SYMBOLIC DEBUGGER

User's Guide

June 1981

Publication: 321-001090-200

Supersedes 323-321514-001

Formerly: 323-321514-002

GOULD

Electronics & Electrical Products

This manual is supplied without representation or warranty of any kind. Gould Inc., S.E.L. Computer Systems Division therefore assumes no responsibility and shall have no liability of any kind arising from the supply or use of this publication or any material contained herein.

LIMITED RIGHTS LEGEND

Use, duplication, or disclosure of data contained within this document is subject to the restrictions stated in SYSTEMS' Proprietary Agreement (Form No. 903) or, for Government customers, DAR 7-104.9A.

First Printed March, 1980
Printed in the U.S.A.

PREFACE

The Symbolic Debugger User's Guide provides a functional description and operating procedures on SYSTEMS Symbolic Debugger. This user's guide contains seven chapters:

- An overview of the Symbolic Debugger
- File assignment usage and interpretation
- Various means of accessing the Symbolic Debugger
- Program execution using the Symbolic Debugger with regard to registers, memory, symbols and expressions
- A description of commands available for use with the Symbolic Debugger
- Error messages and abort code summaries
- Sample programs and debugging sessions showing various debugging techniques.

CONTENTS

CONTENTS

Document Change History

Insert latest change pages and dispose of superseded pages. On a changed page, the portion of the page affected by the latest changes is indicated by a vertical bar in the outer margin of the page. However, a completely changed page will not have a full length bar, but will have the change notation by the page number.

Software Revision	Document Revision	Release Date	Software Release #
O	000	3/80	1.0
C	001	7/80	1.1
F	002	6/81	2.0

O = Original F = Formal Revision C = Change Package

List of Effective Pages

*A "0" in this column indicates an original page, a "1" indicates a page changed by Change Packet 1, a "2" indicates a page changed by Change Packet 2, and so forth.

Documentation Conventions

Notation conventions used in command syntax and examples through this manual are listed below.

Notation	Description
lowercase letters	Lowercase letters identify a generic element that must be replaced with a user-selected value. For example, the syntax statement: !ACTIVATE taskname taskname could be entered as MYTASK, as in: !ACTIVATE MYTASK
CAPITAL LETTERS	Capital letters must be entered as shown for input, and will be printed as shown in output. MEM,class — specifies entering MEM followed by a memory class (E, H, or S).
[ ]	An element inside brackets is optional. Several elements placed one under the other inside a pair of brackets means that the user may select any one or none of those elements. [CURR] — specifies the term CURR may be entered but is not required. T[,taskname / ,taskno] — specifies entering the letter T then either a taskname or a tasknumber may be entered (both are optional).
{ }	Elements placed one under the other inside a pair of braces identify a required choice. T{,taskname / ,taskno} — specifies entering the letter T then either a taskname or a task number must be entered.
...	The horizontal ellipsis indicates that the previous bracketed element may be repeated, or that elements have been omitted.

$name_1,...,name_n$	specifies one or more values may be entered, with a comma inserted between each name value.

. . . The vertical ellipsis indicates that commands, parameters, or instructions have been omitted.

COLLECT 1,3 . . . LIST	specifies there are one or more commands omitted between the COLLECT and LIST commands.

Numbers and special characters

Numbers that appear on the line (i.e., not subscripts), special symbols, and punctuation marks other than dotted lines, brackets, braces, and underlines appear as shown in output messages and must be entered as shown when input.

(value)	specifies the proper value must be entered enclosed in parentheses; e.g., (234).

Underscore

In examples, all terminal input is underscored; terminal output is not.

TSM>ASSIGN1	specifies TSM was displayed on the terminal, but ASSIGN1 was typed by the terminal user.

In syntax statements, underscoring is used to show the acceptable abbreviation (if any) of commands and key words. For example, the syntax:

ACTIVATE taskname	specifies the command verb ACTIVATE can either be spelled out or abbreviated to ACTI. In some cases, any character following the underlined portions of the verb is ignored. In other cases, the verb must either be spelled out correctly or abbreviated to exactly the underlined portion.

CHAPTER 1—OVERVIEW

1.1 General Description

The Symbolic Debugger is an optional software package available for use with the MPX-32 (Mapped Programming Executive) operating system. The Symbolic Debugger is a tool to assist in locating program errors in all languages supported by MPX-32. However, the symbolic capabilities are available only with FORTRAN (both 77+ and 66+) and Assembly Language.

The Symbolic Debugger is used as a replacement for the Debug load module provided with MPX-32 and provides enhanced capabilities including: symbolic access, new commands (including new data commands) and additional capabilities to aid during the testing and debugging phase of program development.

The debugging phase is begun after a program is successfully compiled or assembled and cataloged into an MPX-32 load module.

The Symbolic Debugger provides a stable environment to verify the correct execution of a program or to locate any logic errors that may prevent proper execution. This is accomplished through Symbolic Debugger commands. The commands provide access to specific memory addresses where the user suspects errors. The contents of the addresses can be displayed to verify correctness or to locate errors.

In FORTRAN and Assembly Language, addresses may be accessed through the use of local and global symbols defined in the source program. These symbols represent memory addresses, therefore the user does not need to know exact numerical addresses. If the location is associated with a symbol, it can be accessed by using that symbol name in the appropriate address parameter or command expression of a Symbolic Debugger Command. If the location to be accessed is not identified by a symbol, the name of the previous local symbol plus the offset to the desired location must be entered in the command expression.

1.2 Local and Global Symbols

The symbols used by the Symbolic Debugger are divided into two groups, local symbols and global symbols. Local symbols are those symbols defined within a specific source program, and accessed by that program. Global symbols are symbols defined within a specific source program, and can be referenced by other programs to provide interprogram linkage.

In FORTRAN, the following groups are local symbols:

- Array Names
- Variable Names
- Statement Names
- Internal Functions
- Statement Functions
- Symbolic Constants*
- Statement Numbers**

* A symbolic constant can be used as a local symbol only if its value is less than -2^{15}, greater than 2^{15}, or it is passed as an actual argument in a subroutine or function call.

** FORTRAN-77+ Release 3.0 assigns a statement number to each executable FORTRAN statement. The format for the statement number is S.x where x is the sequential location of the statement from the beginning of the respective program. Statement numbers are treated as local symbols by the Symbolic Debugger and can be used as address parameters or command expressions in Symbolic Debugger Commands. Assembly Language statements embedded in a FORTRAN statement may be accessed by using the FORTRAN statement number plus the offset to the appropriate Assembly Language statement.

In FORTRAN, the following groups are global symbols:

- Program Names
- Subroutine Names
- Function Names
- Entry Points

In Assembly Language, local symbols are all symbols used as address labels. Global symbols must be defined as linkage symbols through the DEF directive in the assembly stage of the defining program, and referenced as linkage symbol through the EXT directive in the assembly stage of the referring program.

1.3 Accessing Program Symbols

Option 19 must be set for both the compiler/assembler and the Cataloger to allow the Symbolic Debugger to access the program's symbols.

If option 19 is set for the compiler/assembler, all local symbols defined in the source program are written to the Cataloger.

If option 19 is set for the Cataloger, all program names and global symbols defined in the source program are placed in a table which is accessable by the Symbolic Debugger for address references.

If option 19 is set for both the compiler/assembler and the Cataloger, all program names and all local and global symbols are placed in tables which are accessable by the Symbolic Debugger for address references.

Option 19 is set through the use of the OPTION command in TSM. The OPTION command and the options to be set are entered followed by a carriage return. Then, the taskname for which the options are to be set is entered (FORTRAN, ASSEMBLER, or CATALOG).

Note:

If option 19 is set for the compiler/assembler, it must also be set for the Cataloger. However, option 19 may be set for the Cataloger only.

1.4 Summary of Symbolic Debugger Capabilities

The Symbolic Debugger is capable of

- Debugging interactively or in batch. In either environment, Symbolic Debugger commands control the execution of the program.
- Accessing program locations (memory addresses) by using the symbols defined in the source program. Addresses are displayed as symbolic expressions.
- Displaying data in several formats (floating point, ASCII, integer, or instruction mnemonic).
- Executing program instructions one at a time and showing the result after each is executed.
- Printing a debugging session log.
- Accessing commands from a Symbolic Debugger command file to alleviate the need of entering each command individually during the debugging session.

CHAPTER 2—FILES AND FILE ASSIGNMENTS

2.1 Symbolic Debugger Files

When the Symbolic Debugger is accessed and gains control of program execution, it determines whether it has been accessed interactively, or in batch mode and makes the appropriate file assignments for the command input and output. If the Symbolic Debugger is running interactively, it also assigns a temporary file for logging the debugging session. This log file can contain up to 100 screens of data, and can be printed through the use of the LOG command or reviewed through the use of the REVIEW command. The log file is unnecessary when running in batch because the output file for the debugging session is equivalent to the log file.

All Symbolic Debugger logical file codes (lfcs) are assigned by default when the Symbolic Debugger is accessed. When running interactively, none of the default lfc assignments may be changed (no other assignment is valid). When running batch, only the default input lfc (#IN) and output lfc (#OT) assignments may be changed. This may be desirable for the output lfc (#OT) if a large quantity of output is to be produced.

2.2 Dynamic Files and Buffers

When cataloging a task for use with the Symbolic Debugger, the Symbolic Debugger requires five dynamic files and three dynamic buffers. If option 19 is set for the Cataloger and no dynamic files or buffers are to be allocated for the task to be debugged, the Cataloger will automatically allocate the files and buffers required for the Symbolic Debugger.

If the task to be debugged requires dynamic files or buffers, then the user must (through the use of the FILES and BUFFERS directives) specify the number of files and buffers that the task needs. If option 19 is set, the Cataloger will automatically add the five files and three buffers required by the Symbolic Debugger to the number of files and buffers specified in the FILES and BUFFERS directives (the number required by the task). The Cataloger will then specify in the load module the total number of dynamic files and buffers needed for execution of this task.

2.3 Interactive Mode Default File Assignments

The following is a list of the default assignments for the Symbolic Debugger when running interactively.

Logical File Code	Assignment	Description
#IN	UT	Command input is from the user's terminal
#OT	UT	Output is to the user's terminal
#01	DC,N	Temporary log file is on disc

Logical File Code	Assignment	Description
#02	SLO,N	Spooled output for LOG and DUMP commands
#03	filename [,password]	User command file. This assignment is made only if the FILE command is specified. The lfc #03 is assigned to the filename and password (if specified) in the FILE command.
#SM	load module	Symbol table. This assignment is made only if option 19 was set for the Cataloger.

2.4 Batch Mode Default File Assignments

The following is a list of the default file assignments for the Symbolic Debugger when running batch.

Logical File Code	Assignment	Description
#IN	SYC	Command input is from the SYC file
#OT	SLO,1000	Spooled output (1000 lines) is to the SLO file
#03	filename [,password]	User command file. This assignment is made only if the FILE command is specified. The lfc #03 is assigned to the filename and password (if specified) in the FILE command.
#SM	loadmodule	Symbol table. This assignment is made only if option 19 is set.

The lfc #IN and #OT may be reassigned by the user in the job control before accessing the Symbolic Debugger.

2.5 Symbolic Debugger Prompts

The Symbolic Debugger has six command prompts that may be issued depending on the last executed command. The following is a description of the six Symbolic Debugger prompts.

Prompt	Description
.	This will always be the first prompt issued by the Symbolic Debugger following its identifying message. The commands entered in response to this prompt come from the lfc #IN and will be executed immediately. This prompt will be reissued after the execution of all commands with the exception of the SET, FILE and EXIT commands.
..	This prompt identifies commands from the lfc #IN that are entered in a trap list. If the SET command was entered in response to the period (.) prompt, the Symbolic Debugger will issue the double period (..) prompt. The commands entered in response to this prompt are not executed immediately. These commands are placed in a trap list and are deferred until the trap is encountered.
>	This prompt identifies commands from a command file. If the FILE command was entered in response to any of the prompts, each command in the command file specified will be written to the lfc #OT preceded by the greater than (>) prompt. The commands preceded by this prompt are executed immediately and the result is written following each command.
>>	This prompt identifies commands from a command file that are entered in a trap list. If the SET command was entered in the command file, all subsequent commands in the command file will be entered in a trap list until a trap list terminator command is entered. When the command file is accessed (the FILE command is entered), all the commands in the trap list after the SET command and until the trap list terminator command are preceded by the double greater than (>>) prompt. The commands preceded by this prompt are not executed immediately. These commands are deferred until the trap is encountered.
!	This prompt identifies commands from a trap list. When a trap is encountered, each command in the trap list will be written to the lfc #IN preceded by the exclamation point (!) prompt. The commands preceded by this prompt are executed immediately and the result is written following each command.
!!	This prompt identifies commands from a trap list that are entered in a nested trap list. If the SET command was entered in a trap list, all commands following the SET command will be entered in a nested trap list until a trap list terminator command is entered. When the first trap is encountered, all the commands which are in the nested trap list are preceded by the double exclamation point (!!) prompt. The commands preceded by this prompt are not executed immediately. These commands are deferred until the nested trap is encountered. Refer to Section 4.6 for a description of nested traps.

CHAPTER 3—ACCESSING THE SYMBOLIC DEBUGGER

3.1 Accessing the Symbolic Debugger via TSM (Interactive Mode)

The most common method of accessing the Symbolic Debugger is via TSM (interactive mode). This is accomplished by entering the DEBUG command in response to the TSM prompt. All of the file assignments for the program must be made prior to entering the DEBUG command. These file assignments can be made by establishing defaults through the use of the Cataloger's ASSIGN directives, or by entering the TSM ASSIGN commands before entering the DEBUG command.

When the Symbolic Debugger assumes control, it will write its identifying message, the PSW at the point of execution, condition code status, program counter value (the entry point to the program) and the registers status to the lfc #OT (output) file, and prompt the user for a Symbolic Debugger command. The sequence is as follows:

1. The user enters

   TSM>DEBUG taskname

   taskname — specifies the name of the task to be debugged

2. The Debugger responds

```
MPX-32 SYMBOLIC DEBUG Vr.r mm/dd/yy,hh:mm:ss TASK NAME = taskname
PSW=pppppppp (CC=cccc)(PC=aaaaaaaa)
REGS=00000000  00000000  00000000  00000000  ................
     00000000  00000000  00000000  00000000  ................
. (the period prompt)
```

Vr.r — specifies the revision level of the Symbolic Debugger

mm/dd/yy — specifies the current date

hh:mm:ss — specifies the current time of day

taskname — specifies the name of the task to be debugged

pppppppp — specifies the program status word (PSW) at the start of the execution

cccc — specifies the value of the condition codes at the start of the execution

aaaaaaaa — specifies the current program counter value at the start of execution. This value can be displayed as a program name, a base plus an offset address, or a program name plus an offset address.

. (period prompt) specifies the Symbolic Debugger prompt. There are six Symbolic Debugger prompts. Refer to Section 2.5 Symbolic Debugger Prompts for a description of each prompt.

3. The user enters

.command

. (period prompt) specifies the Symbolic Debugger immediate prompt from the lfc #IN

command specifies one of the Symbolic Debugger commands described in Chapter 5

3.2 Accessing the Symbolic Debugger via the Batch Stream

The Symbolic Debugger can be accessed via the Batch Stream by entering the $DEBUG command in the job control. The Symbolic Debugger cannot be accessed until the program has been assembled/compiled and cataloged. Therefore, the $DEBUG command must follow the $ASSEMBLE/$FORTRAN and $CATALOG portions of the job control if the program is to be assembled/compiled, cataloged and debugged in one job stream. Otherwise, separate job control can be set up for each phase of the program development.

All the file assignments for the program to be debugged must be made prior to debugging the program. These file assignments can be made by establishing defaults through the use of the Cataloger's ASSIGN directives when the program is cataloged. If no defaults were assigned, the $ASSIGN commands must be entered in the job control preceding the $DEBUG command.

Default file assignments are made for all necessary Symbolic Debugger files (refer to Chapter 2—Files and File Assignments). Only the Symbolic Debugger input (#IN) and output (#OT) files may be changed from their default.

The Symbolic Debugger commands to be executed are entered following the $DEBUG command in the job control.

Example 1

This is an example of job control to assemble, catalog and debug a program.

```
$JOB DBG.TST username
$OPTION 19
$EXECUTE ASSEMBLE
    .
    .
    .
source program
    .
    .
    .
$OPTION 19
$EXECUTE CATALOG
    .
    .
    .
Catalog directives
    .
    .
    .
$ASSIGN1 IN=infile
$ASSIGN2 OUT=SLO,1000
$DEBUG DBG.TST
command1
command2
    .
    .
    .
commandn
$EOJ
$$
```

infile — specifies the name of the permanent disc file which contains the input for the user program

SLO — specifies the user program output is to be written to the system listed output file (spooled output) which is then written to the line printer.

Example 2

This is an example job control for debugging only. This example assumes that the program has been assembled/compiled, and cataloged and that no default program file assignments were made during Cataloging.

```
$JOB DBG.TST username
$ASSIGN1 IN=infile
$ASSIGN2 OUT=SLO,1000
$DEBUG DBG.TST
command1
command2
      .
      .
      .
commandn
$EOJ
$$
```

infile	specifies the name of the permanent disc file which contains the input for the user program
SLO	specifies the user program output is to be written to the system listed output file (spooled output) which is then written to the line printer.

3.3 Accessing the Symbolic Debugger via the Break Key

The Symbolic Debugger can be accessed by depressing the Break key on the user's terminal after a task has been activated via TSM. When the Break key is depressed, TSM will respond with the following prompt:

```
**BREAK**ON:taskname AT:aaaaaaaa  ET:tttt.tt  SEC. CONT,ABORT,OR DEBUG?
```

taskname	specifies the name of the task which was executing at the time of the break
aaaaaaaa	specifies the address at which the break occurred
tttt.tt	specifies the time in seconds at which the break occurred

The user has the choice of continuing (CONT), aborting (ABORT), or debugging (DEBUG) the program. Responding with CONT (continue) will cause the program to continue execution at the point where the break occurred. Responding with ABORT will abort program execution. Responding with DEBUG will cause MPX-32 to load the Symbolic Debugger into the last available 8K words of the 128K word address space. The Symbolic Debugger will then write its identifying message as described in Section 3.1 and wait for a command to be entered. Note that the register values in the message will specify the contents of the registers at the time the break occurred.

3.4 Accessing the Symbolic Debugger via the M.DEBUG Macro

The Symbolic Debugger can be accessed by coding the M.DEBUG macro into a program at the location where the Symbolic Debugger is to be activated. MPX-32 loads the Symbolic Debugger and transfers control to it at that point.

When debugging is complete, the program will issue the EXIT command to exit from the Symbolic Debugger and terminate the debugging session. The Symbolic Debugger will also exit if it encounters an end-of-file (EOF) from the command input device.

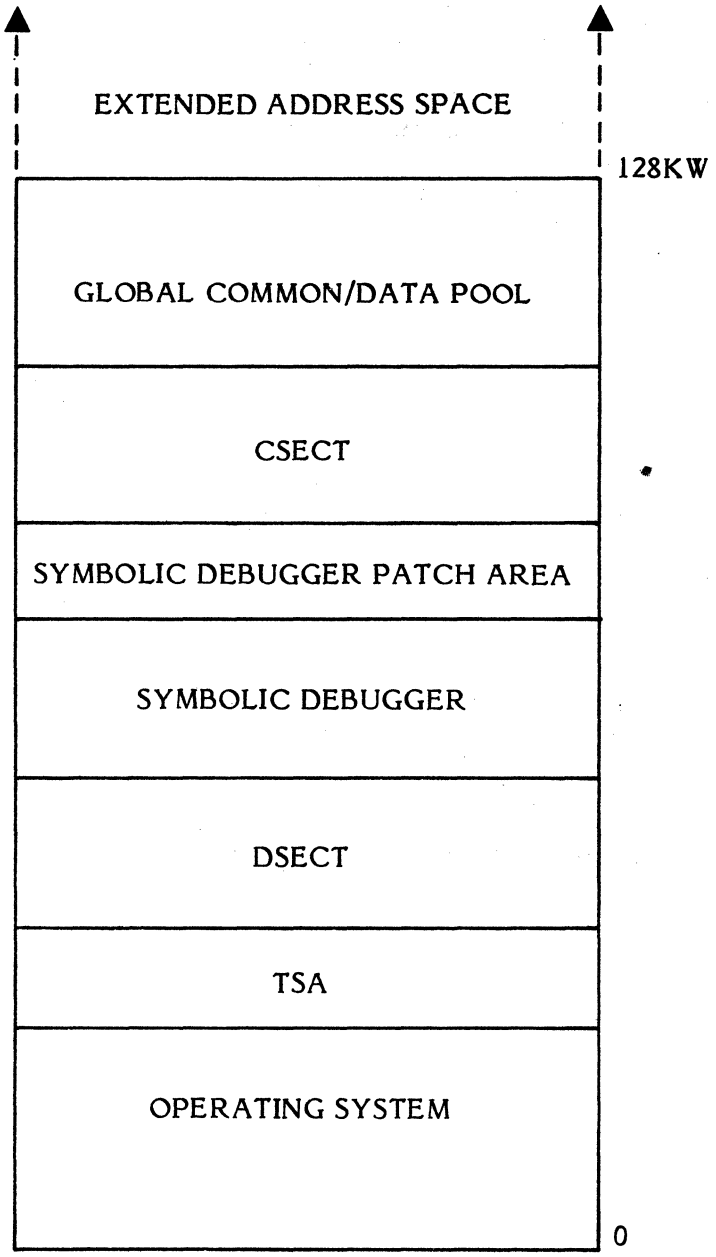
3.5 The Symbolic Debugger's Location in Memory

When the Symbolic Debugger is requested, MPX-32 loads it in the last available 8K words of the 128K word address space along with the program specified in the DEBUG command. If the Symbolic Debugger cannot fit in the available memory along with the task then only the task is loaded. If the user task contains a CSECT, the Symbolic Debugger is loaded in memory just below the beginning of the CSECT. (The CSECT is an area of memory that contains code/data that may be shared by more than one program. The code/data contained in the CSECT has read only status which is accessed almost simultaneously by the sharing programs. The read/write data referenced by the CSECT is contained in the DSECT. This data is individual (not shared) to the programs sharing the CSECT).

When the Symbolic Debugger gains control, MPX-32 preserves the registers in the Task Service Area (TSA) at offset T.CONTXT. T.CONTXT is a ten word area within the TSA of each task which is reserved for the Symbolic Debugger's use. The Symbolic Debugger uses this area as the task's register status when it gains control.

The Symbolic Debugger initializes default bases and opens the load module file for symbolic access (if option 19 is set).

The following diagram illustrates the location of the Symbolic Debugger in memory



Location of the Symbolic Debugger in Memory

CHAPTER 4—USING THE SYMBOLIC DEBUGGER

4.1 Setting the Default for Symbolic References

There are various symbol tables in which the Symbolic Debugger searches for symbol names. These are the program name table, the global symbol table and a number of local symbol tables (one for each of the program names in the program name table). A default may be set to either the global or one of the local symbol tables through the use of the PGM command. The default specifies that the symbol table to which the default is set will be the first table searched for the symbol name specified in a Symbolic Debugger command.

If the PGM command is entered without an argument, the default is set to the global symbol table (this is the default condition when the Symbolic Debugger is accessed). Global symbols may then be accessed by entering the symbol name in a command expression. Local symbols may be accessed in this default condition only if they are entered in a full pathname. A full pathname consists of the program name which defines the local symbol, a back slash (\ ) character and the local symbol name (progname\locsym). Refer to Section 4.16.5 for a more detailed description of symbols and pathnames.

The default may be changed to one of the local symbol tables by entering the PGM command and the name of the program which defines the desired local symbols. Local symbols defined in the program specified in th PGM command may then be entered without specifying the program name and backslash ( ) character. Local symbols defined in another program must still be entered in the full pathname format. Global symbols may be entered in this default condition, but if a global symbol, default local symbol and/or program have the same name, the local symbol will be accessed. Therefore, if the global symbol was desired, the default must be returned to the global symbol table (enter PGM with no argument). If a program name was desired, it must be preceded with the pound sign (#).

Examples

```
.PGM D.EXMPL
```

will allow the Symbolic Debugger to access all local symbols defined in the program D.EXMPL.

```
.PGM
```

will allow the Symbolic Debugger to access all global symbols defined in the program to be debugged (local symbols are no longer accessable without entering a full pathname).

4.2 Command Files

The Symbolic Debugger can accept commands from a permanent disc file called a command file. A command file can be created via the MPX-32 text editor. The command file can contain any number of Symbolic Debugger commands and all commands can be used in the command file except the FILE command. The commands will be executed in the order in which they are entered in the file.

To access a command file, it must first be stored uncompressed with the MPX-32 text editor STORE command. The command file can then be accessed during a debugging session by entering the FILE command and the name of the command file.

4.3 User Break Receiver

If a break occurs during the execution of a program while the Symbolic Debugger is attached, the Symbolic Debugger gains control and reports the occurrence of the break to the user. If the program has a break receiver and the user wants control passed to the break receiver, the BREAK command must be entered. This command causes the Symbolic Debugger to pass control to the break receiver if one exists. If a break receiver does not exist, the Symbolic Debugger will issue the following error message to the lfc #OT:

NO USER BREAK RECEIVER

4.4 Program Execution

The Symbolic Debugger has several commands for transferring control to the user program to begin program execution. These commands are the GO, TRACE, TRACK and WATCH commands.

After the Symbolic Debugger is accessed, the execution of the program to be debugged can be started by entering the GO command. This command can also be used to continue execution after the program has been stopped. The GO command has two optional parameters, the start address and the stop address. If the start address is not specified, the Symbolic Debugger uses the current PSW as the start address. If the stop address is not specified, program execution continues until the program completes or until an abort or trap is encountered. If no parameters are specified, the program will execute as if the Symbolic Debugger was not attached. Refer to Section 5.28, The GO Command, for a more detailed description.

Program execution may also be started with the TRACE command. The TRACE command is used to single step through program execution. This command has two optional parameters, the start address and the stop address. If a start address is specified, the Symbolic Debugger will start execution at that address and display the instruction located at that address. If no start address is specified, the Symbolic Debugger will start execution at the address specified in the current PSW and display the instruction located at that address. After each instruction is displayed a carriage return (cr) must be entered after the instruction is written to execute and display each subsequent instruction. The single step trace continues in this manner until reaching the stop address (if specified) or the end of the program (if no stop address is specified). The trace may be stopped at any time by entering any character other than a carriage return (cr) following the display of an instruction. For a more detailed description, refer to Section 5.45, The TRACE Command.

Note:

When executing a program via the TRACE command, all traps and break points are ignored.

There are two other commands that can be used to start program execution , the TRACK and WATCH commands. Both of these commands are functionally like the TRACE command. The TRACK command differs from the TRACE command in that it writes only branch instructions and their results to the lfc #OT. The WATCH command differs from the TRACE command in that it does not write any instructions or results.

The WATCH command causes the Symbolic Debugger to monitor program execution to detect erroneous branches into memory that is not within the program's address range. If a branching address error occurs, an error message will be written to the lfc #OT. There will be no other output during program execution in WATCH mode. For a more detailed description, refer to Sections 5.46 The TRACK command and 5.47 The WATCH command.

4.5 Traps and Trap Lists

Program execution may be stopped by setting traps at locations within the program. The SET command is used to place traps at the desired locations. The SET command requires one parameter, the trap (stop) address. When program execution is started by the GO command, execution continues until a trap is encountered (or until the program finishes processing). If during execution a trap is encountered, execution of the program will stop at the trap address. This allows the user to execute sections of code that are known to be correct and stop at an address where errors are suspected. When execution is stopped because a trap was encountered, the Symbolic Debugger will execute the commands in the trap list for the trap specified in the SET command.

When the SET command is entered, the Symbolic Debugger defers the execution of subsequent commands and stores them in a trap list for the trap specified in the SET command. A trap list can contain any number of Symbolic Debugger commands. The commands will be executed in the order they were entered in the trap list. All Symbolic Debugger commands can be entered in a trap list, except the LOG and REVIEW commands.

Each command entered in a trap list, except the CLEAR, FORMAT, MODE and SHOW commands, is checked for validity before being stored. If a command contains an error, the command is not entered in the trap list and an appropriate error message is written to the lfc #OT (refer to Section 6.3 Trap Error Messages). Following the error message, the user may re-enter the command or enter another command as desired. The commands CLEAR, FORMAT, MODE, and SHOW will be validated only when they are to be executed.

A trap list command may contain a user base parameter which has not yet been defined. This is not considered an error in a trap list command. Therefore, care should be taken to define all user bases either before building the trap list or before executing the trap which contains an undefined user base reference.

Trap lists are ended by entering any one of the trap list terminator commands. If the trap list contains nested traps, each trap list terminator corresponds to the trap list following the most recent unterminated SET command (refer to Section 4.6 Nested Traps). Valid trap list terminators are the BREAK, END, EXIT, FILE, GO, TRACE, TRACK and WATCH commands.

4.6 Nested Traps

Trap lists can be nested within a trap list. If a SET command is entered in a trap list, a nested trap list is built within the original trap list. When the original trap is encountered during program execution, the SET command in the trap list being executed will cause a second trap to be set at the address specified in the nested SET command. Any number of trap lists may be nested within a trap list. Each nested trap will be set only after the trap list it is nested within is executed.

Each nested trap within a trap list must have a corresponding trap list terminator. Each trap list terminator corresponds to the trap list following the most recent unterminated SET command.

Example

```
.  SET trap1 ------------- - trap is set at address specified by trap1
.. command1-1              - command1-1 is stored in the trap list for trap1
.. command1-2              - command1-2 is stored in the trap list for trap1
.. SET trap2 ------------  - trap2 will be set when trap1 is encountered
.. command2-1             - command2-1 is stored in the trap list for trap2
.. SET trap3 ----------   - trap3 will be set when trap2 is encountered
.. command3-1            - command3-1 is stored in the trap list for trap3
.. terminator3 ---------- - terminator3 is stored in and terminates the trap list for trap3
.. SET trap4 ----------   - trap4 will be set when trap2 is encountered
.. command4-1            - command4-1 is stored in the trap list for trap4
.. terminator4 ---------- - terminator4 is stored in and terminates the trap list for trap4
.. terminator2 ---------- - terminator2 is stored in and terminates the trap list for trap2
.. terminator1 ------------- terminator1 is stored in and terminates the trap list for trap1
```

4.7 Examining Memory and Registers

The Symbolic Debugger provides commands which allow the user to examine the contents of memory or registers.

The commands to display memory are as follows:

Command	Description
DA (Display ASCII)	Displays the contents of memory in ASCII format.
DD (Display Double Precision)	Displays the contents of memory in double precision floating point format.
DF (Display Floating Point)	Displays the contents of memory in single precision floating point format.
DI (Display Instruction)	Displays the contents of memory in instruction format.
DN (Display Numeric)	Displays the contents of memory in a decimal numeric format (the data size is selected from the symbol table entry).

Command	Description
DNB (Display Numeric Byte)	Displays the contents of memory in a decimal numeric byte format.
DNH (Display Numeric Halfword)	Displays the contents of memory in a decimal numeric halfword format.
DNW (Display Numeric Word)	Displays the contents of memory in a decimal numeric word format.
DUMP	Dumps the contents of memory to the line printer in a side-by-side hexadecimal and ASCII format.
SNAP	Displays the contents of memory in a side-by-side hexadecimal and ASCII format.

The eight general purpose registers can be displayed by entering the STATUS command. This command will display the contents of all general purpose registers in a side-by-side hexadecimal and ASCII format. The STATUS command has no parameters.

4.8 Modifying Memory and Registers

The Symbolic Debugger provides commands which allow the user to change memory or register values.

The contents of memory can be changed by entering the CM (Change Memory) command. This command requires two parameters separated by an equal sign (=). The first parameter is the starting address of the address values to be changed. The second parameter, which may be a list of values separated by commas, is the data to be entered in memory starting at the address specified in the first parameter. If there is only one entry in the second parameter (the data list), only the address specified will be changed. Two successive commas in the data list specify that the corresponding address word value will remain unchanged.

Example

```
.CM 100=1,2,,4
```

will cause the values 1, 2, and 4 to replace the contents of addresses 100, 104 and 10C respectively. Address 108 remains unchanged.

The contents of registers can be changed by entering the CR (Change Register) command. This command requires two parameters separated by an equal sign (=). The first parameter is the starting register (R0-R7) of the register(s) to be changed. The second parameter, which may be a list of values separated by commas, is the data to be entered in the register(s) starting with the register specified in the first parameter. If there is only one entry in the second parameter (the data list), only the register specified will be changed. Two successive commas in the data list specify that the corresponding register will remain unchanged.

Example

```
.CR R1=1,2,,4
```

will cause the values 1, 2, and 4 to replace the contents of registers R1, R2 and R4 respectively. Register R3 remains unchanged.

4.9 Selecting the Input Radix

The input radix can be selected using the FORMAT command. The default radix is hexadecimal. To change the default radix to decimal, enter the Symbolic Debugger command FORMAT N. To change the default radix back to hexadecimal, enter FORMAT X. The SHOW OPTIONS command may be entered to display the current default input radix.

4.10 Establishing User Bases

To establish a base at the beginning of a data structure or a subroutine that will be referenced frequently during the debugging process, enter the BASE command. This command requires two parameters, the base name and the expression whose value is assigned to the base. Once a base is defined, it may be used as a term in an expression in Symbolic Debugger commands. To change the value of a base, enter the BASE command. To remove all user defined bases from the Symbolic Debugger's base table, enter the CLEAR BASES command. Refer to Section 5.6 the Base Command, for a more detailed description.

4.11 Selecting Relative or Absolute Addressing

To establish a relative reference point during debugging, use the RELATIVE command. This command uses one optional parameter, a base name or program name to be the relative reference point. If the parameter is omitted, the Symbolic Debugger re-establishes the last relative name used in the program. The ABSOLUTE command is used to make all subsequent address expressions absolute, not relative to a base or program name. The SHOW OPTIONS command may be entered to display the current addressing mode (relative or absolute).

4.12 Selecting Log/No Log File

A temporary log file is allocated by default for the Symbolic Debugger when the Symbolic Debugger is accessed in interactive mode. The log file is used to store all of the commands and results of the debugging session until a LOG or REVIEW command is entered. These commands display the log file to the line printer (LOG command) or the lfc #OT (REVIEW command) and then clear the log file. All subsequent commands will be entered in the log file until another LOG or REVIEW command is entered or until the debugging session is ended.

The log file will not be maintained after the user enters the MODE NOLOG command. Entering this command will allow the Symbolic Debugger to execute faster. The log file can be maintained again by entering the MODE LOG command thus all subsequent commands will then be stored. The SHOW OPTIONS command may be entered to display whether or not a log file is being maintained.

4.13 Selecting Label Field Format

The addresses which are displayed in the label field of all Symbolic Debugger command results can be displayed in two formats. The first format is oriented to FORTRAN programs and displays the address as the program name plus the symbol name plus the offset. Program name specifies the program in which the address to be displayed is located. Symbol name specifies the symbol name within the specified program which has the closest value greater than or equal to the address to be displayed. Offset specifies the positive difference between the symbol name's value and the address to be displayed.

The second format is oriented to non-FORTRAN programs. This format displays the address as the program name plus the offset. Program name specifies the program in which the address to be displayed is located. Offset specifies the positive difference between the symbol name's value and the address to be displayed.

Entering the MODE FORTRAN command will cause the Symbolic Debugger to select the FORTRAN oriented format. Entering the MODE NOFORTRAN command will cause the Symbolic Debugger to select the non-FORTRAN oriented format. Both Assembly Language and FORTRAN programs may use either addressing format. The default setting of the Symbolic Debugger is the non-FORTRAN mode. The SHOW OPTIONS command may be entered to display the current label field format.

4.14 Selecting Extended Memory Access

If the program to be debugged uses extended memory addressing, the Symbolic Debugger can access extended memory only when the extended memory bit is set in the program status word (PSW). The extended memory bit in the PSW can only be set by executing the SEA (set extended addressing) instruction in the program being debugged. The execution of this instruction can be bypassed by entering the Symbolic Debugger MODE EXTENDED command.

The MODE EXTENDED command allows the user access to extended memory without having to execute the SEA instruction within the program. This allows the user to examine or change extended memory at any time in the debugging session. If the MODE EXTENDED command is not entered, the user would have to trace through the program location which contains the SEA instruction (setting the PSW extended memory bit) before attempting to access extended memory via a Symbolic Debugger command.

If the program to be debugged does not require extended memory access, the MODE NOEXTENDED command will not allow the user access to extended memory. This is the default condition in the Symbolic Debugger. The SHOW OPTIONS command may be entered to display the current extended memory access mode.

4.15 Symbolic Debugger Command Expressions

Many Symbolic Debugger commands have required or optional parameters specified as expressions. The Symbolic Debugger expressions are specified as arithmetic, logical, relational or single term expressions. The expressions are evaluated as 32-bit integer expressions. Each expression contains one or more valid terms. Valid terms used in expressions are integers (in the default input radix), constants, register and memory contents, base names, symbolic references, COUNT and period (.).

The rules for entering expressions are

- Operators are binary (arithmetic, logical or relational), requiring two operands.
- Expressions are evaluated left to right.
- Parentheses override left to right evaluation.
- Expressions are evaluated as 32-bit integer operations.
- Expressions contain one or more valid terms.

4.15.1 Arithmetic Expressions

The following is a description of valid arithmetic expressions (X and Y specify any valid term).

Expressions	Type	Description
X+Y	Addition	X is added to Y, overflow is ignored
X-Y	Subtraction	Y is subtracted from X, overflow is ignored
X*Y	Multiplication	X is multiplied by Y, overflow is ignored
X/Y	Division	X is divided by Y, remainder is ignored

4.15.2 Logical Expression

The following is a description of valid logical expressions (X and Y specify any valid term).

Expression	Type	Description
X^Y	Logical Shift	X is shifted Y bits to the left if Y is positive or to the right if Y is negative
X&Y	Logical AND	X is logically anded with Y
X!Y	Logical OR	X is logically ored with Y
X@Y	Exclusive OR	X is exclusively ored with Y

4.15.3 Relational Expressions

The following is a description of valid relational expressions (X and Y specify any valid term).

Expression	Type	Description
X=Y	Equal	if X is equal to Y, evaluated as TRUE or 1 (otherwise, FALSE or 0)
X< >Y	Not Equal	if X is not equal to Y, evaluated as TRUE or 1 (otherwise, FALSE or 0).
X > Y	Greater	if X is greater than Y, evaluated as TRUE or 1 (otherwise, FALSE or 0)
X<Y	Less	If X is less than Y, evaluated as TRUE or 1 (otherwise, FALSE or 0)
X>=Y	Greater or Equal	if X is greater than or equal to Y, evaluated as TRUE or 1 (otherwise, FALSE or 0)
X<=Y	Less or Equal	if X is less than or equal to Y, evaluated as TRUE or 1 (otherwise, FALSE or 0)

Note:

Single terms may be entered as expressions, and their value used as the expression result.

4.16 Terms used in Symbolic Debugger Expressions

Symbolic Debugger expressions contain one or more valid terms. The valid terms are integers, constants, register and memory contents, base names, symbolic references, COUNT and period (.).

4.16.1 Integers

Integers used as terms are entered in the default input radix. If the input radix is hexadecimal, the first digit of the integer must be 0 through 9. Therefore, if a hexadecimal integer beginning with A through F is to be entered, it must be preceded by a leading zero.

If the input radix is hexadecimal, any number of digits can be entered as a hexadecimal integer but only the last eight digits (the least significant digits) will be accepted by the Symbolic Debugger as the integer value.

If the input radix is decimal, one to ten digits can be entered as the decimal integer. If more than ten digits are entered, the Symbolic Debugger expects the eleventh digit to be a valid operator, and writes the message

MISSING OPERATOR

to the lfc #OT and reissues a prompt for another command. The user should re-enter the command with a one to ten digit decimal integer or issue another command.

4.16.2 Constants

The following are six types of constants used as terms in Symbolic Debugger expressions:

- Hexadecimal Constant - A hexadecimal constant is a string of hexadecimal digits enclosed in apostrophes and preceded by the letter X (e.g., X '1EC'). If the default input radix is hexadecimal, the letter X and the apostrophes are unnecessary. If the X and apostrophes are omitted and the hexadecimal value begins with A-F, a leading zero must precede the hexadecimal constant (synonymous with hexadecimal integer). In either format, any number of digits can be entered, but only the last eight digits (the least significant) will be used as the constant.

- Decimal Constant - A decimal constant is a string of one to ten decimal digits enclosed in apostrophes and preceded by the letter N (e.g., N '193'). If the default input radix is decimal, the letter N is unnecessary (synonymous with decimal integer). If more than ten digits are entered in a decimal constant string, the Symbolic Debugger expects the eleventh digit to be a valid operator, and writes the message

  MISSING OPERATOR

  to the lfc #OT and reissues a prompt for another command. The user should re-enter the command with a one to ten digit decimal constant or issue another command.

- Binary Constant - A binary constant is a string of one to 32 ASCII ones and zeros enclosed in apostrophes and preceded by the letter B (e.g., B'101011'). If fewer than 32 digits are entered, leading binary zeros are added to produce a 32-bit value.

- Floating Point Constant - A floating point constant is a string of one to 21 decimal digits enclosed in apostrophes and preceded by the letter E. The floating point string is entered in one of three formats, a single precision value without an exponent, a single precision value with an exponent (denoted by the letter E) or a double precision value with an exponent (denoted by the letter D). The mantissa and the exponent can optionally be designated as positive (+) or negative (-).

  Examples:

  A single precision floating point constant without an exponent.

  E'0.999'

  A positive single precision floating point constant with a negative exponent.

  E'+100.32E-10'

  A negative double precision floating point constant with an exponent

  E'-100.32D10'

- C-Character Constant - A C-character constant is a string of one to four ASCII characters enclosed in apostrophes and preceded by the letter C (e.g., C'A1?'). C-character constants are left justified and trailing blanks are added to produce a 32-bit value, if fewer than four characters are entered.

- G-Character Constant - A G-character constant is a string of one to four ASCII characters enclosed in apostrophes and preceded by the letter G (e.g., G'A1?'). G-character constants are right-justified and leading binary zeros are added to produce a 32-bit value, if fewer than four characters are entered.

4.16.3 Register and Memory Contents

The contents of registers and memory are used as terms in expressions by specifying the register name or the memory address of the contents to be used.

Register contents are used by entering any of the eight general purpose registers in the form Rn (n specifies a register number 0 through 7).

Memory contents are used by entering the address of the contents to be used in one of the following formats:

C (address)
C (address ± hex)
C (address ± dec)
C (hex)
C (dec)

C	specifies the contents of the term enclosed in parentheses is to be used in the expression
address	specifies a base name, program name, symbol, period (.) or explicit pathname (program name plus symbol name plus offset or program name plus offset).
hex	specifies a hexadecimal value
dec	specifies a decimal value

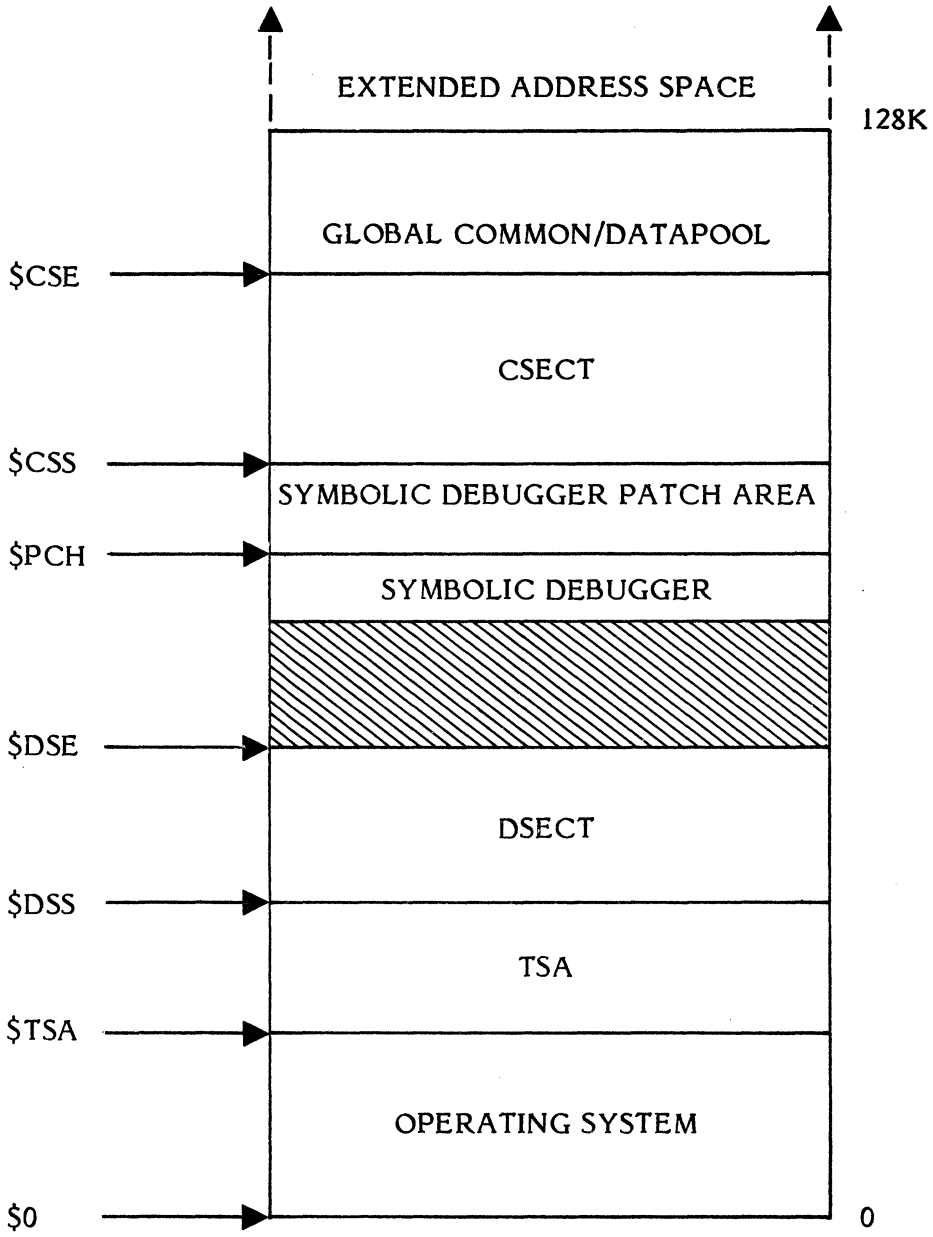
These expressions specify the 32-bit contents of the address or expression inside the parentheses. Bits 30 and 31 of the expression value are zeroed to determine the word address.

4.16.4 Bases

Bases are symbolic terms used in expressions. A base name is denoted by a $ as the first character. The Symbolic Debugger defines nine bases when it is accessed. The nine Symbolic Debugger bases are

Base Name	Description
$	Bits 13-31 of the user task program status doubleword (PSD)
$PSD	Bits 0-31 of the user task PSD
$0	Constant zero
$TSA	Start address of the user task's task service area (TSA)
$DSS	Start address of the user task's DSECT
$DSE	End address of the user task's DSECT
$PCH	Start address of the 256-word patch area (in the Symbolic Debugger)
$CSS	Start address of the user task's CSECT
$CSE	End address of the user task's CSECT
$END	End address of the user task's extended memory

The relative position of some of the bases described above on a memory map of a user task which uses all possible memory areas (CSECT, DSECT, Global Common, and extended memory) is shown below.



SYMBOLIC DEBUGGER BASE NAMES

Bases, other than the reserved Symbolic Debugger bases, can be defined through the use of the BASE command. A user defined base name begins with a $ and an alphabetic character followed by one to seven alphanumeric characters. User defined base names may not be the same as any of the reserved Symbolic Debugger bases.

4.16.5 Symbols

Programs assembled/compiled and cataloged with option 19 set allow the Symbolic Debugger access to program names, global symbols and local symbols. If option 19 is set only for the Cataloger, only program names and global symbols can be accessed.

Program names are denoted by the special character # (pound sign) and symbol names (local and global) are denoted by the special character \ (backslash). Both special characters are optional, but if global symbols, local symbols and/or programs have the same name, the special characters should be entered for clarity.

When the Symbolic Debugger is accessed, it defaults to searching for global symbols. If the default is not changed (via the PGM command), local symbols must be preceded by the program name in which they are located and the backslash (\) character for the Symbolic Debugger to access them.

If the PGM command and a program name are entered, the Symbolic Debugger then defaults to the local symbols within the specified program name. In this default condition, if a symbol is entered without the special character, the Symbolic Debugger will first search the local symbol table (of the specified program). If the symbol is not found, the global symbol table will be searched. If the symbol is not found in the global symbol table, the program table will be searched. Therefore, the special characters should be used to avoid ambiguous cases (symbols and programs with the same name).

Local symbols which are not located in the program specified in the PGM command must be preceded by the program name in which they are defined and the backslash (\) character.

The following syntax shows valid symbolic addresses:

Syntax

[#] progname

#	specifies the optional special character to denote a program name
progname	specifies the program name to be used as a symbolic address

Syntax

[\]glosym

\	specifies the optional special character to denote a symbol name
glosym	specifies the global symbol to be used as a symbolic address. If the default is to local symbols and a local symbol exists with the same name, the PGM command must be entered without a program name to set the default to global symbols. Otherwise, the local symbol by that name will be accessed.

Syntax

[[#]progname] [\]locsym

#	specifies the optional special character to denote a program name
progname	specifies the optional program name if the default is set to local symbols defined in that program name. If the default is set to global symbols or to local symbols defined under a different program name, then the program name must be specified followed by the backslash character and the local symbol name.
\	specifies the optional special character to denote a symbol name. This character is optional if the local symbol name it precedes is in the default local symbol table. Otherwise it must be specified.
locsym	specifies the local symbol to be used as a symbolic address.

4.16.6 COUNT

COUNT is a special term in expressions used to determine how many times a trap has been encountered since it was set. When a trap is set, a counter is established to track the number of times the trap is encountered. COUNT is always updated to reflect the number of times that the last trap in the program was encountered. Therefore, COUNT can be specified after each trap to determine how many times each trap has occurred.

COUNT is useful in conditional trap lists. If a program has a loop which executes properly a number of times and then encounters an error, a trap can be set at the beginning of the loop with a conditional trap list to execute only when COUNT equals the number of times the loop executed properly. Then, through the commands in the trap list, the user can examine memory or register contents during the iteration of the loop which contains the error.

4.16.7 Period (.)

The special character period (.) is equal to the last address displayed by a memory related command. The period (.) is used as a term in an expression in place of re-entering the last displayed address.

The period (.) is set by the execution of the CM, DA, DD, DF, DI, DN, DNB, DNH, DNW and SNAP commands.

Example

The user enters the command

```
.DI #DBGTST\SYMBOL1
```

The Symbolic Debugger responds

```
DBGTST\SYMBOL1          LW R5, DBGTST\SYMBOL2
```

The user enters the command

```
.SET .
```

The Symbolic Debugger issues the trap list prompt and the user enters the command

```
..END
```

The Symbolic Debugger sets a trap at the address specified by period (.) which is DBGTST\SYMBOL1 with no corresponding trap list commands.

CHAPTER 5—SYMBOLIC DEBUGGER COMMANDS

5.1 Using the Symbolic Debugger Commands

The following rules apply to the Symbolic Debugger commands, whether they are entered from the lfc #IN (batch mode or interactive mode) or from a command file (lfc #03) through the use of the FILE command.

- Each command record read from the lfc #IN is placed in a 72-character buffer. If the record size of the file/device assigned to the lfc #IN is other than 72 characters, the command is left-justified and blank-filled or truncated to the 72-character buffer size.
- Compound commands and continuation of command lines are not allowed.
- All commands have a command verb. Some command verbs may be abbreviated by entering the characters underlined in the syntax. If no command verb is entered, the Symbolic Debugger defaults to the SNAP command.
- The command verb is followed by a termination character (any non-alphabetic character) and the command argument list (if required). Multiple arguments are separated by commas (,). Blanks in a command line are ignored except inside a G or C character constant.
- Error messages are written to the lfc #OT when an incorrect command is entered. Refer to Chapter 6 for a description of the error messages.
- The response to each entered command is written to the lfc #OT following that command. (Some commands have no displayed response.)

5.2 Summary of Symbolic Debugger Commands

Command	Description
A	Displays the address of the specified expression.
ABSOLUTE	Sets absolute mode. All subsequent address expressions are evaluated and displayed as absolute addresses until relative mode is set via the RELATIVE command.
B	Evaluates and displays the specified expression in binary format.
BASE	Creates, deletes or modifies a user base.
BREAK	Transfers control to a user task's break receiver.
CC	Displays or modifies condition codes in the user task's program status doubleword (PSD).

Command	Description
CLEAR	Clears all user defined bases or deletes all traps.
CM	Changes memory contents to the 32-bit value(s) specified beginning at the address specified.
CR	Changes register contents to the 32-bit value(s) specified beginning at the register specified.
DA	Displays the contents of the memory range specified in ASCII format.
DD	Displays the contents of the memory range specified in double precision floating-point format.
DELETE	Deletes the specified trap.
DETACH	Detaches the Symbolic Debugger from the user task and transfers control to the task at the address specified or at the last address executed in the task.
DF	Displays the contents of the memory range specified in single precision floating point format.
DI	Displays the contents of the memory range specified in instruction mnemonic format (Assembly language).
DN	Displays the contents of the memory range specified in decimal integer format.
DNB	Displays the contents of the memory range specified in decimal integer byte format.
DNH	Displays the contents of the memory range specified in decimal integer halfword format.
DNW	Displays the contents of the memory range specified in decimal integer word format.
DUMP	Dumps the content of the memory range specified, the task's PSD and the general purpose registers to the line printer (interactive mode) or to the lfc #OT (batch mode) in a side-by-side hexadecimal and ASCII format.
E	Evaluates and displays the result of the expression specified in single precision floating point format.
END	Terminates a trap list and returns control to the lfc #IN.
EXIT or X	Terminates the Symbolic Debugger and the user task.
FILE	Passes control to the command file specified to read and execute the commands in the file, then return control to the lfc #IN.

Command	Description
FORMAT	Sets the default radix to either decimal or hexadecimal for undesignated values in expressions.
GO	Begins execution of the user's task at the address specified or at the current program counter value.
IF	Establishes conditional trap list execution.
LIST	Displays the trap list for the specified trap.
LOG	Writes the temporary log file to the line printer (interactive mode only, not available in batch mode).
MODE	Sets log/no log file, extended/no extended memory access, and FORTRAN/NOFORTRAN label field format.
MSG	Designates a comment line.
N	Evaluates and displays the result of the expression specified in signed decimal format.
PGM	Establishes the program name specified as the default for local symbol searches or if no program name is specified, defaults to global symbol searches.
RELATIVE	Sets relative mode and optionally establishes a new relative base or program name.
REVIEW	Writes the temporary log file (one screen at a time) to the lfc #IN (interactive mode only, not available in batch mode).
RUN	Sets run mode (as opposed to single-step) for tracing or tracking. A full screen of program instructions will be displayed before prompting for continuation or termination of the trace or track.
SET	Sets a trap at the word address specified and prompts for a trap list command.
SHOW	Displays trap addresses, base names and values, option settings and/or symbols.
SNAP	Displays the contents of the memory range specified in side-by-side hexadecimal and ASCII format.
STATUS	Displays the status of the user PSD and general purpose registers at the current address.
STEP	Sets single-step mode for subsequent TRACE or TRACK commands. One program instruction will be displayed before prompting for continuation or termination of the trace or track.
TIME	Displays the current date and time.

Command	Description
TRACE	Transfers control to the user task and displays each instruction after it is executed.
TRACK	Transfers control to the user task and displays each branch instruction after it is executed.
WATCH	Transfers control to the user task and reports any erroneous branches into memory (no instructions are displayed).
X	Evaluates and displays the result of the expression specified in hexadecimal format. If no expression is specified, X is interpreted as the EXIT command.

5.3 The A (Address) Command

The A command is used to evaluate and display an expression in address format. If extended addressing mode is not set, 19 bits are used. If extended addressing mode is set, 24 bits are used.

Syntax:

A expr

expr specifies any valid Symbolic Debugger expression.

Response:

In Relative mode, the address is displayed as the closest base or program name to the value plus the positive offset, in hexadecimal.

In Absolute mode, the address is displayed as a hexadecimal number without leading zeros.

5.4 The ABSOLUTE Command

The ABSOLUTE command is used to set the absolute mode. As a result, subsequent address expressions are interpreted as absolute and displayed as absolute hexadecimal logical addresses. This mode is in effect until a RELATIVE command is executed.

Syntax:

ABSOLUTE

Response:

The command is always valid.

There is no output.

The Symbolic Debugger prompts for the next command.

5.5 The B (Binary) Command

The B command is used to evaluate an expression and display its result in binary format.

Syntax:

B expr

expr — specifies any valid Symbolic Debugger expression

Response:

The 32-character binary equivalent of the expression is displayed.

5.6 The BASE Command

The BASE command is used to define a user base (add its name to the internal base definition table), delete a user base name from the base table, or redefine a user base (change the value specified in the base name's definition).

Up to 16 user bases are allowed. Refer to Section 4.10 Establishing User Bases.

Syntax:

BASE base [,expr]

base — specifies a user base name. A user base name must begin with the character $ and an alphabetic character. A base name can be a maximum of eight alphanumeric characters.

expr — specifies a logical address to be used as the base's value. If the expression is not specified, the base name is deleted. If "expr" is specified and "base" is already defined, "base" is redefined to the value specified by "expr".

Response:

There is no output except error messages. Error messages inform the user if

- the user tries to define a new base and the base table is full (16 user bases)
- "base" is not specified
- "base" is a base name which was defined by the Symbolic Debugger and cannot be redefined
- the user attempts to delete an undefined base

5.7 The BREAK Command

The BREAK command is used to transfer control from the Symbolic Debugger to the user task's break receiver.

Syntax:

BREAK

Response:

The user break receiver gets control. The Symbolic Debugger regains control upon the occurrence of the next break, trap, user abort, or break receiver exit.

An error message informs the user if the user task has no break receiver.

The BREAK command is a trap list terminator.

5.8 The CC (Condition Code) Command

The CC command is used to display the four condition code bits in the Symbolic Debugger base $PSD (bits 0-31 of the user PSD) or to display the old condition code of $PSD and insert a new value.

Syntax:

CC [cc]

cc — is a string of four binary digits that will replace the existing condition codes in $PSD. If not specified the Symbolic Debugger displays the present condition codes.

Response:

An error message informs the user if the condition code is specified incorrectly.

The Symbolic Debugger prompts for the next command.

5.9 The CLEAR Command

The CLEAR command is used to delete all user defined bases or traps.

Syntax:

$$\underline{\text{CL}}\text{EAR} \left\{ \begin{array}{l} \underline{\text{B}}\text{ASES} \\ \underline{\text{T}}\text{RAPS} \end{array} \right\}$$

BASES indicates that all user base definitions are to be deleted.

TRAPS indicates that all traps are to be deleted.

Response:

An error message informs the user of any argument specification errors.

There is no output except for error messages. The Symbolic Debugger prompts for the next command.

5.10 The CM (Change Memory) Command

The CM command is used to alter the contents of one or more consecutive words in the task's logical address space.

Syntax:

CM addr=expr$_1$[,expr$_2$,...,expr$_n$]

addr — specifies the address of the first word to be changed (bits 30 and 31 of addr are ignored).

expr — specifies the 32-bit value to be stored at the specified address. Successive values are stored in consecutive words beginning at "addr". Two consecutive commas with no intervening value can be used to skip the memory address corresponding to the missing value, leaving its contents unchanged.

Response:

Error messages inform the user if:

- "addr" and "expr" are not both present and valid
- memory changes must be stopped because "addr" or an address derived from it (multiple values) violates a Symbolic Debugger address restriction
- an error occurs in evaluating one of the "expr" values

Note: In the third case, the error message will specify which memory words, if any, were successfully changed.

A SNAP is automatically performed by the Symbolic Debugger for the modified range and the new contents are displayed.

The Symbolic Debugger prompts for the next command.

When storing a double precision floating point constant into memory, two words are changed.

The special character period (.) is set at the completion of this command (refer to Section 4.16.7 Period (.)).

5.11 The CR (Change Register) Command

The CR command is used to alter the contents of one or more user registers.

Syntax:

CR Rn=expr$_1$ [,expr$_2$,...,expr$_n$]

Rn — specifies a user register (R0-R7)

expr — specifies the 32-bit value to be stored in the specified register. Succeeding values, if any, are stored in consecutive user registers. Two consecutive commas with no intervening value can be used to skip the user register corresponding to the missing value, leaving its contents unchanged. If user register R7 has been altered or skipped and one or more unused values remain, they are ignored.

Response:

An error message informs the user if:

- A register specification is absent or not in the range 0-7.
- The first value is not specified.

The Symbolic Debugger prompts for the next command.

When changing a register to a double precision floating point constant, two registers are changed.

5.12 The DA (Display ASCII) Command

The DA command is used to display a memory range in ASCII format.

Syntax:

DA [low][,high]

low — specifies the first byte address to be displayed. If not specified, the last location displayed plus one word is used as the default.

high — specifies the last byte address of the range to be displayed. If not specified, only the low address is displayed.

Response:

The memory address(es) are displayed in label-field format. The contents of memory are displayed in ASCII format.

The special character period (.) is set at the completion of this command (refer to Section 4.16.7 Period (.)).

5.13 The DD (Display Double Precision) Command

The DD command is used to display a memory range in double precision floating point format.

Syntax:

DD [low] [,high]

low	specifies the first word address to be displayed. If not specified, the last location displayed plus one word is used as the default.
high	specifies the last word address of the range to be displayed. If not specified, only the low address is displayed.

Response:

The address(es) specified are displayed in label-field format. The contents of the specified memory address(es) plus the contents of the next word are converted to their floating point double precision equivalent and displayed.

If a range is given, the second display begins two words (8 bytes) after the first display.

The special character period (.) is set at the completion of this command (refer to Section 4.16.7 Period (.)).

5.14 The DELETE Command

The DELETE command is used to delete a specified trap and restore the user instruction to its original location.

Syntax:

DELETE trap

trap specifies a trap address.

Response:

An error message informs the user if:

- "addr" is not specified
- "addr" is not an address at which a trap has been set by the SET command

The user instruction replaced by the trap instruction is restored to its original location.

The Symbolic Debugger prompts for the next command.

5.15 The DETACH Command

The DETACH command is used to detach the Symbolic Debugger from the user task and transfer control to the task at the specified address or at $ (bits 13-31 of user PSD).

Syntax:

DETACH [addr]

addr — specifies the address within the user task to which control is transferred. If not specified, defaults to $.

Response:

All traps are deleted (there is no need to enter CLEAR TRAPS to restore user instructions replaced by trap instructions).

The Symbolic Debugger files and memory are deallocated.

The Symbolic Debugger transfers control to the specified address.

An error message informs the user if the specified address violates the Symbolic Debugger's address restriction.

DETACH is a trap list terminator.

5.16 The DF (Display Floating Point) Command

The DF command is used to display a memory range in floating point format.

Syntax:

DF [low] [,high]

low — specifies the first word address to be displayed. If not specified, the last location displayed plus one word is used as the default.

high — specifies the last word address of the range to be displayed. If not specified, only the low address is displayed.

Response:

The memory address(es) are displayed in label-field format. The content of the specified memory address is displayed in single precision floating point format.

If a range is given, the second display begins one word (4 bytes) after the first display.

The special character period (.) is set at the completion of this command (refer to Section 4.16.7 Period (.)).

5.17 The DI (Display Instruction) Command

The DI command is used to display a memory range as mnemonic instructions (Assembly Language).

Syntax:

DI [low] [,high]

low — specifies the first word or halfword address to be displayed. If not specified, the last location displayed plus one word is used as the default.

high — specifies the last word or halfword address of the range to be displayed. If not specified, only the low address is displayed.

Response:

The memory address(es) are displayed in label-field format. The contents of the memory address(es) are displayed in Assembly Language format.

The special character period (.) is set at the completion of this command (refer to Section 4.16.7 Period (.)).

5.18 The DN (Display Numeric) Command

The DN command is used to display a memory range in decimal integer format.

Syntax:

DN [low] [,high]

low — specifies the first word address to be displayed. If not specified, the last location displayed plus one word is used as the default.

high — specifies the last word address of the range to be displayed. If not specified, only the low address is displayed.

Response:

The memory address(es) are displayed in label-field format. The contents of the memory address(es) are displayed in decimal integer format.

The size is determined by the Symbol Table Entry. If there is no Symbol Table Entry, the default is one word.

The special character period (.) is set at the completion of this command (refer to Section 4.16.7 Period (.)).

5.19 The DNB (Display Numeric Byte) Command

The DNB command is used to display a memory range in decimal byte format.

Syntax:

DNB [low] [,high]

low — specifies the first byte address to be displayed. If not specified, the last location displayed plus one word is used as the default.

high — specifies the last byte address of the range to be displayed. If not specified, only the low address is displayed.

Response:

The memory address(es) are displayed in label-field format. The contents of the memory address(es) are displayed in decimal integer byte format.

The special character period (.) is set at the completion of this command (refer to Section 4.16.7 Period (.)).

5.20 The DNH (Display Numeric Halfword) Command

The DNH command is used to display a memory range in decimal halfword format.

Syntax:

DNH [low] [,high]

low — specifies the first halfword address to be displayed. If not specified, the last location displayed plus one word is used as the default.

high — specifies the last halfword address of the range to be displayed. If not specified, only the low address is displayed.

Response:

The memory address(es) are displayed in label-field format. The contents of the memory address(es) are displayed in decimal integer halfword format.

The special character period (.) is set at the completion of this command (refer to Section 4.16.7 Period (.)).

5.21 The DNW (Display Numeric Word) Command

The DNW command is used to display a memory range in decimal word format.

Syntax:

DNW [low] [,high]

low — specifies the first word address to be displayed. If not specified, the last location displayed plus one word is used as the default.

high — specifies the last word address of the range to be displayed. If not specified, only the low address is displayed.

Response:

The memory address(es) are displayed in label-field format. The contents of the memory address(es) are displayed in decimal integer word format.

The special character period (.) is set at the completion of this command (refer to Section 4.16.7 Period (.)).

5.22 The DUMP Command

The DUMP command is used to write a range of memory to the line printer in side-by-side hexadecimal and ASCII format. In batch mode, the dump is written to lfc #OT.

Syntax:

DUMP [low] [,high]

low and high — are expressions representing memory addresses. If high is not specified or is not greater than low, only the single word at low is displayed.

If no addresses are specified, the Symbolic Debugger will dump the addresses following the last address dumped. If no addresses were dumped, the Symbolic Debugger will dump the contents of memory starting at absolute address zero.

Response:

The memory range between the specified addresses is dumped to the line printer (or lfc #OT in batch mode). The user PSD and registers are also shown.

An error message is displayed if any address in the range violates an address restriction.

5.23 The E (Single Precision Floating Point) Command

The E command is used to display an expression value in single precision floating point format.

Syntax:

E expr

expr specifies the expression to be displayed in floating point format.

Response:

The single precision floating point equivalent of the expression is displayed.

5.24 The END Command

The END command is used to terminate a trap list. Using a carriage return <CR>in interactive mode performs the same function.

Syntax:

END or<CR>

Response:

END is a trap list terminator.

5.25 The EXIT Command

The EXIT command is used to terminate debugging and return to the TSM prompt. Both the user task and the Symbolic Debugger exit.

Syntax:

EXIT (or) X

Response:

The Symbolic Debugger calls the M.EXIT service.

EXIT is a trap list terminator.

The Symbolic Debugger also exits in response to a Control C (end-of-file).

5.26 The FILE Command

The FILE command is used to read subsequent Symbolic Debugger commands from a command file instead of from the lfc #IN.

Syntax:

FILE filename [,password]

filename	specifies the name of a command file on disc.
password	specifies the password, if any, associated with the file.

Response:

An error message informs the user if:

- "filename" is absent or invalid, or the file does not exist
- the password is invalid
- the user is not allowed access to the file, e.g., a password is associated with the file and has not been supplied
- the FILE command is read from a command file

If there are no errors, the Symbolic Debugger assigns lfc #03 to the specified file and reads subsequent commands from #03 instead of #IN. When the Symbolic Debugger reaches end-of-file on #03 or a break is recognized, command input returns to #IN.

The Symbolic Debugger searches for a user file by the specified filename. If a user file is not found, it then searches for a system file.

Use of the FILE command terminates a trap list.

5.27 The FORMAT Command

The FORMAT command is used to set the default input format for undesignated numeric constants and integers in expressions to hexadecimal or decimal.

Syntax:

FORMAT { X / N }

X — sets the input radix to hexadecimal, which is the default when the Symbolic Debugger is accessed.

N — sets the input radix to decimal.

Response:

An error message informs the user if the format specification is absent or invalid.

The Symbolic Debugger prompts for the next command (no output).

5.28 The GO Command

The GO command is used to transfer control to the user task, optionally setting a one-shot trap.

Syntax:

GO [addr][,trap]

addr — specifies the address within the user task to which the Symbolic Debugger transfers control. If not specified, the Symbolic Debugger base $ (bits 13-31) of the user PSD is used.

trap — specifies the address within the user task at which the Symbolic Debugger sets a trap. The list of Symbolic Debugger commands executed when the trap occurs is as follows:

```
!* ONE-SHOT TRAP SET BY GO COMMAND
!DEL $
!END
```

"$" is the special Symbolic Debugger base equal to bits 13-31 of the user PSD.

If a trap address is not specified the Symbolic Debugger does not set a trap before transferring control to the user task.

Response:

An error message informs the user if:

- either the transfer address or trap address violate the Symbolic Debugger address restrictions
- a trap address is specified and a trap is already set there
- no trap table space remains and a trap address is specified
- "addr" is an odd number
- "trap" is not on a word boundary

If GO is successful, the Symbolic Debugger transfers control to the user task at the specified address. If the last control transfer into the Symbolic Debugger was caused by a trap and control is passed to the trap address for that trap, the user instruction replaced by the trap instruction is executed first. Control is then passed to the trap address plus one word unless the replaced user instruction is any instruction which terminates the TRACE , TRACK, or WATCH commands--such a replaced instruction may not be executed without first deleting the trap set on it.

Control remains with the user task until a trap, break, or user abort occurs, at which time the Symbolic Debugger regains control and prompts for the next command.

GO is a trap list terminator.

5.29 The IF Command

The IF command is used to make a trap list conditional. (The trap list is executed only if specified conditions are met.) When used, this command must be the first command of the trap list.

Syntax:

IF expr

expr specifies any valid Symbolic Debugger expression.

Response:

If the value of "expr" is nonzero, the trap is reported and remaining commands in the trap list are executed. The relational operators produce a value of 1 if the relation is true, and a value of 0 if false (refer to Section 4.15.3 for a description of relational operators).

If the value of "expr" is zero, no trap is reported and the program continues executing as if the user issued a GO $ command.

The trap's COUNT is incremented whether the trap is reported or not .

An error message informs the user when the IF command is entered as an immediate command or when "expr" is absent or invalid.

5.30 The LIST Command

The LIST command is used to display the trap list for a specific trap.

Syntax:

LIST trap

trap specifies a trap address

Response:

An error message informs the user if "trap" is not a trap address.

5.31 The LOG Command

The LOG command is used to print the current contents of the terminal log file.

Note:

A log file is maintained only when the LOG option has been specified by the MODE command. The default condition is to maintain a log file.

Syntax:

LOG

Response:

All log file records which have not already been printed are copied to an SLO file. The SLO file is then closed and deallocated. All log file records thus copied are no longer accessible (their space is released). The LOG command is ignored in batch.

5.32 The MODE Command

The MODE command sets the following modes for the debugging session:

- A log file is/is not maintained to log the debugging session
- Extended memory access is/is not allowed
- FORTRAN display format is/is not set

Syntax:

MODE { LOG | NOLOG | EXTENDED | NOEXTENDED | FORTRAN | NOFORTRAN }

LOG	specifies that a log file will be maintained for the debugging session.
NOLOG	specifies that a log file will not be maintained for the debugging session.
EXTENDED	specifies that extended addressing will be allowed, thus the user has access to program locations in extended memory.
NOEXTENDED	specifies that extended addressing will not be allowed, thus the user must trace through an SEA (set extended addressing) instruction to access extended memory.
FORTRAN	specifies that FORTRAN addressing format is set. The address label field will be displayed as the program name and closest previous symbol name and the offset address (i.e., program\symbol + 04).
NOFORTRAN	specifies that NOFORTRAN addressing format is set. The address label field will be displayed as the program name plus the offset address (i.e., program + 04).

Response:

An error message informs the user if the mode is invalid or missing.

The Symbolic Debugger prompts for the next command (no output).

5.33 The MSG Command

The MSG command is used to denote a comment in a debugging session. It is most useful in command files and trap lists to document the commands.

Syntax:

MSG message

(or)

* message

message specifies any character string.

Response:

The character string is displayed.

5.34 The N (Numeric) Command

The N command is used to evaluate and display the expression's value in signed decimal integer format.

Syntax:

N expr

expr specifies the expression to be evaluated and displayed in signed decimal integer format.

Response:

The signed decimal integer equivalent of the expression is displayed.

5.35 The PGM (Program) Command

The PGM command is used to establish a program name in which the Debugger will search for local symbols. This command also sets a new relative program name (see RELATIVE command).

Syntax:

PGM[progname]

progname — specifies a program name which may begin with the character # (the designating character # is optional). If a program name is not specified, the current program name is cleared and the Symbolic Debugger defaults to the global symbol table.

Response:

An error message informs the user if the program name could not be found.

5.36 The RELATIVE Command

The RELATIVE command is used to set relative mode. Subsequent addresses that do not include an explicit base, program, or symbol name are interpreted as relative to the base or program name set by this command.

Syntax:

RELATIVE [base / progname]

base — is a base name which must begin with the character $ (refer to Section 4.10 Establishing User Bases).

progname — is a program name which may begin with the character # (the designating character # is optional).

If neither parameter is specified, the last base or program name that was set by a RELATIVE command is used. During initialization, the Symbolic Debugger sets the relative mode and establishes $DSS as the default base.

Response:

Each address subsequently displayed is represented as a displacement from the nearest base or program name which is not greater than the address. If a base and a program name have the same value, the Symbolic Debugger uses the program name.

An error informs the user if the specified base or program name is not defined.

5.37 The REVIEW Command

The REVIEW command is used to write the log file to the lfc #OT.

Syntax:

REVIEW [screens]

screens — specifies the number of screens from the current position in the log file for the Symbolic Debugger to backspace before beginning the log file display. If not specified or if specified as a number greater than the number of screens currently contained in the log file, the display begins at the first record in the log file.

Response:

The Symbolic Debugger displays the log file one screen at a time.

When the Symbolic Debugger reaches the end of the log file, the display is terminated and the Symbolic Debugger prompts for the next command. None of the above terminal I/O is copied to the log file.

REVIEW is treated as a comment in batch.

An error message informs the user if:

- REVIEW is entered as a deferred command
- REVIEW is read from a command file

5.38 The RUN Command

The RUN command is used to set the run mode. This results in trace or track continuing until the Symbolic Debugger reaches a full screen of output instead of prompting for input after each instruction.

Syntax:

RUN

Response:

Until a STEP command is executed, the TRACE and TRACK commands will display a full screen of output before prompting for continuation or termination of the trace or track.

5.39 The SET Command

The SET command is used to set a trap in the user task at a specified location.

Syntax:

SET trap

trap specifies the address at which the Symbolic Debugger sets a trap.

Response:

An error message informs the user if:

- The trap address is not specified.
- The specified address is already a trap address.
- The specified address violates an address restriction.
- Debug's trap table is full and thus no more traps can be set until a trap is deleted.
- "trap" is not on a word boundary.

The user instruction at the specified trap address is replaced by a trap instruction (SVC 1,X'66').

The Symbolic Debugger then prompts for commands to be placed in the trap list (i.e., deferred commands).

The user can enter any Symbolic Debugger command in a trap list. All commands placed in the trap list are checked for validity before they are actually stored in the trap list except for the commands CLEAR, FORMAT, MODE and SHOW. These commands will be validated only when they are to be executed. If a command is invalid, the Symbolic Debugger will write an error message and issue another prompt.

A nested trap list occurs if a user enters a SET command in a trap list. This means the second trap is set only when the first trap is encountered. Nesting can continue as far as the user desires, however, there must be a terminator for each SET command in the nested trap list. Refer to Section 4.6 for a detailed description of nested traps.

To terminate a trap list, enter one of the following commands: BREAK, END, EXIT, FILE, GO, TRACE, TRACK, or WATCH.

5.40 The SHOW Command

The SHOW command is used to display current base definitions, trap addresses, option settings, or symbols.

Syntax:

SHOW [BASES | TRAPS | OPTIONS | SYMBOLS]

BASES	displays the current definitions of all special bases and user bases.
TRAPS	displays all trap addresses.
OPTIONS	displays the settings of the options controlled by the following commands: ABSOLUTE/RELATIVE RUN/STEP FORMAT MODE
SYMBOLS	displays all symbols defined in the default program (i.e., program name established by the most recent PGM command). If there is no default program name established, all global symbols are displayed.

If no parameters are specified, all displays are produced.

Response:

An error message informs the user if any argument but BASES, TRAPS, OPTIONS, or SYMBOLS is used.

5.41 The SNAP Command

The SNAP command is used to write the contents of a range of logical addresses to the file or device assigned to lfc #OT. The format is a side-by-side hexadecimal and ASCII display.

This command is also a default (implied) command. Any expression entered without a command verb performs as if it were preceded by SNAP. If a carriage return without a command verb or expression is entered, the Symbolic Debugger will snap the address following the last address snapped. If no addresses were snapped, the Symbolic Debugger will snap the contents of memory starting at absolute address zero.

Syntax:

SNAP [low][,high]

low — specifies the first address to snap. If not specified, the snap begins at the address following the last address snapped or at absolute zero if no previous address was snapped

high — specifies the last address to snap. If not specified, only the single word at the low address is snapped. Bits 30 and 31 are ignored and assumed to be zero.

Response:

The specified memory contents are written to lfc #OT.

The special character period (.) is set at the completion of this command (refer to Section 4.16.7 Period (.)).

5.42 The STATUS Command

The STATUS command is used to display a status report indicating the user PSD and the general purpose registers for the address contained in the program counter.

Syntax:

STATUS

Response:

The Symbolic Debugger displays a status report on the terminal.

5.43 The STEP Command

The STEP command is used to set step mode.

This allows a single step trace or track through the execution of each instruction in the user task.

Syntax:

STEP

Reponse:

Until a RUN command is issued, all TRACE and TRACK commands will pause after each instruction displayed so the user can inspect each instruction and its results before the next instruction is executed.

STEP is ignored in batch.

5.44 The TIME Command

The TIME command is used to display the current date and time of day.

Syntax:

TIME

Response:

The Symbolic Debugger displays the calendar date as stored in the Communications Region (C.DATE) and the time of day as returned by the M.TDAY service.

5.45 The TRACE Command

The TRACE command is used to execute and display each user instruction and its results. To trace only branching instructions, use the TRACK command.

Syntax:

TRACE [start][,stop]

start — specifies the address of the first user instruction to be executed. If not specified, the special base $ (bits 13-31 of the user PSD) is used.

stop — specifies the address of the last user instruction to be traced. If not specified, the trace continues as described below.

Response:

The Debugger executes user instructions beginning at the specified start address and displays each instruction with its operands in an Assembler-like format. The instruction results are displayed on the right-hand side of the output.

In Step mode, the Symbolic Debugger pauses after each instruction is executed or simulated and waits for a 1-character response from the user. To proceed to the next instruction, enter only a carriage return. Any other response terminates TRACE. If the Symbolic Debugger is in Run mode, TRACE does not pause after each instruction but proceeds immediately to the next instruction; thus the only opportunity to stop the display is at the end of each screen. Note that in batch, TRACE functions as if a RUN command were in effect.

This process continues until one of the following occurs:

- An instruction has been fetched, executed, and displayed from the specified stop address. The user context indicates that the instruction has been executed, as shown in the status report announcing trace termination.

- A user instruction is aborted by a privilege violation or a map fault. The Symbolic Debugger executes most user instructions by transferring control to the user task one instruction at a time. When these instructions execute, it is as if the user had entered "GO a,b" where a is the address of an instruction and b is the address of the next instruction (logically next, not necessarily a+1W). Any abort condition caused by such instructions is reported as it would be after a GO command and the trace is terminated. The user context is reported in a status report.

- The Symbolic Debugger fetches an instruction that breaks the trace (refer to the list of instructions at the end of this section). The instruction is displayed and TRACE is terminated. The user context still points to the untraceable instruction, as shown in the status report announcing trace termination.

- The address of the next instruction to be fetched would violate an address restriction. No instruction is displayed, the trace is terminated, and the user context points to the bad address as shown in the status report announcing trace termination.

If the last control transfer to the Symbolic Debugger is caused by a trap, and the starting address is $ (the user PSD), the user instruction replaced by the trap instruction at $ is traced as if it were at $, and the trace is continued.

An error message informs the user if the starting address violates an address restriction, if the starting address is greater than the stop address, or if the start and/or stop address is an odd address.

TRACE is a trap list terminator.

The following Assembly Language instructions will cause a trace to stop (returning control to the Symbolic Debugger):

AI
BEI
BRI
CD
CEMA
DAI
DI
EI
EWCS
HALT
JWCS
LEM
LPCM
LPSD
RDST
RI
RWCS
SEM
SCPU
TD
TMTR
TPR
TRP
UEI
WAIT
WWCS
All undefined opcodes

5.46 The TRACK Command

The TRACK command functions exactly like TRACE, except that it displays only instructions that result in a change in the flow of control.

Syntax:

TRACK [start][,stop]

start — specifies the address of the first user instruction to be executed. If not specified, the special base $ (bits 13-31 of the user PSD) is used.

stop — specifies the address of the last user instruction to be executed. If not specified, the track is continued as described for TRACE.

Response:

TRACK functions exactly like TRACE, except only instructions which actually cause a branch are displayed (BCT, TRSW, LPSD, etc.).

5.47 The WATCH Command

The WATCH command functions like TRACE, but does not display instructions. It is used to detect erroneous branches into areas such as extended address space or MPX-32.

Syntax:

WATCH [start][,stop]

start — specifies the address of the first user instruction to be executed. If not specified, the special base $ (bits 13-31 of the user PSD) is used.

stop — specifies the address of the last user instruction to watch. If not specified, the watch continues as described below.

Response:

The Symbolic Debugger performs a TRACE but inhibits the usual instruction display. When, as often happens in a new program, an erroneous branch is taken, it is often into an area completely out of the program (e.g., a branch to location 0). Especially in the case of a privileged task, many instructions may precede the inevitable disaster. While the system crumbles, many of the most useful hints as to the cause (e.g., register contents) are destroyed. WATCH provides a convenient means of detecting such branches when they happen without all the terminal output caused by TRACE or TRACK.

5.48 The X (Hexadecimal) Command

The X command is used to evaluate and display the expression's value in hexadecimal format.

Syntax:

X expr

expr specifies the expression to be displayed in hexadecimal.

Response:

The hexadecimal equivalent of the expression is displayed.

Note:

If the expression is not specified, the Symbolic Debugger exits (refer to Section 5.25 the EXIT Command).

CHAPTER 6--ERROR MESSAGES

6.1 Symbolic Debugger File Assignment Error Messages

There are four error messages which may be written to the lfc #OT if an error occurs during Symbolic Debugger file assignments.

If the user did not assign enough dynamic file space for the executing task via the Cataloger's FILES directive, the message

NO FAT/FPT SPACE AVAILABLE

will be written to the lfc #OT. The user should recatalog the task specifying the correct number of dynamic file assignments. At catalog time the user task must specify the number of files required to execute the task. If Option 19 is not set for the Cataloger, the user task will not have symbols available for use in the debugging session.

If Option 19 is set for the Cataloger, the Cataloger will automatically add five files to the number the user requested. These five files are needed for symbolic debugging.

If the user did not assign enough blocking buffers for the executing task via the Cataloger's BUFFERS directive, the message

NO BLOCKING BUFFER AVAILABLE

will be written to the lfc #OT. The user should recatalog the task specifying the correct number of blocking buffers. At catalog time the user task must specify the number of buffers required to execute the task.

If option 19 is not set for the Cataloger, the user task will not have symbolic support during debugging. If option 19 is set for the Cataloger, the Cataloger will automatically add three buffers to the number the user requested. These three buffers are needed for symbolic debugging.

If there is not enough disc space available for the Symbolic Debugger to allocate for the SLO file (1000 lines for the lfc #OT in batch mode and 300 screens of data for the temporary log file in interactive mode), the message

NO DISC SPACE AVAILABLE

will be written to the lfc #OT. The user must wait until disc space becomes available.

If the Symbolic Debugger attempts to assign the temporary log file (lfc #01) and the log file has been statically assigned by the user before accessing the Symbolic Debugger, the message

LOG FILE ALREADY ALLOCATED

will be written to the lfc #OT. The user must deallocate the user defined log file assignment (no static file assignments are allowed) to allow the Symbolic Debugger to assign the log file to its default assignment.

6.2 Addressing Error Messages

There are eleven error messages which may be written to the lfc #OT if an invalid address is specified in a command.

If the user enters the CM (change memory) command without specifying the address to be changed, the message

ADDRESS MISSING

will be written to the lfc #OT. The user should re-enter the command specifying the address to be changed.

If the user enters the CM (change memory) command without specifying the value to be placed in the address specified, the message

NO VALUE SPECIFIED

will be written to the lfc #OT. The user should re-enter the command specifying the value to replace the contents of the address specified.

If the user enters a command (which allows a range of addresses as its parameters) followed by a single address and a comma (,) but failed to enter a second address, the message

NO HIGH ADDRESS

will be written to the lfc #OT. The user should re-enter the command followed by the low and high addresses separated by a comma. If only one address is desired as the parameter, no comma should be entered.

If the user enters a command (which allows a range of addresses as its parameters) followed by a low address (first address specified) which is greater than the high address (second address specified), the message

LOW > HIGH

will be written to the lfc #OT. The user should re-enter the command insuring that the first address is lower than the second address specified.

If the user enters a command (which allows an address as its parameter) other than the CM (change memory) command followed by an address which is not within the user program's addressing space, the message

ADDRESS OUTSIDE YOUR AREA

will be written to the lfc #OT. The user should re-enter the command insuring that the address specified is within the allowable addressing space.

If the user enters the CM (change memory) command followed by an address which is not within the user program's addressing space, the message

CAN'T WRITE TO THAT ADDRESS

will be written to the lfc #OT. The user should re-enter the command insuring that the address specified is within the allowable addressing space.

If the user enters a SNAP or DUMP command followed by a range of addresses which are not within the user program's addressing space, the message

MAP HOLE

will be written to the lfc #OT. The user should re-enter the command insuring that the range of addresses is within the allowable addressing space.

If the user enters the DETACH command followed by an address which is incorrectly bounded, the message

CAN'T BRANCH TO ODD ADDRESS

will be written to the lfc #OT. The user should re-enter the command insuring that the address specified falls on the correct boundary.

If the user enters the TRACE command followed by an address which is not within the user program's addressing space, the message

CANNOT TRACE INSTRUCTION

will be written to the lfc #OT. The user should re-enter the command insuring that the address is within the allowable addressing space.

If the user enters the CR (change register) command followed by an invalid register number, the message

REG NOT 0-7

will be written to the lfc #OT. The user should re-enter the command insuring that the register number specified is zero through seven.

If the user enters the CR (change register) command followed by a valid register number and invalid or missing values to be placed in the register(s), the message

NO CHANGE VALUE

will be written to the lfc #OT. The user should re-enter the command insuring that the value(s) to be placed in the register(s) are valid 32-bit values (refer to Section 5.11 CR (Change Register) Command).

6.3 Trap Error Messages

There are eight error messages which may be written to the lfc #OT if an invalid trap or trap list command is entered.

If the user enters a SET or GO command followed by a trap address that the Symbolic Debugger can't locate because

- The trap address is not on a word boundary
- The trap address is a local symbol name and the default is to global symbols
- The trap address is a local symbol name that is not in the default local symbol table
- The trap address is a global symbol that is not in the global symbol table

The following message will be written to the lfc #OT:

TRAPS ON WORD BOUNDARIES ONLY

The user should re-enter the command insuring that the trap address is on a word boundary, and that the symbol name specified is in the default symbol table (local or global).

If the user enters a SET command with no trap address specified, the message

NO TRAP ADDRESS SPECIFIED

will be written to the lfc #OT. The user should re-enter the command specifying the address where the trap is to be set.

If the user enters a LOG or REVIEW command in a trap list, the message

NOT ALLOWED IN TRAP LIST

will be written to the lfc #OT. The user should enter any valid trap list command, or enter a trap list terminator before re-entering the LOG or REVIEW command (these are the only two commands which are not allowed in a trap list).

If the user enters a SET command followed by the address of a previously set trap, the message

ALREADY A TRAP THERE

will be written to the lfc #OT. The user should re-enter the command followed by an address at which no trap exists.

If the user enters a DELETE or LIST command followed by an address at which no trap exists, the message

NO TRAP THERE

will be written to the lfc #OT. The user should verify the trap addresses, and re-enter the command followed by a valid trap address.

If the user enters the SET command and the trap table is full (maximum number of traps are set), the message

TRAP TABLE FULL; TRAP NOT SET

will be written to the lfc #OT. The user must delete one or more traps before another trap can be set.

If the user enters the IF command outside of a trap list, the message

IMMEDIATE "IF" NOT ALLOWED

will be written to the lfc #OT. The user must set a trap and enter the IF command as the first command in the trap list if the trap is to be conditional. The IF command may not be used at any other time.

If the user enters the IF command in a trap list and it is not the first command in that trap list, the message

"IF" COMMAND OUT OF SEQUENCE

will be written to the lfc #OT. The user must reset the trap (or set a new trap) and enter the IF command as the first command in the trap list if the trap list is to be conditional.

6.4 Command Expression Error Messages

There are sixteen expression error messages which may be written to the lfc #OT if an invalid expression is entered.

If the user enters an IF or LIST command with no expression (expression is a required parameter), the message

NO EXPRESSION

will be written to the lfc #OT. The user should re-enter the command with the IF conditional expression or the expression (trap address) to be listed.

If the user enters a command followed by an expression parameter which contains paired parentheses with no value inside '( )' the message

NULL SUBEXPRESSION

will be written to the lfc #OT. The user should re-enter the command with a value inside the parentheses, or delete the parentheses.

If the user enters a command followed by an expression which contains a symbol that the Symbolic Debugger cannot locate within the default local symbol table (if PGM command was entered) or within the global symbol table, the message

UNDEFINED SYMBOL

will be written to the lfc #OT. The user should verify the default symbol table (local to a program specified in the PGM command or global symbols) by entering the SHOW SYMBOLS command. If the desired symbol is a local symbol to another program name, enter the PGM command followed by the program name which defines the desired symbol before re-entering the command.

If the user enters a command followed by an expression which contains a term that the Symbolic Debugger cannot recognize, the message

UNRECOGNIZABLE TERM

will be written to the lfc #OT. The user should re-enter the command insuring that the expression contains valid terms (refer to Sections 4.16.1 through 4.16.7 for a description of valid terms used in expressions).

If the user enters a command followed by an expression which does not contain an operator, the message

MISSING OPERATOR

will be written to the lfc #OT. The user should re-enter the command insuring that the expression contains a valid operator (refer to Section 4.15 Symbolic Debugger Command Expressions, for a description of valid operators).

Note:

This message will also be written if a decimal integer greater than ten digits is entered as a term in an expression. Decimal integers may not exceed ten digits.

If the user enters a command followed by an expression which contains an operator with only one operand, the message

DANGLING OPERATOR

will be written to the lfc #OT. The user should re-enter the command insuring that the expression contains two operands for each operator specified.

If the user enters a command followed by an expression which contains a sequence of two or more operators that are not separated by operands, the message

CONSECUTIVE OPERATORS

will be written to the lfc #OT. The user should re-enter the command insuring that the expression contains two operands for each operator specified.

If the user enters a command followed by an expression which contains a left parenthesis that is not paired with a corresponding right parenthesis or vice versa, one of the messages

UNMATCHED LEFT (
or UNMATCHED RIGHT)

will be written to the lfc #OT. The user should re-enter the command insuring that the expression contains paired left and right parentheses.

If the user enters a command followed by an expression which contains a memory contents term that (when evaluated) produces an indirect address which would cause a map fault, the message

ADDRESS WOULD CAUSE MAP FAULT

will be written to the lfc #OT. The user should re-enter the command insuring that the evaluated expression does not produce an invalid address. The contents of the memory location specified in the term can be examined through the use of the display memory commands (refer to Section 4.7 Examining Memory and Registers for a summary of the display memory commands).

If the user enters a command followed by an expression which produces an invalid effective address, the message

EFFECTIVE ADDRESS CAUSES MAP FAULT

will be written to the lfc #OT. The user should re-enter the command insuring that the evaluated expression does not produce an invalid effective address.

If the user enters a command followed by an expression which contains an invalid constant or integer, one of the following messages will be written to the lfc #OT (depending on the type of constant or integer contained in the expression).

INVALID FLOATING POINT NUMBER

or

INVALID DECIMAL NUMBER

or

INVALID HEXADECIMAL NUMBER

or

INVALID BINARY NUMBER

or

INVALID CHARACTER STRING

The user should re-enter the command insuring that the constant or integer value is valid (refer to Sections 4.16.1 and 4.16.2 for a description of valid integer and constant terms in expressions).

6.5 Base Error Messages

There are four error messages which may be written to the lfc #OT if a user base is incorrectly defined, redefined or deleted.

If the user enters the BASE command and does not specify a base name, the message

NO BASE NAME

will be written to the lfc #OT. The user should re-enter the command followed by an existing or new user defined base name.

If the user enters the BASE command followed by an invalid base name, the message

BAD BASE NAME

will be written to the lfc #OT. The user should re-enter the command insuring that a valid base name is specified. (Valid base names begin with the $ and an alphabetic character followed by one to seven alphanumeric characters).

If the user enters the BASE command followed by a Symbolic Debugger base name, the message

SPECIAL BASE NOT ALLOWED

will be written to the lfc #OT. There are special Symbolic Debugger defined base names (refer to Section 4.16.4 Bases) which may not be redefined or deleted by the BASE command. The user should re-enter the command insuring that the base name is a new or existing user defined base.

If the user enters the BASE command followed by a new base name and the base table is full, the message

BASE TABLE FULL

will be written to the lfc #OT. The user must delete one or more bases before a new base can be defined.

6.6 Command File Error Messages

There are five error messages which may be written to the lfc #OT if a command file is incorrectly accessed.

If the user enters the FILE command and does not specify a file name, the message

NO FILE NAME

will be written to the lfc #OT. The user should re-enter the command followed by a valid command file name and password (if required).

If the user enters the FILE command followed by an invalid file name, the message

NO SUCH FILE

will be written to the lfc #OT. The user should re-enter the command followed by a valid command file name and password (if required).

If the user enters the FILE command followed by a file name without a password and a password is needed, the message

FILE PASSWORD PROTECTED

will be written to the lfc #OT. The user should re-enter the command followed by the valid file name and required password.

If the user enters the FILE command followed by a file name which is more than eight characters (eight bytes) in length, the message

FILE NAME > 8 BYTES

will be written to the lfc #OT. The user should re-enter the command followed by a valid file name (not exceeding eight characters) and password (if required).

If the user enters the FILE command followed by a file name and a password which is more than eight characters (eight bytes) in length, the message

PASSWORD > 8 BYTES

will be written to the lfc #OT. The user should re-enter the command followed by a valid file name and password (not exceeding eight characters).

6.7 Command Argument Error Messages

There are nine error messages which may be written to the lfc #OT if an invalid or missing argument is entered following a command.

If the user enters the CLEAR command with an invalid or missing argument, the message

ARGUMENT SHOULD BE "BASES" OR "TRAPS"

will be written to the lfc #OT. The user should re-enter the command specifying either bases or traps as the argument.

If the user enters the SHOW command with an illegal or missing argument, the message

ARGUMENT SHOULD BE BLANK, "BASES", "OPTIONS", OR "TRAPS"

will be written to the lfc #OT. The user should re-enter the command specifying one of the valid arguments listed in the message.

If the user enters the FORMAT command with an illegal or missing argument, the message

ARGUMENT SHOULD BE "X" OR "N"

will be written to the lfc #OT. The user should re-enter the command specifying either X (hexadecimal input radix) or N (decimal input radix).

If the user enters the MODE command with an illegal or missing argument, the message

ARGUMENT SHOULD BE "NOFORTRAN", "FORTRAN", "EXTENDED", "NOEXTENDED", "LOG", "NOLOG"

will be written to the lfc #OT. The user should re-enter the command specifying one of the valid arguments listed in the message.

If the user enters the RELATIVE command with an invalid base or program name, the message

RELATIVE NAME NOT FOUND

will be written to the lfc #OT. The user should re-enter the command insuring that a valid base or program name is specified.

If the user enters the RELATIVE command with a $ or PSD as its argument, the message

CAN'T USE "$" OR "PSD"

will be written to the lfc #OT. The Symbolic Debugger special bases $ and $PSD can not be specified for relative addressing. The user should re-enter the command insuring that a valid base or program name is specified.

If the user enters the DELETE command with no argument, the message

DELETE WHAT

will be written to the lfc #OT. The user should re-enter the command insuring that the trap to be deleted is specified.

If the user enters the PGM command with an invalid program name, the message

NO SUCH PROGRAM NAME

will be written to the lfc #OT. The user should re-enter the command insuring that a valid program name (to be established as default for local symbols) is specified.

If the user enters the CC (condition code) command with an invalid value to replace the existing condition codes, the message

BAD CONDITION CODES

will be written to the lfc #OT. The user should re-enter the command insuring that the value to replace the existing condition codes is a 4-digit binary value. (To display existing condition codes, enter the CC command with no argument).

6.8 Other Error Messages

There are three other error messages which may be written to the lfc #OT if one of the following errors occur.

If the user enters the BREAK command and there is no break receiver in the user program, the message

NO USER BREAK RECEIVER

will be written to the lfc #OT. This command can only be used to transfer control to the user program's break receiver. If no break receiver exists, the command is invalid.

If the user enters an invalid command, the message

UNRECOGNIZED COMMAND

will be written to the lfc #OT. The user should verify the valid command syntax, and re-enter the command.

If the Symbolic Debugger temporary log file is filled, the message

LOG FILE IS FULL, USE "LOG" COMMAND TO OUTPUT IT

will be written to the lfc #OT. The user should enter the LOG command (to write the log file to the line printer) or the REVIEW command (to write the log file to the lfc #OT). If neither command is entered, the contents of the log file will be destroyed and a new log file started.

6.9 Abort Codes

The Symbolic Debugger has two abort codes which may be written to the lfc #OT if an abort occurs. The two abort codes are

- DB01 In batch mode, the end of the file assigned to lfc #OT has been encountered before the end of job (EOJ).

- DB02 A fatal I/O error has occurred on the lfc specified after the abort code in the abort message.

  The following are examples of fatal I/O errors:

  - The input file is not assigned and there is no default input file.
  - The output file is not assigned and there is no default output file.
  - The same lfc is assigned to both the input and output file.

The above abort codes usually refer to errors within the job control. Therefore, the job control should be examined for errors before the program code.

CHAPTER 7—SAMPLE DEBUGGING SESSIONS

7.1 Debugging Session Introduction

This chapter illustrates how to use the Symbolic Debugger. The programs which are shown are not realistic programs but, for the purpose of simplicity these programs will show some useful features in the Symbolic Debugger. In reality, programs such as these may be a subprogram or module of an entire system of subprograms or modules. Each of these separate subprograms or modules should be debugged separately then added to the stable system. This method of debugging sections of an entire system then adding the sections to the stable system is highly recommended.

The following sample programs and command keys illustrate the use and results of the debugging sessions.

7.2 Example 1: Scanning Data in a Program Loop

The sample program for this debugging session searches through a table of seven values looking for the value five. If the value is contained in the table, the program will successfully exit. Otherwise, the program will abort.

The debugging session shows how to set a trap at the beginning of a loop and build a trap list that will display the register contents for the trap address, then continue execution through the loop until the trap is encountered again. This cycle continues through each successive iteration of the loop. This allows the user to examine the register contents to insure correct program execution for each iteration of the loop.

The following subsections contain the sample program and the debugging session which demonstrate the procedure described above.

```
DBGTST                                                          VER 9.3

00001                                   PROGRAM   DBGTST
00002                               *
00003                               *       THIS PROGRAM WILL DEMONSTRATE THE SYMBOLIC CAPABILITIES
00004                               *       WHICH ARE USED THROUGH THE SYMBOLIC DEBUGGER.
00005                               *
00006                               *       THIS PROGRAM WILL SIMPLY SCAN A TABLE OF DATA ITEMS
00007                               *       LOOKING FOR A SPECIFIC ITEM.  IF THE ITEM IS FOUND
00008                               *       THEN A SUCCESSFUL EXIT IS PERFORMED.  IF THE ITEM
00009                               *       IS NOT FOUND THEN THE PROGRAM IS ABORTED.
00010                               *
00011    00000                      R0       EQU      0
00012    00001                      R1       EQU      1
00013    00002                      R2       EQU      2
00014    00003                      R3       EQU      3
00015    00004                      R4       EQU      4
00016    00005                      R5       EQU      5
00017    00006                      R6       EQU      6
00018    00007                      R7       EQU      7
00019                               *
00020    P00000  00000001           BEGTABLE DATAW    X'1'
00021    P00004  00000002                    DATAW    X'2'
00022    P00008  00000003                    DATAW    X'3'
00023    P0000C  00000004                    DATAW    X'4'
00024    P00010  00000005                    DATAW    X'5'
00025    P00014  00000006                    DATAW    X'6'
00026    P00018  00000007                    DATAW    X'7'
00027    P0001C                      ENDTABLE EQU      $               ! ESTABLISH ENDING ADDRESS OF TABLE
00028    P00020                              BOUND    1D              ! DOUBLEWORD ALIGN TASK NAME
00029    P00020  54535444           TASKNAME DATAB    C'TSTDBG  '     ! NAME OF THIS TASK
         P00024  42472020
00030                               *
00031                               *    START OF PROGRAM
00032                               *
00033    P00028  34800000   P00000  START    LA       R1,BEGTABLE     ! GET ADDR. OF BEGINNING OF TABLE
00034    P0002C  C9000005           LOOP     LI       R2,5            ! PUT IN VALUE OF ITEM SEARCHED FOR
00035    P00030  91200000    00000          CAMW     R2,0,R1         ! CHECK IF FOUND VALUE
00036    P00034  EE000051   P00050          BEQ      ENDSUCC         ! BRANCH TO END SUCCESSFULLY IF FOUND
00037                               *
00038    P00038  C8810004                   ADI      R1,1W           ! INC. TO LOOK AT NEXT ITEM IN TABLE
00039    P0003C  90800010   P0001C          CAMW     R1,ENDTABLE     ! CHECK IF AT END OF TABLE
00040    P00040  E2000031   P00030          BNE      LOOP            ! BRANCH IF NOT AT END OF TABLE
00041                               *
00042    P00044  AE800054   P00054          LW       R5,=C'ERR'      ! LOAD IN ABORT CODE
00043    P00048  AE000022   P00020          LD       R6,TASKNAME     ! LOAD IN ABORT TASK NAME
00044    P0004C  C8061056                   SVC      1,X'56'         ! ABORT THIS TASK
00045                               *
00046    P00050  C8061055           ENDSUCC SVC      1,X'55'         ! SUCCESSFUL END OF PROGRAM
         P00054  45525220
00047    P00058             P00028          END      START
*  0000  ERRORS IN DBGTST
```

7.2.2 Sample Debugging Session for program DBGTST

All commands in the sample debugging session are numbered to correspond to a key which describes the commands and responses. Each command is immediately followed by its response.

Command Key

1) The Symbolic Debugger is accessed by entering the DEBUG command in response to the TSM > prompt.

   The Symbolic Debugger responds with its identifying message.

2) The command to show the symbols (SHOW SYMBOLS) is entered.

   The default is to the global symbol table, the response displays the global symbols header and indicates that there are no global symbols in the program by displaying no symbol names.

3) The command to set the default program name and accessable symbols (PGM DBGTST) is entered.

   The default program name is set to DBGTST. The symbols local to DBGTST are accessable. There is no written response to this command.

4) The command to show symbols (SHOW SYMBOLS) is entered again (this time to display the local symbols).

   The response displays the local symbols header and all symbols local to the program DBGTST.

5) The command to set a trap at the beginning of a program loop at the location whose address is specified by the symbol name LOOP (SET LOOP) is entered.

   The trap is set at the address specified by the symbol name LOOP and the Symbolic Debugger responds with the trap list prompt (..).

6,7) The command (command 6) to snap register one (SN R1) is entered in the trap list followed by the GO command (command 7). These commands are deferred until the trap is encountered. When they are executed, R1 will be displayed and program execution resumed. Because the trap is set at the beginning of a loop, each time program execution is resumed the trap is encountered again and the trap list executed. This allows the user to insure correct program execution for each iteration of the loop. The GO command in the trap list is a trap list terminator, therefore the trap list is ended and the immediate lfc #IN prompt (.) is issued.

8) The command to begin program execution (GO) is entered. Program execution will begin at the base $ (bits 13-31 of the user PSD or the current program counter value) because no start address was specified.

   The program will begin execution. There is no written response to the GO command.

6a) The trap is encountered and the SNAP command (deferred in the trap list) is executed.

The response specifies the address contained in register one (DBGTST) followed by the contents of that address (00000001).

7a) The GO command (deferred in the trap list) is executed.

Program execution is resumed. There is no written response.

6b) The trap is encountered and the SNAP command (deferred in the trap list) is executed.

The response specifies the address contained in register one (DBGTST+4) followed by the contents of that address (00000002).

7b) The GO command (deferred in the trap list) is executed.

Program execution is resumed. There is no written response.

6c) The trap is encountered and the SNAP command (deferred in the trap list) is executed.

The response specifies the address contained in register one (DBGTST+8) followed by the contents of that address (00000003).

7c) The GO command (deferred in the trap list) is executed.

Program execution is resumed. There is no written response.

6d) The trap is encountered and the SNAP command (deferred in the trap list) is executed.

The response specifies the address contained in register one (DBGTST+C) followed by the contents of that address (00000004).

7d) The GO command (deferred in the trap list) is executed.

Program execution is resumed. There is no written response.

6e) The trap is encountered and the SNAP command (deferred in the trap list) is executed.

The response specifies the address contained in register one (DBGTST+10) followed by the contents of that address (00000005).

7e) The GO command (deferred in the trap list) is executed.

Program execution is resumed and the value five encountered therefore a successful exit from the program is performed. The Symbolic Debugger displays the status of the program at the exit address.

Debugging Session

```
1)  TSM> DEBUG DBGTST

    MPX-32 SYMBOLIC DEBUG V2.0  05/13/81,  13:00:00  TASK NAME = DBGTST
    PSW=01029828   (CC=0000)   (PC=DBGTST+28)
    REGS=00000000  00000000  00000000  00000000  ................
         00000000  00000000  00000000  00000000  ................
2)  .SHOW SYMBOLS
    GLOBAL SYMBOLS
3)  .PGM DBGTST
4)  .SHOW SYMBOLS
    SYMBOLS LOCAL TO PROGRAM #DBGTST
    START        TASKNAME    ENDSUCC      BEGTABLE     LOOP
    ENDTABLE
5)  .SET LOOP
6)  ..SN R1
7)  ..GO
8)  .GO
    TRAP @ DBGTST+30
    PSW=21029830   (CC=0100)   (PC=DBGTST+30)
    REGS=00000000  00029800  00000005  00000000  ................
         00000000  00000000  00000000  00000000  ................
6a) !SN R1
    DBGTST                    00000001                        /..../
7a) !GO
    TRAP @ DBGTST+30
    PSW=21029830   (CC=0100)   (PC=DBGTST+30)
    REGS=00000000  00029804  00000005  00000000  ................
         00000000  00000000  00000000  00000000  ................
6b) !SN R1
    DBGTST+4                   00000002                        /..../
7b) !GO
    TRAP @ DBGTST+30
    PSW=21029830   (CC=0100)   (PC=DBGTST=30)
    REGS=00000000  00029808  00000005  00000000  ................
         00000000  00000000  00000000  00000000  ................
6c) !SN R1
    DBGTST+8                   00000003                        /..../
7c) !GO
    TRAP @ DBQTST+30
    PSW=21029830   (CC=0100)   (PC=DBGTST+30)
    REGS=00000000  0002980C  00000005  00000000  ................
         00000000  00000000  00000000  00000000  ................
6d) !SN R1
    DBGTST+C                   00000004                        /..../
7d) !GO
    TRAP @ DBGTST+30
    PSW=21029830   (CC=0100)   (PC=DBGTST+30)
    REGS=00000000  00029810  00000005  00000000  ................
         00000000  00000000  00000000  00000000  ................
6e) !SN R1
    DBGTST+10                  00000005                       /..../
```

7.3 Example 2: Searching Through a Linked List

The sample program for this debugging session contains a linked list with four data words in each node. If the nodes are linked correctly, the program will successfully exit. Otherwise, the program will abort.

The debugging session shows how to establish a value for the special character, period (.), which will be used to display the node address and data words of each node. The debugging session also shows how to build a conditional trap list which will be executed only if the counter (CTR, which is used to count the number of nodes that are linked) is greater than two (a conditional trap list allows the user to execute the trap list only if a specified condition is met).

The following subsections contain the sample program and the debugging session which demonstrate the procedure described above.

```
DBGTST2                                                                    VER 9.3

00001                                          PROGRAM   DBGTST2
00002                                  *
00003                                  *       THIS PROGRAM WILL DEMONSTRATE THE SYMBOLIC CAPABILITIES
00004                                  *       WHICH ARE USED THROUGH THE SYMBOLIC DEBUGGER.
00005                                  *       THIS PROGRAM WILL ESTABLISH A LINKED LIST WHICH CAN
00006                                  *       THEN BE DISPLAYED THROUGH THE USE OF DEBUGGER COMMANDS.
00007                                  *
00008    00000                         R0      EQU     0
00009    00001                         R1      EQU     1
00010    00002                         R2      EQU     2
00011    00003                         R3      EQU     3
00012    00004                         R4      EQU     4
00013    00005                         R5      EQU     5
00014    00006                         R6      EQU     6
00015    00007                         R7      EQU     7
00016  P00000  00000014   P00014       LINKSTRT ACW    NODE1          ! INITIALIZE BEGINNING OF LIST
00017  P00004  00000000                NODE4   DATAW   0              ! FORWARD POINTER = 0 => NO MORE NODE
00018  P00008  4E4F4445                        DATAW   C'NODE'        ! 1ST WORD OF DATA IN NODE 4
00019  P0000C  34202020                        DATAW   C'4   '        ! 2ND WORD OF DATA IN NODE 4
00020  P00010  00000004                        DATAW   4              ! 3RD WORD OF DATA IN NODE 4
00021  P00014  00000034   P00034       NODE1   ACW     NODE2          ! FORWARD POINTER IN NODE 1
00022  P00018  4E4F4445                        DATAW   C'NODE'        ! 1ST WORD OF DATA IN NODE 1
00023  P0001C  31202020                        DATAW   C'1   '        ! 2ND WORD OF DATA IN NODE 1
00024  P00020  00000001                        DATAW   1              ! 3RD WORD OF DATA IN NODE 1
00025  P00024  00000004   P00004       NODE3   ACW     NODE4          ! FORWARD POINTER IN NODE 3
00026  P00028  4E4F4445                        DATAW   C'NODE'        ! 1ST WORD OF DATA IN NODE 3
00027  P0002C  33202020                        DATAW   C'3   '        ! 2ND WORD OF DATA IN NODE 3
00028  P00030  00000003                        DATAW   3              ! 3RD WORD OF DATA IN NODE 3
00029  P00034  00000024   P00024       NODE2   ACW     NODE3          ! FORWARD POINTER IN NODE 2
00030  P00038  4E4F4445                        DATAW   C'NODE'        ! 1ST WORD OF DATA IN NODE 2
00031  P0003C  32202020                        DATAW   C'2   '        ! 2ND WORD OF DATA IN NODE 2
00032  P00040  00000002                        DATAW   2              ! 3RD WORD OF DATA IN NODE 2
00033  P00044  00000000                CTR     DATAW   0              ! INITIALIZE COUNTER OF NUMBER OF
00034  P00048                                  BOUND   1D             ! DOUBLEWORD ALIGN TASK NAME
00035  P00048  54535444                TASKNAME DATAB  C'TSTDBG2 '    ! NAME OF THIS TASK
       P0004C  42473220
00036                                  *
00037                                  *    START OF PROGRAM
00038                                  *
00039  P00050  AC800000   P00000       START   LW      R1,LINKSTRT    ! LOAD IN POINTER TO LINKED LIST
00040  P00054  C8850000                        CI      R1,0           ! CHECK IF LINKED LIST IS NULL
00041  P00058  F2000069   P00068               BNE     LOOP           ! BRANCH IF LINKED LIST IS NOT NULL
00042                                  *
00043  P0005C  AE800078   P00078               LW      R5,=C'ERR'     ! LOAD IN ABORT CODE
00044  P00060  AF00004A   P00048               LD      R6,TASKNAME    ! LOAD IN ABORT TASK NAME
00045  P00064  C8061056                        SVC     1,X'56'        ! ABORT THIS TASK
00046                                  *
00047  P00068  A3880047   P00047       LOOP    ABM     31,CTR         ! INC. CTR. OF NUMBER OF NODES FOUND
00048  P0006C  AC800000    00000               LW      R1,0,R1        ! LOAD IN POINTER TO NEXT NODE
00049  P00070  F2000069   P00068               BNE     LOOP           ! BRANCH IF NOT AT END OF LIST
00050                                  *
00051  P00074  C8061055                ENDSUCC SVC     1,X'55'        ! SUCCESSFUL END OF PROGRAM
       P00078  45525220
00052  P0007C                   P00050          END     START
*  0000  ERRORS IN DBGTST2
```

7.3.2 Sample Debugging Session for program DBGTST2

All commands in the sample debugging session are numbered to corresponding to a key which describes the commands and responses. Each command is immediately followed by its response.

Command Key

1) The Symbolic Debugger is accessed by entering the DEBUG command in response to the TSM> prompt.

   The Symbolic Debugger responds with its identifying message.

2) The command to set the default program name and the accessable symbols (PGM #DBGTST2) is entered.

   The default program name is set to DBGTST2. The symbols local to DBGTST2 are now accessable. There is no written response to this command.

3) The command to snap the address which contains the address of the 1st node in the linked list (SN LINKSTRT) is entered.

   The response specifies the address to be snapped (DBSTST2) and the contents of that address (00029814). The address is displayed in the program name plus offset format. In this example, there is no offset. LINKSTRT is located at the start address, program name +0 (the +0 is not displayed). The contents of the snapped address (00029814) specifies the address of the first node in the link. The value of the special character period (.) is set to the value of LINKSTRT (period (.) can be entered in place of the last address specified in a memory related command). The value of period (.) is reset each time a memory related command is entered.

4) The command to execute the commands in a command file (FILE FILINK) is entered.

   The response to the FILE command is to execute the commands in the command file (FILINK) specified. The commands in command file FILINK are

```
SN C(.), C(.) + 0C
SN . - 0C
```

5) The first command in the command file is the SNAP command (SN C(.), C(.) + 0C). The range of addresses to be snapped specifies the contents of the special character period (.) (which is set to the address LINKSTART) through the contents of period (.) plus 12 decimal bytes. Prior to the execution of this command the contents of period (.) is the address of the first node in the linked list.

   The response specifies the address of the first node in the linked list (DBGTST2+14) and the contents of the first node (00029834 4E4F4445 31202020 00000001 /...4NODE1 ..../). The period (.) is now set to the address of the third data word of the first node (00029820, which was the last address in the specified range).

6) The second and last command in the command file is also the SNAP command (SN.-0C). The address to be snapped specifies the period (.) minus 12 decimal bytes (.-0C).

The response specifies the address of the first node in the linked list (DBGTST2+14) and the contents of that address (00029834). The contents (00029834) specifies the address of the second node. This command is entered to reset the value of period (.) to the first word of the node just displayed which contains the address of the next node.

7) The FILE command is entered.

The commands in the command file (FILINK) are accessed.

8) The SNAP command (specified in the command file) is executed. The range of addresses to be snapped specifies the contents of period (.) (which is set to the address of the first word of the first node in the linked list) through the contents of period (.) plus 12 decimal bytes. Prior to the execution of this command, the contents of period (.) is the address of the second node.

The response specifies the address of the second node (DBGTST2+34) and the contents of the second node (00029824 4E4F4445 32202020 00000002 /...$ NODE 2 ..../). The period (.) is now set to the address of the third data word of the second node (00029840).

9) The second SNAP command (specified in the command file) is executed. The address to be snapped specifies the period (.) minus 12 decimal bytes (.-0C).

The response specifies the address of the second node in the linked list (DBGTST2+34) and the contents of that address (00029824). The contents (00029824) specifies the address of the third node in the linked list. The special character period (.) is now set to the address of the second node (00029834).

10) The FILE Command is entered again.

The commands in the command file (FILINK) are accessed.

11) The SNAP command (specified in the command file) is executed. The range of addresses to be snapped specifies the contents of period (.) (which is set to the address of the first word of the second node in the linked list) through the contents of period (.) plus 12 decimal bytes. Prior to the execution of this command the contents of period (.) is the address of the third node.

The response specifies the address of the third node (DBGTST+24) and the contents of the third node (00029804 4E4F4445 33202020 00000003 /.... NODE 3 ..../). The period has the value of the address of the third data word of the third node (00029830).

12) The second SNAP command (specified in the command file) is executed. The address to be snapped specifies the period (.) minus 12 decimal bytes (.-0C).

The response specifies the address of the third node in the linked list (DBGTST2+24) and the contents of that address (00029804). The contents (00029804) specify the address of the fourth node in the linked list. The period (.) is now set to the address of the third node (00029824).

13) The FILE command is entered again.

The commands in the command file (FILINK) are accessed.

14) The SNAP command (specified in the command file) is executed. The range of addresses to be snapped specifies the contents of period (.) (which is set to the address of the first word of the third node in the linked list) through the contents of period (.) plus 12 decimal bytes. Prior to the execution of this command the contents of period (.) is the address of the fourth node.

The response specifies the address of the fourth node (DBGTST+4) and the contents of the fourth node (00000000 4E4F4445 34202020 00000004 /.... NODE 4 ..../). The period (.) is now set to the address of the third data word of the fourth node (00029810).

15) The second SNAP command (specified in the command file) is executed. The address to be snapped specifies the period (.) minus 12 decimal bytes (.-0C).

The response specifies the address of the fourth node in the linked list (DBGTST2+4) and the contents of the address (00000000). The contents (00000000) specifies the start address of the program (this indicates that there are no other nodes). The period (.) is now set to the address of the fourth node (00029804). All nodes and their data words have been examined.

16) The command to set a trap at the beginning of a program loop (SET LOOP) is entered.

The trap is set at the address specified by the symbol name LOOP and the Symbolic Debugger responds with the trap list prompt (..).

17) The command to establish a conditional trap list (IF C(CTR) > 2) is entered. The argument specifies that the trap list at location LOOP will be executed only if the contents of the local symbol CTR is greater than two. The conditional command (IF) is deferred (along with commands 18 and 19) until the trap at LOOP is encountered.

18) The command to snap the start address of the program (SNAP LINKSTRT) is entered. This command is deferred until the trap is encountered.

19) The command to execute the commands in a command file (FILE FILINK) is entered. The commands in the command file (FILINK) specified will be executed when the trap is encountered. The FILE command is a trap list terminator, therefore the trap list is ended.

20) The command to begin program execution (GO) is entered. Program execution will begin at the base $ (bits 13-31 of the user PSD or the current program counter value) because no start address was specified.

The program will begin execution. There is no written response to the GO command.

17a) The IF command (deferred in the trap list) is executed.

The response specifies the IF conditional statement and the current status at the address where the condition became true. The relational value of C(CTR) 2 is equal to one (the condition is true).

18a) The SNAP command (deferred in the trap list) is executed.

The response specifies the address to be snapped (DGBTST2), which is the same address as the symbol name (LINKSTRT), and the contents of that address (00029814).

19a) The FILE command (deferred in the trap list) is executed.

The commands in the command file (FILINK) are accessed.

The SNAP command (specified in the command file) is executed. The range of addresses to be snapped specifies the contents of the period (.) (which is set to the start address of the program) through the contents of period (.) plus 12 decimal bytes. The contents of period (.) is the address of the first node in the linked list.

The response specifies the address of the first node in the linked list (DBGTST2+14) and the contents of the first node (00029834 4E4F4445 31202020 00000001 /...4NODE1 ..../). The period (.) is now set to the address of the third data word of the first node (00029820).

The second SNAP command (specified in the command file) is executed. The address to be snapped specifies the period (.) minus 12 decimal bytes (.-0C).

The response specifies the address of the first node in the linked list (DBGTST2+14) and the contents of that address (00029834). The contents (00029834) specifies the address of the second node. The period is now set to the address of the first node in the linked list (00029814).

Debugging Session

```
1)    TSM> DEBUG DBGTST2

      MPX-32 SYMBOLIC DEBUG V2.0 05/13/81, 13:30:00 TASK NAME = DBGTST2
      PSW=01029850    (CC=0000)    (PC=DBGTST2+50)
      REGS=00000000  00000000  00000000  00000000 ................
           00000000  00000000  00000000  00000000 ................

2)    .PGM #DBGTST2
3)    .SN LINKSTRT
      DBGTST2          00029814                                 /..../
4)    .FILE FILINK
5)    SN  C(.),C(.)+0C
      DBGTST2+14       00029834  4E4F4445  31202020  00000001 /...4NODE1..../
6)    SN . -0C
      DBGTST2+14       00029834                                 /...4/
7)    .FILE FILINK
8)    SN C(.),C(.)+0C
      DBGTST2+34       00029824  4E4F4445  32202020  00000002 /...$NODE2..../
9)    SN .-0C
      DBGTST2+34       00029824                                 /...$/
10)   .FILE FILINK
11)   SN C(.),C(.)+0C
      DBGTST2+24       00029804  4E4F4445  33202020  00000003 /....NODE3..../
12)   SN .-0C
      DBGTST2+24       00029804                                 /..../
13)   .FILE FILINK
14)   SN C(.),C(.)+0C
      DBGTST2+4        00000000  4E4F4445  34202020  00000004 /....NODE4..../
15)   SN .-0C
      DBGTST2+4        00000000                                 /..../
16)   SET LOOP
17)   ..IF C(CTR)>2
18)   ..SN LINKSTRT
19)   ..FILE FILINK
20)   .GO
      TRAP @ DBGTST2+68
17a)  IF C(CTR) >  2
      'IF' VALUE =    00000001
      PSW=21029868    (CC=0100)  (PC=DBGTST2+68)
      REGS=00000000  00029804  00000000  00000000 ................
      00000000       00000000  00000000  00000000 ................
18a)  !SN LINKSTRT
      DBGTST2          00029814                                 /..../
19a)  !FILE FILINK
      SN C(.),C(.)+0C
      DBGTST2+14       00029834  4E4F4445  31202020  00000001 /...4NODE1..../
      SN .-0C
      DBGTST2+14       00029834                                 /...4/
```