction will be displayed.

You can also display the contents of the stack at the time of the abort. The stack analysis showed that S, at this time, was at DB-relative location 52. Thus, to display the last three words in the stack at that time, enter:

```
?DDB+ 52- 3,3
```

Depending on what you find, you may want to remove the values currently in the stack. You can do this with the E command as follows:

```
?E2      Remove 2 words from top of stack and resume.  
?R
```

Since the particular problems causing a program to abort vary widely, only suggestions can be given here of the particular action to take from among the many options provided through the DEBUG commands.

USING SETDUMP INTRINSIC INTERACTIVELY. You may include a call to the SETDUMP intrinsic anywhere within your program. If you want to transfer to DEBUG in case of an abort anywhere in the program, you should place the call at the beginning. If you only want DEBUG in case of an abort in a specific section of the program, place the intrinsic call at the beginning of this section and a call to RESETDUMP at the end.

Used interactively, no parameters need be included in the intrinsic call since they will be ignored; control transfers to the DEBUG program exactly as if the :SETDUMP command had been entered in an interactive session as described above. If, however, the program may be executed in batch mode, you should include parameters to specify the portion of the stack you want dumped in case of an abort in a batch execution. These parameters are described below under the heading USING SETDUMP INTRINSIC IN A JOB.

Only bits 12 through 15 have meaning. They are interpreted as follows:

bit DB = 1 dump DL to Q initial
ST = 1 dump Q initial to S
QS = 1 dump Q-63 to S (ignored if bit 14 = 1)
AS = 1 dump ASCII as well as octal values

The remaining values are zero and, if the entire parameter is zero, only the stack trace and analysis is provided.

Suppose you want to call the system intrinsic SETDUMP from a FORTRAN program and dump the entire stack with ASCII equivalence, include the following call in your program:

```
CALL SETDUMP(%13)      (Dump DL through S with ASCII)
```

Bits 12, 14, and 15 are set by this parameter, octal 13.

To simply dump the stack with no ASCII conversion, use the call:

```
CALL SETDUMP(3)        (Dump DL through S, no ASCII)
```

If you want to dump Q-63 to S, use the call:

```
CALL SETDUMP(4)        (Dump Q-63 through S, no ASCII)
```

Set the entire parameter value to zero if you only want a stack trace and analysis upon abort:

```
CALL SETDUMP(0)        (Stack trace and analysis only)
```

The stack trace and analysis is illustrated in figure 4-1. Refer to figure 4-4 for a sample of a stack dump.

TERMINATING WITH RESETDUMP

:SETDUMP is terminated automatically when you log off in an interactive session. If, however, you want to continue in the session without using :SETDUMP, you must specifically terminate with the :RESETDUMP command by entering:

```
:RESETDUMP
```

Note that you must return to MPE control before entering :RESETDUMP; it cannot be specified during program execution.

Specified in a batch job, the :SETDUMP command remains in effect until the end of the job unless you specifically terminate with :RESETDUMP.

When the SETDUMP intrinsic is used in a program, it remains in effect during execution of that program. You may set limits to its range by calling the RESETDUMP intrinsic at some point in your program subsequent to SETDUMP. Thus, you can request a dump and stack analysis in case of an abort in selected areas of your program.

USING STACKDUMP

The `STACKDUMP` intrinsic provides a dump of your stack from three points in the stack: counting up from `DB`, counting down from `S`, and relative to initial `Q` between `Q-63` and `S`. The areas to be dumped are, thus, similar to the areas dumped by `SETDUMP` in case of a program abort. The difference is that `STACKDUMP` provides a dump in an executing program, not following an abort, and the dump is taken at the point `STACKDUMP` is specified.

HOW TO CALL STACKDUMP

The parameters used for `STACKDUMP` are defined in section II. If no parameters are specified no dump is taken so it is usual to set parameters.

The first parameter specifies the filename where the information is to be dumped. If this parameter is omitted, the dump is sent to the standard list device, for a job, the line printer and in a session, your terminal.

If you do specify a filename and plan to collect your dump from a pile of line printer output, you may want to include the second parameter that is an integer printed on the dump to identify it. Also, if you are making several calls to `STACKDUMP` within a program, you can use this parameter to differentiate between the resulting dumps. It is also useful if you have several consecutive versions of the program in order to document the changes.

The third parameter can be used to suppress the ASCII conversion that is otherwise printed automatically beside the octal dump. It can also be used to suppress the trace and stack analysis normally provided with the dump.

The fourth parameter allows you to select the portion of the stack you want dumped. Actually, you can dump the entire stack from `DB` to `S` by specifying a positive count relative to `DB`. If the count is greater than the number of words in your stack, the dump terminates when it has displayed the top of the stack.

Figure 4-3 illustrates a sample FORTRAN program that calls `STACKDUMP` in order to dump the stack in an upward direction relative to `DB`. Since `STACKDUMP` is a system intrinsic, it should be declared in a `SYSTEM INTRINSIC` declaration (see item 1, figure 4-3). The call itself is shown in item 6, figure 4-3. The first parameter specifies the file to which the dump is sent, `STDUMP`, and the fourth intrinsic specifies the name of the array containing the area to be dumped. The filename is contained in a byte array declared in the program as an 8-character array (see item 2, figure 4-3). The contents of this array, the filename, is specified in a `DATA` statement (see item 3, figure 4-3).

The array, `S`, containing the *selec* parameter value is declared in item 4, figure 4-3 to have three double-word values. These values for `S` are defined in a `DATA` statement (see item 5, figure 4-3). Only the first two double words are given values; the second word contains the terminator for the array: zero in the first word and `-1` in the second. This value is specified as an octal integer `177777`. Double word constants in FORTRAN must be terminated by the letter `J` to indicate to the compiler that they contain 32 bits. Only the second word need be specified; the first word is set to all zeros automatically. The first double word of the array specifies the count and that the count is to be relative to `DB+0`. When specifying a count relative to `DB`, the mode is zero. For this reason, only the count need be specified; zero in the first word is relative to `DB` since the initial zeros in the second word indicate that the mode is `DB` relative.

:FORTRAN DBTESTST

PAGE 0001 HP32102B.00.0

```
00001000 $CONTROL MAP,LABEL,LOCATION,USLINIT
00015 00002000 PROGRAM MAIN
00015 00003000 SYSTEM INTRINSIC STACKDUMP ← 1
00015 00004000 CHARACTER*8 C,C1,C2(3)
00015 00005000 CHARACTER*8 STDUMP ← 2
00015 00006000 INTEGER*4 S(3) ← 4
00015 00007000 COMMON I4,I5
00015 00008000 DATA S/100J,%177777J/ ← 5
00015 00009000 DATA C/"DBTEST1"/
00015 00010000 DATA C1/"MAINPROG"/,C2/"MONE","MTWO","MTHREE"/
00015 00011000 DATA STDUMP/"STDUMP"/ ← 3
00015 00012000 DATA I6/10000/,I7/700/,I8/800/
00015 00013000 10 DISPLAY C1,C2
00034 00014000 I1 = -1
00036 00015000 I2 = 1
00040 00016000 I3 = 10
00042 00017000 I4 = 100
00044 00018000 I5 = 1000
00046 00019000 CALL STACKDUMP(STDUMP,,S) ← 6
00054 00020000 DISPLAY "CALLING SUBROUTINE"
00106 00021000 20 CALL SUBROUT(C1,C2,I1,I2,I3)
00116 00022000 DISPLAY "BACK FROM SUBROUTINE"
00150 00023000 STOP
00151 00024000 END
```

SYMBOL MAP

NAME	TYPE	STRUCTURE	ADDRESS
C	CHARACTER	SIMPLE VAR	Q+ 2,I
C1	CHARACTER	SIMPLE VAR	Q+ 3,I
C2	CHARACTER	ARRAY	Q+ 3,I
I1	INTEGER	SIMPLE VAR	Q+ 8 ← 7
I2	INTEGER	SIMPLE VAR	Q+ 9
I3	INTEGER	SIMPLE VAR	Q+ 10
I4	INTEGER	SIMPLE VAR	0 COMMON
I5	INTEGER	SIMPLE VAR	1 COMMON
I6	INTEGER	SIMPLE VAR	Q+ 5,I
I7	INTEGER	SIMPLE VAR	Q+ 6,I
I8	INTEGER	SIMPLE VAR	Q+ 7,I
S	INTEGER*4	ARRAY	Q+ 1,I
STACKDUMP		SUBROUTINE	
STDUMP	CHARACTER	SIMPLE VAR	Q+ 4,I
SUBROUT		SUBROUTINE	

COMMON BLOCKS

NAME	LENGTH
COM'	2

LABEL MAP

STATEMENT LABEL	CODE OFFSET	STATEMENT LABEL	CODE OFFSET	STATEMENT LABEL	CODE OFFSET
10	15	20	106		

Figure 4-3. Sample FORTRAN Program Using STACKDUMP

If you want to select a dump relative to S, you must specify the first three bits of the second word as octal 7 (bits 0 through 2=111). To do this, use a composite format as shown below:

DATA S/%[16/0,3/7,13/100]J,%177777J/
S-0 mode = 111 count = 100

The brackets allow you to indicate bit values within the word. As with the previous example, the terminator is specified as a double-word octal value 177777.

ANALYZING THE STACK DUMP

When the program shown in figure 4-3 is executed, a stack dump is taken just prior to location 54 in the listing. (The starting code location for this program unit MAIN is the same as that shown in the PMAP in figure 3-2.) The trace and stack analysis shows the current program location to be at 240, or 165+53 (octal). (Refer to figure 4-4 for the printout resulting from execution of STACKDUMP.)

Since DB+100 was out of the bounds of the program's stack, the dump stopped at location 72 (the location of S) and a message is printed. The value of Q prior to the stack dump is shown in the second line of the listing. The value of Q after the stack dump is four words greater than the top of the stack to allow four words for the stack markers. This Q value is shown in the first line of the trace and stack analysis.

Figure 4-4 shows how to determine Q and S from the trace and stack analysis. Once you know where these values are and that the dump starts with DB+0, you can locate all the values shown in the symbol map in figure 4-3. For example, look at the location Q+8 in the symbol map (item 7, figure 4-3). This location contains the integer simple variable I1 that has been set to -1 in the program. Then look at the location Q+8 in the stack dump and you will see that it correctly contains the value 177777 (octal)

The character values in C, C1, and C2 are found indirectly through the byte addresses stored in Q+2 and Q+3. Note that Q+2 contains the value 20 and Q+3 the value 30. Since these are byte addresses, they indicate that C starts in the 20th byte or word 10 and that C1, which is word zero of the array C2, starts in the 30th byte or word 14. *Remember that all numbers shown in the dump are octal.*

All other program values set when the dump was taken can be ascertained in this way.

NOTE

Before running a program in which a call to STACKDUMP specifies a filename, be sure to equate the file name to a device where you want the dump sent. Otherwise the dump is sent to a null file.

```

:FILE STDUMP=$STDLIST
:RUN DBPROGST

MAINPROG MONE      MTWO      MTHREE

***      STACK DISPLAY      ***

          S=000072      DL=177602      Z=001507
Q=000076 P=000240 LCST= 000 STAT=U,1,1,L,0,0,CCG X=000000
          Q=000053 P=177777 LCST= S132 STAT=P,1,0,L,0,0,CCG X=000000

..DB..
00000      000043 000044 000000 000144      .# .S .. .D
00004      000000 177777 000000 000000      .. .. .. ..
00010      042102 052105 051524 030440      DB TE ST 1
00014      046501 044516 050122 047507      MA IN PR OG
00020      046517 047105 020040 020040      MO NE
00024      046524 053517 020040 020040      MT WO
00030      046524 044122 042505 020040      MT HR EE
00034      051524 042125 046520 020040      ST DU MP
00040      023420 001274 001440 000144      '. .. . .D
00044      001750 003003 177777 000000      .. .. .. ..
00050      000000 000000 140033 000004      .. .. .. ..
00054      000000 000020 000030 000070      .. .. .. .8
00060      000040 000041 000042 177777      . ! ." ..
00064      000001 000012 000070 000110      .. .. .8 .H
00070      000000 000002      ← S-0      .. ..

** AREA OUT OF BOUNDS **

          CALLING SUBROUTINE
SUBROUT SBONE SBTWO SBTHREE
RETURNING TO MAIN PROGRAM
BACK FROM SUBROUTINE

END OF PROGRAM
:

```

Figure 4-4. Sample Stack Dump

DEBUG ERROR MESSAGES

SECTION

V

When an error is detected by DEBUG, it issues a message in the form:

message n

where:

message indicates the type of error.

n indicates the character position in the command where the error was detected.

For example, suppose you enter the following command:

?DQ+ 1:,8,A
SYNTAX 6 message issued by DEBUG

This indicates that there is a syntax error following character position 6.

In this case, it is the decimal value 8 that is in error. To correct, re-enter the command:

?DQ+ 1:,# 8,A

The five possible messages issued by DEBUG are defined in Table 5-1.

Table 5-1. DEBUG Error Messages

MESSAGE	MEANING	CORRECTIVE ACTION
BOUNDS	A specified location is outside the permitted bounds. For example, in non-privileged mode, you referenced a location above S or below DL in the data stack; or, in privileged mode, you referenced code outside the specified segment, or an absolute address outside memory.	Check your command to make sure it is entered correctly. If it is, lower or raise the referenced location. Note that using an indirect address where you want direct can produce a bounds violation.
CHECK	New breakpoint conflicts with established breakpoint. This usually occurs when you specify as new a breakpoint that already exists.	This is a warning only. You may continue. Any new breakpoints will be established, existing breakpoints are not affected. If you wish, check all existing breakpoints with the B@ command.
FULL	The breakpoint table established at system configuration is full.	Probably too many users are currently using DEBUG. Wait and try again. If this message happens consistently, request your system manager to re-configure with a larger breakpoint table.
NO-NO	Invalid information was provided in a command whose syntax is correct. For instance, you specified a logical segment number that does not exist.	Check the information provided in your command. For instance, if you entered B 5.166+70, make sure that you actually have six segments in your code. Correct and continue.
SYNTAX	The command syntax is in error.	Check the syntax and continue.

USING DECOMP

APPENDIX

A

This appendix contains the compilation breakdown provided by the program DECOMP for the sample FORTRAN used in this manual. The decompilation is generated by running program DECOMP as shown below:

```
:RUN DECOMP.PUB.SYS

** HP3000 DECOMPILER 2.0 **

OUTPUT TO TERMINAL(T) OR PRINTER(P)? P
PROGRAM FILE NAME? DBPROG1
NO. OF SEGMENTS: %1
THEY ARE NUMBERED FROM ZERO UPWARDS
ENTER STARTING SEGMENT (PRECEDED BY %) %0

THIS SEG HAS LENGTH OF %340
ENTER STARTING P-VALUE (PRECEDED BY %) %0

END OF PROGRAM
:
```


000107	021010	".	LDI	8	
000110	006000	..	LADD,	NOP	
000111	031007	2.	PCAL	ASIO"	
000112	031003	2.	PCAL	TFORM"	
000113	025011	*.	LDNI	9	
000114	053606	W.	STOR	Q- 006,I	
000115	021011	".	LDI	9	
000116	053605	W.	STOR	Q- 005,I	
000117	021132	"Z	LDI	90	
000120	053604	W.	STOR	Q- 004,I	
000121	040012	@.	LOAD	P+ 012	
000122	053000	V.	STOR	DB+000,I	
000123	040011	@.	LOAD	P+ 011	
000124	053001	V.	STOR	DB+001,I	
000125	000707	..	DZRO,	DZRO	
000126	021002	".	LDI	2	
000127	172003	..	LRA	P+ 003,I	
000130	031004	2.	PCAL	FMTINIT"	
000131	140021	..	BR	P+ 021	
000132	000032	..	NOP	, XCH	
000133	001604	..	DXCH,	INCX	
000134	021450	#(LDXI	40	
000135	051105	RE	STOR	DB+105	
000136	052125	TU	MTBA	P+ 125	
000137	051116	RN	STOR	DB+116	
000140	044516	IN	LOAD	P- 116,X	
000141	043440	G	LOAD	Q+ 040,I	
000142	052117	TU	MTBA	P+ 117	
000143	020115	M	----		
000144	040511	AI	LOAD	P- 111	
000145	047040	N	LOAD	DB+040,I,X	
000146	050122	PR	TBA	P+ 122	
000147	047507	UG	LOAD	Q+ 107,I,X	
000150	051101	RA	STOR	DB+101	
000151	046407	M.	LOAD	P- 007,I,X	
000152	025415	+	LDXN	13	
000153	044401	I.	LOAD	P- 001,X	
000154	011202	..	IXBZ	P+2	
000155	140402	..	BR	P- 002	
000156	021031	".	LDI	25	
000157	171715	..	LRA	S- 015	
000160	010201	..	LSL	1 BIT	
000161	031006	2.	PCAL	SIO"	
000162	035415	7.	SUBS	X0015	
000163	031003	2.	PCAL	TFORM"	
000164	031405	3.	EXIT	X0005	
000165	000004	..	NOP	, INCX	
000166	000014	..	NOP	, DIVL	
000167	000026	..	NOP	, STBX	
000170	000027	..	NOP	, DST	
000171	000030	..	NOP	, DFLT	
000172	034405	9.	LDPN	X0005	
000173	034404	9.	LDPN	X0004	
000174	040403	A.	LOAD	P- 003	
000175	035004	!	ADDS	X0004	
000176	004000	..	DEL	, NOP	
000177	000707	..	DZRO,	DZRO	
000200	021002	".	LDI	2	
000201	170015	..	LRA	P+ 015	
000202	031004	2.	PCAL	FMTINIT"	
000203	041402	C.	LOAD	Q+ 002	
000204	021010	".	LDI	8	
000205	003200	..	XCH	, NOP	
000206	031006	2.	PCAL	SIO"	
000207	021010	".	LDI	8	
000210	021003	".	LDI	3	
000211	041402	C.	LOAD	Q+ 002	
000212	021010	".	LDI	8	
000213	006000	..	LADD,	NOP	
000214	031007	2.	PCAL	ASIO"	
000215	031003	2.	PCAL	TFORM"	
000216	025001	*.	LDNI	1	
000217	051406	S.	STOR	Q+ 006	
000220	021001	".	LDI	1	
000221	051407	S.	STOR	Q+ 007	
000222	021012	".	LDI	10	
000223	051410	S.	STOR	Q+ 010	
000224	021144	".	LDI	100	
000225	053000	V.	STOR	DB+000,I	
000226	040010	@.	LOAD	P+ 010	
000227	053001	V.	STOR	DB+001,I	
000230	000707	..	DZRO,	DZRO	
000231	021002	".	LDI	2	
000232	172003	..	LRA	P+ 003,I	
000233	031004	2.	PCAL	FMTINIT"	
000234	140014	..	BR	P+ 014	

MAIN:

<== MAIN PROG STARTS

Figure A-1. DECOMP Listing (Sheet 2 of 3)


```

000235 000025 .. NOP , TEST
000236 001750 .. CMP , FCMP
000237 041501 CA LOAD Q+ 101
000240 046114 LL LOAD P+ 114,I,X
000241 044516 IN LOAD P- 116,X
000242 043440 G LOAD Q+ 040,I
000243 051525 SU STOR Q+ 125
000244 041122 BR LOAD DB+122
000245 047525 OU LOAD Q+ 125,I,X
000246 052111 TI MTBA P+ 111
000247 047105 NE LOAD DB+105,I,X
000250 025411 +. LDXN 9
000251 044401 I. LOAD P- 001,X
000252 011202 .. IXBZ P+2
000253 140402 .. BR P- 002
000254 021022 .. LDI 18
000255 171711 .. LRA S- 011
000256 010201 .. LSL 1 BIT
000257 031006 2. PCAL SIO"
000260 035411 1. SUBS %0011
000261 031003 2. PCAL TFURM"
000262 041402 C. LOAD Q+ 002
000263 041402 C. LOAD Q+ 002
000264 021010 .. LDI 8
000265 006000 .. LADD, NOP
000266 171406 .. LRA Q+ 006
000267 171407 .. LRA Q+ 007
000270 171410 .. LRA Q+ 010
000271 031001 2. PCAL %0001
000272 000707 .. DZRO, DZKO
000273 021002 .. LDI 2
000274 172003 .. LRA P+ 003,I
000275 031004 2. PCAL FMTINIT"
000276 140014 .. BR P+ 014
000277 000025 .. NOP , TEST
000300 041101 BA LOAD DB+101
000301 041513 CK LOAD Q+ 113
000302 020106 F MVBL SDEC=2
000303 051117 RO STOR DB+117
000304 046440 M LOAD P- 040,I,X
000305 051525 SU STOR Q+ 125
000306 041122 BR LOAD DB+122
000307 047525 OU LOAD Q+ 125,I,X
000310 052111 TI MTBA P+ 111
000311 047105 NE LOAD DB+105,I,X
000312 025412 +. LDXN 10
000313 044401 I. LOAD P- 001,X
000314 011202 .. IXBZ P+2
000315 140402 .. BR P- 002
000316 021024 .. LDI 20
000317 171712 .. LRA S- 012
000320 010201 .. LSL 1 BIT
000321 031006 2. PCAL SIO"
000322 035412 1. SUBS %0012
000323 031003 2. PCAL TFURM"
000324 031010 2. PCAL TERMINATE"
000325 177777 .. LRA S- 077,I,X
000326 177777 .. LRA S- 077,I,X
000327 101033 SEGMENT TRANSFER TABLE (PL-%010) SEGMENT %033 STT %002
000330 105524 SEGMENT TRANSFER TABLE (PL-%007) SEGMENT %124 STT %013
000331 110124 SEGMENT TRANSFER TABLE (PL-%006) SEGMENT %124 STT %020
000332 120113 SEGMENT TRANSFER TABLE (PL-%005) SEGMENT %113 STT %040
000333 104524 SEGMENT TRANSFER TABLE (PL-%004) SEGMENT %124 STT %011
000334 105124 SEGMENT TRANSFER TABLE (PL-%003) SEGMENT %124 STT %012
000335 000172 SEGMENT TRANSFER TABLE (PL-%002) INTERNAL
000336 000000 SEGMENT TRANSFER TABLE (PL-%001) INTERNAL

```

Figure A-1. DECOMP Listing (Sheet 3 of 3)

A

- A command,
 - definition, 2-5
 - use, 3-28
- A prefix,
 - D command, 2-12
 - M command, 2-20
 - = command, 2-27
- Abort,
 - debug at, 4-1
- Abort stack analysis, 2-31; 4-1
- Absolute code,
 - breakpoints in, 2-6; 3-28
 - clear break in, 2-9
 - display, 2-11; 3-29
 - modify, 2-19; 3-29
- Absolute code segment index (CST),
 - display, 2-13; 3-30
- Absolute DB relative (DX),
 - display base, 2-11
 - modify base, 2-19
- Absolute relative (A),
 - display base, 2-11
 - modify base, 2-19
- Account segmented library,
 - break in, 2-6, 23
 - clear break in, 2-9
 - display location of, 2-25
- Address,
 - in expression, 2-2
- Addressing,
 - array, 3-13
 - byte, 3-13
 - common, 3-14
 - indirect, 2-2, 12, 19; 3-12
- Array addressing, 3-13
- ASCII characters,
 - in expression, 2-2
- ASCII mode,
 - display, 2-11
 - expression result, 2-27
 - modify, 2-19
- ASCII parameter, 2-34

B

- B command,
 - definition, 2-6
 - use, 3-6
 - use in privileged mode, 3-28
- Bank number absolute (EA),
 - display, 2-13; 3-30
 - extended absolute address base, 2-11; 3-29
- Batch job,
 - using SETDUMP in, 4-6

- Breakpoint,
 - conditional, 2-6, 23; 3-7
 - clear, 2-9; 3-8
 - definition, 3-6
 - display, 2-7; 3-7
 - establish, 2-6; 3-6
 - establish in privileged mode, 3-28
 - permanent, 2-6, 23; 3-7
 - private, 2-5; 3-28
 - repeated, 2-7, 23; 3-7
 - system global, 2-5; 3-28
- Bit field,
 - extract, 2-2
- Byte addressing, 3-13

C

- C command,
 - definition, 2-9
 - use, 3-8
- Code segment relative (CO),
 - display base, 2-11; 3-29, 31
 - freeze, 2-17; 3-30
 - unfreeze, 2-26; 3-30
- Commands,
 - DEBUG, 2-2
 - MPE, 2-2
 - privileged mode summary, 3-27
 - specification, 2-1
 - summary, 2-4
- Common,
 - addressing, 3-14
- CST register,
 - display, 2-13, 25; 3-30

D

- D command,
 - definition, 2-11
 - use, 3-11; 3-18
 - use in privileged mode, 3-29
- Data segment index, extra (DST),
 - display, 2-13; 3-30
- Data segment relative (DA),
 - display base, 2-11; 3-29, 31
 - freeze, 2-17; 3-30
 - modify, 2-19; 3-29
 - unfreeze, 2-26; 3-30
- Data stack,
 - finding value in, 3-11 thru 3-15
- DB absolute address, 2-11, 19
- DB parameter, 2-34
- DB register,
 - display absolute value of, 2-13
- DB-relative addressing, 3-12

DEBUG facility,
 capabilities, 1-1
 command specifications, 2-2
 command summary, 2-4
 execution with, 3-5
 preparing to use, 3-1
 privileged mode capability, 3-26
 standard capability, 3-5

DEBUG intrinsic,
 definition, 2-29
 use, 3-21

Decimal mode,
 display, 2-11
 expression result, 2-27
 modify, 2-19

Decimal value,
 in expression, 2-2

DECOMP program,
 example, A-1
 use, 3-1

Disc sector,
 display, 2-15; 3-29

Display,
 code, 2-11; 3-18
 privileged mode, 2-11; 3-29
 register, 2-13; 3-16
 stack marker, 2-25; 3-20
 value, 2-11; 3-12

Display base, 2-11; 3-30

DL register,
 assign value, 2-28; 3-17
 display, 2-14
 modify, 2-21; 3-16

DR command,
 definition, 2-13
 use, 3-16
 use in privileged mode, 3-30

DST register,
 display, 2-13; 3-30

Dump stack, 2-35; 4-8

DV command,
 definition, 2-15
 use, 3-29

DX register,
 display, 2-13; 3-30

E

E command,
 definition, 2-16
 use, 3-9

EA register,
 display, 2-13; 3-30

Error messages, 5-1

Expressions,
 calculation of, 2-27; 3-17
 use in DEBUG commands, 2-2

Extended absolute relative (EA),
 display base, 2-11; 3-29
 modify base, 2-19

Extra data segment index (DST),
 display, 2-13; 3-30

Extract bit field,
 in expression, 2-2

F

F command,
 definition, 2-17
 use, 3-30

Freezing segments, 2-17; 3-30

G

Global breakpoints, 2-5; 3-28

Group segmented library,
 break in, 2-6, 23
 clear break in, 2-9
 display location of, 2-25

I

I prefix,
 D command, 2-12
 M command, 2-20
 = command, 2-27

Indirect addressing, 3-12

Interactive session,
 using SETDUMP in, 4-3

Intrinsics,
 specification of, 2-3
 summary, 2-30

J

Job,
 using SETDUMP in, 4-6

L

L command,
 definition, 2-18
 use, 3-9
 use in privileged mode, 3-28

Label map, 3-2

LABEL parameter,
 use, 3-1
 listing, 3-2

LCST register,
 display, 2-13, 25
 stack analysis, 4-1, 2

Line printer,
 display to, 2-18; 3-9, 28

Logical code segment index (LCST),
 display, 2-13, 25
 stack analysis, 4-2

Logical device,
display to, 2-18; 3-28

M

M command,
definition, 2-19
use, 3-16
use in privileged mode, 3-29

MAP parameter,
use, 3-1
listing, 3-2

Memory location,
in expression, 2-2

Modify,
code, 2-11; 3-18
privileged mode, 2-19; 3-29
register, 2-21; 3-16
value, 2-19; 3-16

Modify base, 2-19

MPE commands,
specification, 2-2
use in DEBUG, 4-3, 6, 7

MR command,
definition, 2-21
use, 3-16
use in privileged mode, 3-30

O

O prefix,
D command, 2-12
M command, 2-20
= command, 2-27

Octal mode,
display, 2-11
expression result, 2-27
modify, 2-19

Octal value,
in expression, 2-2

Offset,
display base, 2-12
modify base, 2-19

P

P register,
assign value, 2-18; 3-17
display, 2-13, 25
modify, 2-21; 3-16
stack analysis, 3-20; 4-2

P-relative addressing, 2-11; 3-18

Parameters,
DEBUG commands, 2-2
DEBUG intrinsics, 2-3
MPE commands, 2-3

PB-relative addressing, 2-11; 3-18

PCB register,
display, 2-13; 3-13

PL-relative addressing, 2-11; 3-20

PMAP parameter,
use, 3-1
listing, 3-4

Private breakpoints, 2-5; 3-28

Privileged Mode,
command summary, 3-27
use with DEBUG, 3-26

Process control block (PCB),
display, 2-13; 3-30

Q

Q register,
assign value, 2-28; 3-17
displacement from, 2-25; 4-2
display, 2-13; 3-16
modify, 2-21; 3-16
stack analysis, 3-20; 4-2

Q-relative addressing, 3-11, 12

QS parameter, 2-34

R

R command,
definition, 2-23
use, 3-9
use in privileged mode, 3-28

Registers,
assign value, 2-28; 3-17
display, 2-13; 3-16
modify, 2-21; 3-16

RESETDUMP command,
definition, 2-32
use, 4-7

RESETDUMP intrinsic,
definition, 2-31
use, 4-7

Resuming execution,
with E command, 2-16; 3-9
with R command, 2-23; 3-9

S

S register,
assign value, 2-28; 3-17
display, 2-13; 3-16
modify, 2-21; 3-16
stack analysis, 4-2

Sector,
display, 2-15; 3-29

Segment number,
B command, 2-7
location, 3-6
R command, 2-23

Segments,
freezing, 2-17; 3-30
unfreezing, 2-26; 3-30

Session,
using SETDUMP in, 4-3

SETDUMP command,
 definition, 2-34
 use in job, 4-6
 use in session, 4-3
SETDUMP intrinsic,
 definition, 2-33
 use in job, 4-6
 use in session, 4-5
ST parameter, 2-34
ST register,
 assign value, 2-28; 3-17
 display, 2-14; 3-16
 modify, 2-21; 3-16
Stack,
 delete values from, 2-16; 4-5
STACKDUMP intrinsic,
 definition, 2-35
 use, 4-8
Stack dump,
 analysis, 4-10
Stack Dump facility,
 capabilities, 1-2
 specifications, 2-30
 use, 4-1
Stack markers,
 display, 2-25; 3-20; 4-1
Stack segment index (STAK),
 display, 2-13; 3-30
Stack trace and analysis, 4-1
STAK register,
 display, 2-13; 3-30
Symbol map, 3-2
System global breakpoints, 2-5; 3-28
System global relative (SY),
 display base, 2-11; 3-29
 modify base, 2-19
System segmented library,
 break in, 2-6, 23
 clear break in, 2-9
 display location of, 2-25

T

T command,
 definition, 2-25
 use, 3-30

Temporary registers,
 assign value, 2-28; 3-17
 display, 2-13; 3-16
 modify, 2-21; 3-16
Terminate execution, 2-16; 3-9
Trace stack markers, 2-25; 3-20

U

U command,
 definition, 2-26
 use, 3-30
Unfreezing segments, 2-26; 3-30

X

X register,
 assign value, 2-28; 3-17
 display, 2-13; 3-16
 modify, 2-21; 3-16

Z

Z register,
 assign value, 2-28; 3-17
 display, 2-13; 3-16
 modify, 2-21, 28; 3-16

Special Characters

prefix, 2-2
\$ command,
 definition, 2-28
 use, 3-17
% prefix, 2-2
= command,
 definition, 2-27
 use, 3-17

READER COMMENT SHEET

**HP 3000 Computer Systems
MPE Debug/Stack Dump
Reference Manual**

30000-90012 Oct 1978

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

Is this manual technically accurate?

Did you have any difficulty in understanding concepts or wording? Where?

Is the format of this manual convenient in size, arrangement, and readability? What improvements would you suggest?

Other comments?

FROM:

Name _____

Company _____

Address _____

FOLD

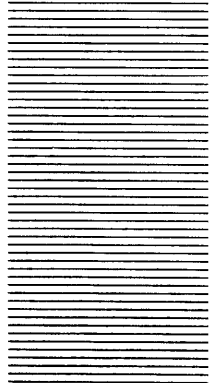
FOLD

FIRST CLASS
PERMIT NO. 1020
SANTA CLARA
CALIFORNIA

BUSINESS REPLY MAIL

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

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



FOLD

FOLD

Part No. 30000-90012
Printed in U.S.A. 9/76
Update #1 Incorporated 10/78



Sales and service from 172 offices in 65 countries.
5303 Stevens Creek Blvd. Santa Clara, California 95050