TAM-1001-008

DMS-100 Family

DEBUG

Technical Assistance Manual

BCS36 and up   Standard 03.02   December 1993

northern telecom

DMS-100 Family

DEBUG
Technical Assistance Manual

Publication number: TAM-1001-008
Product release: BCS36 and up
Document release: Standard 03.02
Date: December 1993

Publication history

December 1993

BCS36 Standard 03.02 release of this document changed various commands, parameters, and variables

October 1993

BCS36 Preliminary 03.01 release of this document to verification office

September 1991

BCS32 Standard 02.03 changed command format and made changes to various commands, parameters, and variables

March 1991

BCS32 Standard 02.02 restructured document

Contents

About this document ix

Applicability of this document ix
Software Identification x
    How to identify the software in your office x
Reason for Reissue x
References x
    Where to find information x
    Prerequisite References x
    Informative References x
NT and BNR trademarks and the products they represent xi
Effect on Switch xi
What precautionary messages indicate xi
How commands, parameters, and responses are represented xii
Syntax Notes xiv

DEBUG utility 1-1

DEBUG access level 1-1
When to use DEBUG 1-1
    Displaying program and data store 1-1
    Setting tracepoints 1-1
    Example use of DEBUG 1-2
Restrictions and limitations 1-2
Informative references 1-3

Determining terminal and node numbers 2-1

Node and terminal numbers 2-1
Terminal 0 2-1
Calculations 2-1

DEBUG commands and subcommands 3-1

Accessing DEBUG 3-1
CI level commands 3-2
DEBUG execs ($READ command) 3-2
Adding comments 3-3
DEBUG command description 3-4
    ACTIVATE 3-4
    Responses 3-4
    Usage notes 3-6
    ALLOC 3-8
    Responses 3-8

BIND 3-11
Responses 3-11
CLEAR 3-15
Responses 3-15
Usage notes 3-16
CPID 3-17
Responses 3-17
Usage notes 3-18
DEACTIVATE 3-19
Responses 3-19
DEFINE 3-22
Responses 3-22
Usage notes 3-28
DELBIND 3-29
Responses 3-29
Usage notes 3-30
DISPLAY (data store) 3-31
Responses 3-33
Usage notes 3-34
DISPLAY PS (program store) 3-35
Responses 3-35
EXTEND 3-39
Responses 3-39
FDEFINE 3-42
Responses 3-42
Usage notes 3-47
GO 3-48
Responses 3-48
HELP 3-49
Responses 3-50
LIMIT 3-52
Responses 3-52
LISTBIND 3-54
Responses 3-54
PRINT 3-55
Responses 3-55
QUIT 3-58
Responses 3-58
RESET 3-59
Responses 3-59
Usage notes 3-60
SHELP 3-61
Responses 3-62
STATUS 3-64
Responses 3-64
Tracepoint subcommands 3-67
ACTIVATE 3-67
Responses 3-67
Usage notes 3-68
BIND 3-69
Responses 3-69

Usage notes 3-70
DEACTIVATE 3-71
Responses 3-71
DISPLAY CCBPTR, ECCBPTR, CDBPTR, and RUNPPTR 3-73
Responses 3-73
DISPLAY (data store) 3-74
Responses 3-76
Usage notes 3-77
EXITIF 3-78
Responses 3-79
Usage notes 3-80
SAVECCB 3-81
Responses 3-81
Usage notes 3-81
SAVECDB 3-85
Responses 3-85
Usage notes 3-85
SAVEEXT 3-87
Responses 3-87
Usage notes 3-88
TIMESTAMP 3-89
Responses 3-89
TRACEBACK 3-90
Responses 3-90
Usage notes 3-91

Displaying data store 4-1

Format of DEBUG output from DISPLAY command 4-1
Specifying an offset in a module 4-4
Displaying contents of an absolute address 4-6
Displaying a base register 4-6
Displaying the expression stack 4-7
Displaying local variables and parameters 4-8
Output of multiple items 4-10
Using the dereference operator @ 4-11
Displaying fields in a structure 4-12
Bit field selection 4-14
Indexing a table 4-16
- Word size table indexing 4-16
- Bit table indexing 4-17

Indexing a descriptor 4-19
- Indexing a word descriptor 4-19
- Indexing a bit descriptor 4-22

Examples of displaying data 4-23
- Displaying update parameters 4-23
- Displaying queue problems 4-24
- Displaying a field in the CCB 4-25

Differences between DEBUG on the NT40 and SuperNode 5-1

Program store 5-1
Data store 5-3

Expression stack 5-3
Local variables 5-3
Registers 5-4
Tables 5-5
Descriptors 5-6
Pointers and addresses 5-9
Tracepoint procedure code offset 5-10
EXITIFs 5-11

List of terms 6-1

List of figures

Figure 2-1	CONVERT command example 2-2
Figure 2-2	QDN command example 2-3
Figure 2-3	QLEN command example 2-4
Figure 2-4	QUERYPM command example 2-4
Figure 2-5	NODENO subcommand example 2-5
Figure 2-6	DCM carrier and timeslot to terminal number cross-reference 2-6
Figure 3-1	Issuing CI commands from within DEBUG 3-2
Figure 3-2	DEBUG exec 3-3
Figure 3-3	Adding comments 3-3
Figure 4-1	Using different output formats 4-3
Figure 4-2	Displaying data using offsets 4-5
Figure 4-3	Displaying contents of an absolute address 4-6
Figure 4-4	Displaying data using a base register 4-7
Figure 4-5	Displaying the expression stack 4-8
Figure 4-6	Displaying the expression stack 4-8
Figure 4-7	Displaying local variables and parameters 4-10
Figure 4-8	Displaying multiple items 4-11
Figure 4-9	Displaying the contents of a location 4-12
Figure 4-10	TRUNK_MEMBERS_PROT display 4-14
Figure 4-11	Displaying fields in a structure 4-14
Figure 4-12	Bits 9-12 of Second Word of Protected Store 4-15
Figure 4-13	Displaying data using bit selection 4-16
Figure 4-14	Word size table indexing 4-17
Figure 4-15	Bit Table Indexing 4-18
Figure 4-16	Displaying the third item of table 4-20
Figure 4-17	Displaying the third item of bit table 4-22
Figure 4-18	Recursive dereferencing 4-25
Figure 4-19	Displaying a field of the CCB 4-26
Figure 5-1	Displaying program store on NT40 versus SuperNode 5-2
Figure 5-2	Displaying data store on NT40 versus SuperNode 5-3
Figure 5-3	Displaying data store (bit fields) on SuperNode 5-3
Figure 5-4	Locals in the SuperNode stack 5-4
Figure 5-5	Displaying tables on the NT40 and SuperNode 5-6
Figure 5-6	Descriptor layout on the NT40 5-6
Figure 5-7	Descriptor layout on SuperNode 5-7
Figure 5-8	Displaying descriptors on NT40 and SuperNode 5-8
Figure 5-9	Descriptors on NT40 versus SuperNode 5-9
Figure 5-10	Pointers on NT40 versus SuperNode 5-10

Figure 5-11 EXITIFs on NT40 versus SuperNode 5-11

List of tables

Table 5-1 Important register equivalencies 5-5

About this document

This Technical Assistance Manual (TAM) describes the DEBUG software tool and provides instructions for the proper use of the tool.

This TAM is intended for use by skilled maintenance personnel that have a thorough knowledge of the DMS-100 software and experience in maintaining the DMS-100.

The DEBUG utility is a Central Control (CC) debugging tool used to display the contents of specified program and data store locations. By setting tracepoints, you can display data or make procedure tracebacks at the interrupt level when the tracepoint is encountered by a running process. Data is saved in a buffer for later display.

Data store is displayed in hexadecimal, decimal, or symbolic format (for any type known to the Data Dictionary). Symbolic formats include DESC, CHAR, BOOL, and many others.

Program store is displayed in assembler code format.

Applicability of this document

Northern Telecom (NT) software releases are referred to as batch change supplements (BCS) and are identified by a number, for example, BCS29.

This document applies to DMS-100 Family offices that have BCS36. Unless the document is revised, it also applies to offices that have software releases greater than BCS36.

More than one version of this document may exist. To determine which version applies to the BCS in your office, check the release information in *Northern Telecom publications master index, 297-1001-001*.

This manual is for the DEBUG version that runs on the NT40. The version that runs on the DMS SuperNode is slightly different. Refer to Chapter 5 on page 5-1 for an explanation of some of the differences in the versions.

The information contained in this TAM is applicable to offices having Batch Change Supplement release 36 (BCS36) software. It is also applicable to offices having a BCS release greater than 36 unless reissued.

Software Identification

How to identify the software in your office

Software applicable to a specific DMS-100 Family office is identified by a BCS release number and by Northern Telecom (NT) Product Engineering Codes (PEC). The significance of the BCS number and the PEC is described in 297-1001-450 (section 450/32) and in the Office Feature Record D-190.

For BCS22 software loads and greater, a display of the BCS number and PEC for the NT feature packages available in a specific office can be obtained by entering the following command string at the Maintenance and Administration Position (MAP):

PATCHER;INFORM LIST;LEAVE

Reason for Reissue

This TAM is reissued to make minor modifications to several DEBUG command descriptions.

Changes or additions to this document are indicated by change bars (|) along the left margin.

References

Where to find information

References listed as prerequisites are essential for an understanding of this TAM. Those listed as informative contain detailed information concerning other items mentioned in this TAM, but are not essential. References are inserted at the appropriate places in the text.

Note: The documents listed may exist in more than one version. See 297-1001-001 to determine the release code of the version compatible with a specific release of software.

Prerequisite References

Document	Title
297-1001-100	*System Description*

Informative References

Document	Title
TAM-1001-000	*Technical Assistance Manual Index of Documents*
297-1001-001	*Master Index of Practices*

Document	Title
297-1001-106	*Maintenance System DMS-100/200*
297-1001-107	*Maintenance and Administration Tools Description*
297-1001-110	*Maintenance and Administration Position*
297-1001-129	*Input/Output System Reference Manual*

NT and BNR trademarks and the products they represent

The following chart lists all NT and BNR trademarks that occur in this document, and associates them with the products they represent.

Trademark	Product
DMS	*Digital multiplex system* telephone switching equipment
DMS SuperNode	telecommunications switching equipment
MAP	*Maintenance and administration position* telephone communication equipment

Effect on Switch

DEBUG has minimal impact on switch operation. The impact is dependent upon the number of tracepoints being hit, where the tracepoints are set, and the amount of data being stored.

What precautionary messages indicate

In this document, caution, danger and warning messages indicate potential risks, as identified in the following chart.

Message	Significance
CAUTION	Possibility of service interruption or degradation
DANGER	Possibility of personal injury
WARNING	Possibility of equipment damage

Examples of the precautionary messages follow.

CAUTION
Calls are dropped when line group controller is busied.
Manually removing the line group controller from service removes all its subtending peripheral modules from service. All calls in progress are dropped.

DANGER
Risk of electrocution
The inverter contains high voltage lines. Do not open the front panel of the inverter unless fuses F1, F2, and F3 have been removed first. Until these fuses are removed, the high voltage lines inside the inverter are active, and you risk being electrocuted.

WARNING
Backplane connector pins may become damaged.
Use light thumb pressure to align the card with the connectors. Next use the levers to seat the card into the connectors. Failure to align the card first may result in bending of backplane connector pins.

How commands, parameters, and responses are represented

In this TAM, a uniform system of notation is used to illustrate system commands and responses. It shows the order in which command elements appear, the punctuation, and the options. Where the conventions are not used, an explanation is given in the text.

In this document, commands, parameters, and responses are represented according to the following conventions.

Input prompt (>)

An input prompt (>) indicates that the information that follows is a command.

Type the command that follows the input prompt and press `ENTER`.

COMMAND	

Capital letters or special characters

Capital letters show constants, commands, or keywords that the system accepts when entered as written.

Enter the command or fixed parameter exactly as it appears on the page.

Lowercase letters
Lowercase letters show a user- or system-supplied parameter. Definitions are given for each parameter.

For commands and parameters, enter the letters or numbers that the variable represents. In most instances, the name that is used for the variable indicates clearly what you must enter. Where it does not, further explanations are provided.

In responses (which are presented in capital letters), lowercase letters represent a range of values.

Brackets [ ] or [ ]
Brackets enclose optional parameters. A vertical list enclosed in brackets means that one or more of the parameters may be selected.

Underlined parameter
Is a default. If no choice is entered, the system acts as though the underlined parameter had been entered.

Underscore connecting words
Means the words are to be treated as one item, for example, pm_type or #_one_two.

•••
Indicates repeated steps or items.

In addition, the following conventions are used.

n (lowercase n)
Is a number from 0 to 9.

a (lowercase a)
Is a letter from A to Z.

h (lowercase h)
Is a hexadecimal integer from 0 to F.

The following example illustrates the command syntax that is used in this document.

Examples of command syntax used in this document	
Step	**Action**
1 *input>*	Post the card in the inactive unit. **>POST unit_no card_no state**
parameters>	where unit_no is the number of the inactive unit (0 or 1) card_no is the number of the card you replaced (22-27) state is the state of the unit in which you wish to re-place the card (Insv, SysB, ManB or Offl)
Example input>	For example: **>POST 7 1 INSV**
Example output>	`CARD 7 IS POSTED IN UNIT 1 OF MSB16`

Syntax Notes

In the examples in this manual, the commands are preceded with the prompt >. The next lines after the command, until a blank line, give the output from the switch. For example:

```
>DI LOGS: PR.3A          This line contains the command.
016F48:   5645           This line contains the output.
```

DEBUG utility

DEBUG access level

Access the DEBUG utility by entering the DEBUG command from any level of the MAP. Refer to Chapter 3 on page 3-1 for correct command syntax.

When to use DEBUG

DEBUG is a multiuser tool. The DEBUG software includes commands for displaying data and program store using PROTEL-like expressions. You may execute the commands at the terminal or define them for execution at any point in a running PROTEL program.

Displaying program and data store

DEBUG allows you to display program store as well as private, shared, and protected data store. Data can be referenced as the PROTEL type; for example, a DESC of TABLE of DESC. In addition to displaying this type of structure, DEBUG easily displays simple variables in hexadecimal, decimal, character or Data Dictionary format. DEBUG also displays program store in assembler code format.

Setting tracepoints

Maintenance personnel can set tracepoints in order to analyze run-time data. Because DEBUG is used in live switches, the program is not halted when a tracepoint is encountered. An interrupt is generated to snapshot and save the data specified in the tracepoint subcommands. When a tracepoint is hit, data (for example, call condense blocks or call header blocks) can be collected for display at a later time.

If the tracepoint is hit often, then the data is overwritten. If the commands SAVECDB and SAVECCB (see SAVECDB on page 3-85 and SAVECCB on page 3-81) or the output formats CDB or CCB (see Format of DEBUG output from DISPLAY command on page 4-1) are used in the tracepoint subcommand list, then only four tracepoint hits are stored before the oldest one is overwritten, unless the buffers have been enlarged by the ALLOCate command (see ALLOC on page 3-8). If you are unsure as to whether or not the data will be overwritten, then print out the data regularly or use the LIMIT command (see LIMIT on page 3-52).

It is possible to overcome the problem of overwriting data by using the EXITIF tracepoint subcommand (see EXITIF on page 3-78) or by having the tracepoint (de)activate itself or another tracepoint.

DEBUG is a useful tool both for lab testing and for tracing bugs in live offices.

Example use of DEBUG

Assume you modified or created a module called 'OWNIOUI'. During testing of this module, you have discovered (by setting a DEBUG tracepoint, for example) that the ELSE clause of an IF statement was entered when the THEN clause should have been entered. The variable being checked is the 47th entry in a bit table pointed to by a SHARED pointer variable.

By using DEBUG, only the following command is required to display the 47th entry of the bit table pointed to by the shared variable:

```
>DI OWNIOUI:SH.5@ BT 4 (&47) HEX
40A40E:   000D
```

Restrictions and limitations

A maximum of 50 tracepoints can be defined during a single DEBUG session. Each tracepoint definition can include up to 30 tracepoint subcommands. A tracepoint definition may be displayed and extended to include up to 30 subcommands.

Informative references

The listed publications provide a foundation for understanding a broad scope of information surrounding the DEBUG utility. These publications are not referenced within the text of this document.

Note: The documents listed may exist in more than one version. See 297-1001-001 to determine the release code of the version compatible with a specific release of software.

Document	**Title**
TAM-1001-000	*Technical Assistance Manual Index of Documents*
297-1001-001	*Master Index of Practices*
297-1001-106	*Maintenance System DMS-100/200*
297-1001-107	*Maintenance and Administration Tools Description*
297-1001-110	*Maintenance and Administration Position*
297-1001-129	*Input/Output System Reference Manual*

Determining terminal and node numbers

This part describes how to calculate node and terminal numbers. The CPID command in DEBUG requires node and terminal numbers as parameters. For more information on this command, see CPID on page 3-17.

Node and terminal numbers

Message tracing requires identifying the terminals to be monitored. A terminal is an external connection to the DMS-100, such as a line, a trunk, or a data link. A terminal identifier is composed of a node number and a terminal number.

A node is any unit that can accept or originate messages. A node number is a unique number assigned by the system to a node. A terminal number is a number assigned to a specific terminal attached to a node. Terminal 0 is reserved for maintenance messaging, and the remaining terminals (1 to N) are associated with individual lines, trunks, and so on.

Terminal 0

Each node has a terminal 0 that sends and receives maintenance messages specific to the peripheral processor.

When a terminal requires a maintenance activity be performed, such as Return to Service, a message indicating that maintenance is needed is sent from terminal 0 to the CC.

When a peripheral module controller receives maintenance action commands from the MAP, such as load, busy, or test, those messages are sent to terminal 0. Therefore, if a problem is related to activities within a peripheral, monitoring the messages going to and coming from terminal 0 aids in troubleshooting.

Calculations

The CPID command in DEBUG displays a node and terminal number formatted as a Line Equipment Number (LEN) or trunk ID. Before issuing the CPID command, you must first determine the external node and terminal numbers.

To determine the PM node number and terminal number composing a terminal identifier, use one of the following methods.

- Method 1. Use the CONVERT command in PMIST to determine the terminal identifier, as follows:

  CONVERT <format_id> <value>

  For Example:

  CONVERT DN dn
  CONVERT LEN len
  CONVERT TRK clli external_trk_name

  Refer to Figure 2-1 on page 2-2 for an example of the CONVERT command.

Figure 2-1xxx
CONVERT command example

```
PMIST MULTI USER:
>convert dn 6211234

NN= 0023   TN= 013C

PMIST MULTI USER:
>
```

  In the previous example, the external node number in hexadecimal is 23 and the external terminal number in hexadecimal is 13C.

- Method 2. Use the CI command QDN with the directory number of the line to be traced as follows:

  QDN <directory number>

  Refer to Figure 2-2 on page 2-3 for an example of using the QDN command.

Figure 2-2xxx
QDN command example

```
 CI:
 >qdn 6213010
-----------------------------------------------------------
 DN:      6213010
 TYPE: SINGLE PARTY LINE
 SNPA: 613
 LINE EQUIPMENT NUMBER:    REM1 00 0 00 23
 LINE CLASS CODE:   1FR
 SIGNALLING TYPE:  DIGITONE
 LINE TREATMENT GROUP:     0
 LINE ATTRIBUTE INDEX:        0
 CARDCODE:   2X17AB   GND: N  PADGRPL STDLN  BNV: NL MNO: N
 PM NODE NUMBER        :    21
 PM TERMINAL NUMBER    :    27
 OPTIONS:
 DGT
-----------------------------------------------------------
```

In the preceding example, the node number is 21 and the terminal number is 27.

- Method 3. Use the CI command QLEN with the LEN as follows:

  QLEN <len>

  Refer to Figure 2-3 on page 2-4 for an example of the QLEN command.

Figure 2-3xxx
QLEN command example

```
CI:
>qlen 1 0 11 1
-----------------------------------------------------------
LEN:     HOST 01 0 11 01
TYPE: SINGLE PARTY LINE
SNPA: 613
DIRECTORY NUMBER:   6215111
LINE CLASS CODE:  1FR
SIGNALLING TYPE:  DIGITONE
LINE ATTRIBUTE INDEX:         32
CARDCODE  6X17    GND N PADGRP STDLN BNV NL MNO N
OPTIONS:
DGT
PM NODE NUMBER     :  63
PM TERMINAL NUMBER :  354
-----------------------------------------------------------
```

In the preceding example, the node number is 63 and the terminal number is 354.

- Method 4. Use QUERYPM at the PM access level to find the node number as follows:

  POST <node type> <device number>
  QUERYPM

  Refer to Figure 2-4 on page 2-4 for an example of the QUERYPM command.

Figure 2-4xxx
QUERYPM command example

```
PM:
>post TM8 1
POST:
>querypm
  PM TYPE TM8 PM NO.: 1 NODE NO.: 16
PM_STATUS: InSv NOTE_STATUS: OK,FALSE,CHKSUM: 018 1
PP LOAD: VALID PP EXECS: VALID FNAME: BTMIA01
PMS EQUIPPED: 30 PM INT.   :1
 Site Flr RPos  Bay_Id   Shf Description  Slot  EqPEC
 HOST  00 B01   TME 0001 04 LTC  :  000        6X02AA
```

In the preceding example, the node number is 16.

- Method 5. Use the NODENO command in PMIST to find the node number as follows:

NODENO <node type> <device class> <device number>

Refer to Figure 2-5 on page 2-5 for an example of the NODENO command.

Figure 2-5xxx
NODENO subcommand example

```
PMIST MULTI USER:
>nodeno tm_node tm8 1
NODENO=16
PMIST MULTI USER:
>
```

Calculate the terminal number for DTCs and LTCs as follows:

terminal number = (carrier * 32) + <channel> + 1

Note: You can find the carrier number, channel number, circuit number, and similar information by posting the peripheral.

For TM, MTM, TM2, TM4, and similar PM, calculate the terminal number as follows:

1 + circuit number

For LM and LCM, calculate the terminal number as follows:

(drawer * 32) + <line card> + 1

- Method 6. Terminal Identifiers for DCM

  To determine the terminal number of a DCM, post the DCM at the TTP level of the MAP. Locate the carrier number and timeslot associated with the DCM.

  Refer to Figure 2-6 on page 2-6. In the CCT column, locate the carrier number and timeslot associated with the posted DCM. The terminal number associated with the carrier and timeslot number is given in the TN column.

Figure 2-6xxx
DCM carrier and timeslot to terminal number cross-reference

CCT	TN	CCT	TN	CCT	TN	CCT	TN	CCT	TN
0-01	01	1-01	31	2-01	61	3-01	91	4-01	02
0-02	32	1-02	62	2-02	92	3-02	03	4-02	33
0-03	63	1-03	93	2-03	04	3-03	34	4-03	64
0-04	94	1-04	05	2-04	35	3-04	65	4-04	95
0-05	06	1-05	36	2-05	66	3-05	96	4-05	07
0-06	37	1-06	67	2-06	97	3-06	08	4-06	38
0-07	68	1-07	98	2-07	09	3-07	39	4-07	69
0-08	99	1-08	10	2-08	40	3-08	70	4-08	100
0-09	11	1-09	41	2-09	71	3-09	101	4-09	12
0-10	42	1-10	72	2-10	102	3-10	13	4-10	43
0-11	73	1-11	103	2-11	14	3-11	44	4-11	74
0-12	104	1-12	15	2-12	45	3-12	75	4-12	105
0-13	16	1-13	46	2-13	76	3-13	106	4-13	17
0-14	47	1-14	77	2-14	107	3-14	18	4-14	48
0-15	78	1-15	108	2-15	19	3-15	49	4-15	79
0-16	109	1-16	20	2-16	50	3-16	80	4-16	110
0-17	21	1-17	51	2-17	81	3-17	111	4-17	24
0-18	52	1-18	82	2-18	112	3-18	23	4-18	53
0-19	83	1-19	113	2-19	24	3-19	54	4-19	84
0-20	114	1-20	25	2-20	55	3-20	85	4-20	115
0-21	26	1-21	56	2-21	86	3-21	116	4-21	27
0-22	57	1-22	87	2-22	117	3-22	28	4-22	58
0-23	88	1-23	118	2-23	29	3-23	59	4-23	89
0-24	119	1-24	30	2-24	60	3-24	90	4-24	120

For example, if DCM 1 is posted at the TTP level of the MAP, and the following is displayed:

DCM 1 0 09

the carrier number is 0 and the timeslot is 09. According to the chart in Figure 2-6 on page 2-6, 0-09 is associated with terminal number 11.

To determine the node number of the DCM, post the DCM at the PM level of the MAP and issue the command QUERYPM. The node number is contained in the output of the QUERYPM command.

DEBUG commands and subcommands

This part describes the DEBUG commands, tracepoint subcommands, and system responses.

All addresses, offsets, field widths, and base register numbers ae interpreted as hexadecimal numbers. However, they may optionally be preceded by an octothorpe (#). Constants and repetition values are interpreted as decimal numbers unless they are preceded by an octothorpe (#). Strides are also interpreted as decimal numbers unless preceded by an octothorpe (#).

Accessing DEBUG

The DEBUG utility commands ae available to the user following successful access to the utility.

Type `DEBUG` from any level of the MAP to enter the DEBUG utility. Following are system responses that may occur as the user attempts to access the DEBUG utility:

```
debug mode
```

Explanation: The user issued the DEBUG command.

System Action: The DEBUG mode is entered.

User Action: No user action is required.

```
type DEBUG to enter debug mode
```

Explanation: The user issued the DEBUG command, specifying a parameter, or the word DEBUG was preceded by another command (for example, HELP DEBUG).

System Action: The command is not executed.

User Action: Reenter the DEBUG command without specifying any parameters.

CI level commands

CI level commands are accepted by the DEBUG subsystem if the commands are preceded by a dollar sign ($).

The example in Figure 3-1 on page 3-2 shows the $ character used to execute the DSKUT and LISTVOL commands from within DEBUG.

Figure 3-1xxx
Issuing CI commands from within DEBUG

```
>DEBUG
>$DSKUT
DSKUT:

>$LISTVOL D010TEMP
RECORDFILE
BUZZINIT
RECORDFILE
```

DEBUG execs ($READ command)

DEBUG command sequences are often lengthy and may be the same for many work sessions. DEBUG commands can be stored in a file in order to retrieve and execute them again.

If the file specified in the READ command contains DEBUG commands, the file is a DEBUG exec. Any commands valid at the CI level of DEBUG may be included in the file. This exec file has the effect of operating exactly as though the user was typing the commands at the terminal: prompts are output, reply messages are echoed, and error messages and actions take place. However, the input is read from the file. See Figure 3-2 on page 3-3 for an example of a DEBUG exec.

Note: There is a slight difference in using the execs when defining tracepoints. Since empty lines are difficult to work with in files, identifying the end of a subcommand list is done by inserting a line with only blanks (spaces).

Figure 3-2xxx
DEBUG exec

```
        File EX1$EXEC contains

          DEF T1 LNUTILUI CLOSE_LINE_CHANNEL #12
          TIMESTAMP
          DI   OAUUI1:PR.42   (&0)    (&0) TRMNL_ID
          DI   OAUUI1:PR.42   (&0)    (&0) .1,8,8 CARD_CODE
             <line with only a space>
          ACT T1

    >$READ EX1$EXEC
    enter subcommands
    end of subcommand list of 3 items
    tracepoint T1 defined
    tracepoint T1 activated
    >
```

Adding comments

Comments can be entered on separate lines or added to any command line. Precede a comment line with a % character. Alternatively, follow a command by a % character and some explanation. This latter step is often done in the definition of tracepoint commands, to indicate what data is being displayed.

The example in Figure 4.3 shows the % character used to add a comment to a tracepoint subcommand.

Figure 3-3xxx
Adding comments

```
    >DEFINE E1 LLERPRCI LL_ERROR 0
    enter subcommands
    >DI R 4.3C.0.1F.0,1,8  %display NCOS
    >
    end of subcommand list of 1 items
    tracepoint E1 defined
```

DEBUG command description

ACTIVATE

The ACTivate command activates a previously defined tracepoint so it can begin collecting data. All tracepoints are deactivated following a restart, except when the RESTART parameter is used in the ACTivate command; in that case, the tracepoint is scheduled to be activated after the next restart. ACTivate is also a tracepoint subcommand. For more information, refer to ACTIVATE on page 3-67.

ACTivate	**tracepoint_name** [**RESTART**] **ALL**

Where:

tracepoint_name	is the name of an existing tracepoint.
ALL	specifies that all defined tracepoints are to be activated.
RESTART	specifies that the tracepoint is not to be activated until after a restart.

Responses

* token error * invalid character at column ____

Explanation: The user issued the ACTivate command, and the tokenizer encountered a character it did not recognize.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid tracepoint name or ALL.

expecting tracepoint name

Explanation: The user issued the ACTivate command with no parameters.

System Action: The command is not executed.

User Action: Reenter the command, specifying either a tracepoint name or ALL.

Someone is changing the buffer size.
Wait and try the command again.

Explanation: Another user is currently changing the size of the tracepoint data buffers, so tracepoints cannot be activated at this time.

System Action: The command is not executed.

User Action: Reenter the command.

tp ____ not activated, clear and redefine it

Explanation: The tracepoint cannot be activated because there is no store for bind table values. If the tracepoint handler attempts to use the store, a restart may result.

System Action: The command is not executed.

User Action: Clear the tracepoint, redefine it, and then activate it.

TP ____ not activated. Clear and redefine it.

Explanation: The tracepoint cannot be activated because there is no store for bind table values. If the tracepoint handler attempts to use the store, it may result in a restart.

System Action: The command is not executed.

User Action: Clear the tracepoint, redefine it, and then activate it.

tracepoint ____ activated

Explanation: The user issued the ACTivate command with the tracepoint parameter.

System Action: The command is executed.

User Action: No user action is required.

Tracepoint ____ not activated: ____

Explanation: The user issued the ACTivate ALL command, and there was an invalid opcode at that location.

System Action: The command is not executed.

User Action: Clear the tracepoint and redefine it.

tracepoint ____ already active.

Explanation: The user attempted to activate a tracepoint that was already active.

System Action: The command is not executed.

User Action: No user action is required.

Tracepoint ____ was already active.

Explanation: The user entered the ACTivate ALL command, and one or more tracepoints were already active.

System Action: The tracepoints that were not already activated are activated.

User Action: No user action is required.

tracepoint ____ will be activated during next restart

Explanation: The user entered the ACTivate command with the RESTART parameter.

System Action: The command is executed.

User Action: No user action is required.

unknown symbol ____ at column ____

Explanation: The user issued the ACTivate command and specified extra parameters beyond the command.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name or ALL.

unknown tracepoint

Explanation: The user attempted to activate a tracepoint that had not been defined.

System Action: The command is not executed.

User Action: Define the tracepoint and then activate it, or activate a tracepoint that is already defined.

unknown tracepoint name

Explanation: The user attempted to activate a tracepoint that had not been defined.

System Action: The command is not executed.

User Action: Define the tracepoint and then activate it, or activate a tracepoint that is already defined.

Usage notes

1 All tracepoints are deactivated following a restart, except when the RESTART parameter is used; in that case, the tracepoint is not activated until a restart. In order to have a tracepoint activated both before and after a restart, the user must issue the ACTIVATE command twice, first with the RESTART parameter and next without the RESTART parameter.

2 ACTivate is also a tracepoint subcommand, which means that a tracepoint can activate another tracepoint when the first tracepoint gets hit. This is useful if a tracepoint gets hit very often, and the user is interested only when another piece of code is executed first.

Examples:

1 The following command string activates tracepoint TP1:

```
>ACT TP1
tracepoint TP1 activated
```

2 The following command string activates all tracepoints defined in the system and makes them active both before and after a restart:

```
>ACT ALL RESTART
tracepoint TP1 will be activated during next restart
tracepoint TP2 will be activated during next restart
tracepoint TP3 will be activated during next restart
>ACT ALL
tracepoint TP1 activated
tracepoint TP2 activated
tracepoint TP3 activated
```

ALLOC

The ALLOC command changes the size of the tracepoint subcommand buffer and the tracepoint data buffer accordingly. The tracepoint data buffer is always 10 words/subcommand, or 10 times the tracepoint subcommand buffer. The buffer size can only be changed when no tracepoints are active and no tracepoint data has been saved. To ensure these conditions, type the following command strings:

```
>DEACT ALL
>RESET ALL
```

If no parameter is specified with the ALLOC command, the current tracepoint buffer sizes are displayed. The default for the tracepoint subcommand buffer size is 220 subcommands, so the data buffer size default is 2200 words.

ALLOC	[**number_of_subcommands**]

Where:

number_of_subcommands
specifies the size you desire to change the subcommand buffer to (1-3276).

Responses

* token error * invalid character at column ____

Explanation: The user issued the ALLOC command, and the tokenizer encountered a character it did not recognize.

System Action: The command is not executed.

User Action: Reenter the command, specifying the number of subcommands or no parameters.

A positive number must be specified.

Explanation: The requested size of the subcommand buffer was less than one.

System Action: The command is not executed.

User Action: Reenter the command, specifying positive number for the number of subcommands.

Data buffer size: ____ words.
Subcommand buffer size: ____ subcommands.

Explanation: The user issued the ALLOC command with no parameters.

System Action: The command is executed.

User Action: No user action is required.

```
Failed to allocate new buffers.
The old buffers remain unchanged.
```

Explanation: DEBUG requested data store from the system, and the request was refused because data store on the switch was exhausted.

System Action: The command is not executed.

User Action: Wait and try the command again. (Hopefully, someone will release some data store, and the command will be executed.)

```
New data buffer size:  ____ words.
New subcommand buffer size:  ____ subcommands.
```

Explanation: The ALLOC command was issued with the number_of_subcommands parameter.

System Action: No system action is required.

User Action: No user action is required.

```
The buffers cannot be changed with tracepoints active.
```

Explanation: The user attempted to change the buffer sizes while tracepoints were still active.

System Action: The command is not executed.

User Action: Type the following command string:

```
>DEACT ALL
>RESET ALL
```

Then reissue the ALLOC command

```
The requested size is too large.  Max is ____.
```

Explanation: The user attempted to change the subcommand buffer size to more than the maximum allowable number of subcommands.

System Action: The command is not executed.

User Action: Reenter the command specifying a valid subcommand buffer size.

There is data present in the buffer. You cannot change
the buffer size until the buffer is empty.
To ensure this issue the command RESET ALL.

Explanation: The user attempted to change the buffer sizes while the buffers still contained saved data.

System Action: The command is not executed.

User Action: Issue the RESET ALL command, and then reissue the ALLOC command.

unknown symbol ____ at column ____

Explanation: The user issued the ALLOC command and specified extra parameters beyond the command.

System Action: The command is not executed.

User Action: Reenter the command, specifying the number of subcommands or no parameters.

Examples:

1 The following command string displays the current tracepoint buffer sizes:

```
>ALLOC
Data buffer size:  2200 words
Subcommand buffer size:  220 subcommands
```

2 The following command string sets the tracepoint buffer size to 500 subcommands:

```
>ALLOC 500
Data buffer size:  5000 words
Subcommand buffer size:  500 subcommands
```

BIND

The BIND command assigns a symbolic name (bind name) to the specified storage reference. Illegal bind names include names which can be interpreted as hex numbers or storage reference elements (for example, R, SH, or L). BIND is also a tracepoint subcommand. For more information, refer to BIND on page 3-69.

BIND	**bind_name [TO] storage_reference**

Where:

bind_name	is a string of up to eight characters, excluding certain values reserved for DEBUG, such as hex numbers and storage reference elements (1, 2, 3, ... D, E, E, F, L, PP, PR, R, S, SB, SH).
storage_reference	is the storage reference which the bind name will now represent. Acceptable types include the following: • absolute address • base register • S (expression stack top) • SB or L (base of current stack frame) • bind name • module name followed by store type. Store type can be any of the following: - PR.offset (for protected global variables) - PP .offset (for private global variables) - SH.offset (for shared global variables).

Responses

* token error * invalid character at column ____

Explanation: The user issued the BIND command, and the tokenizer encountered a character it did not recognize.

System Action: The command is not executed.

User Action: Reenter the command, specifying a bind name and storage reference.

attempt to add existing name

Explanation: The user issued the BIND command with a bind name that already exists.

System Action: The command is not executed.

User Action: Reenter the command, specifying an unused bind name.

directory error

Explanation: An internal directory error has occurred. If this error occurs, there are problems with the switch.

System Action: The command is not executed.

User Action: Try the command again.

directory illegal

Explanation: An internal directory error has occurred. If this error occurs, there are problems with the switch.

System Action: The command is not executed.

User Action: Try the command again.

directory is full

Explanation: The user issued the BIND command, and the maximum number of BINDs has been specified.

System Action: The command is not executed.

User Action: Delete some BINDs using the DELBIND command. Then reissue the BIND command.

directory string error

Explanation: An internal directory error has occurred.

System Action: The command is not executed.

User Action: Try the command again.

expecting bind_id

Explanation: The user issued the BIND command without specifying any parameters, or the user entered a hexadecimal number for a bind name.

System Action: The command is not executed.

User Action: Reenter the command, specifying a bind name and storage reference.

expecting MOD: ,PP,PR,SH or L,S,B,R,S,& number, or addr

Explanation: The user issued the BIND command without specifying the storage reference.

System Action: The command is not executed.

User Action: Reenter the command, specifying a bind name and storage reference.

expecting storage reference

Explanation: The user issued the BIND command without specifying the storage reference.

System Action: The command is not executed.

User Action: Reenter the command, specifying a bind name and storage reference.

L,S,R,SB,SH,PR,PP are illegal bind names

Explanation: The user attempted to BIND using an illegal bind name or entered BIND and a storage reference without entering a bind name.

System Action: The command is not executed.

User Action: Reenter the command, specifying a legal bind name.

unknown symbol ____ at column ____

Explanation: The user issued the BIND command and specified extra parameters beyond the command.

System Action: The command is not executed.

User Action: Reenter the command, specifying a bind name and storage reference.

Example:

The following command string binds storage reference OAUUI1:PR.60 to bindname JTG:

```
>BIND JTG TO OAUUI1:PR.60
```

This results in an alternative reference for OAUUI1:PR.60. Compare the following two equivalent DIsplay commands:

```
>DI OAUUI1:PR.60 BT 4 (&0)
08354D:  0008

>DI JTG BT 4 (&0)
08354D:  0008
```

CLEAR

The CLEAR command deletes one or all tracepoints that have been defined in the system, regardless of whether they are activated or deactivated. All tracepoint definitions and data collected for the tracepoints are lost.

CLEAR	**tracepoint_name** **ALL**

Where:

tracepoint_name	is the name of an existing tracepoint.
ALL	specifies that all tracepoints that are defined are to be cleared.

Responses

*token error * invalid character at column ____

Explanation: The user issued the CLEAR command, and the tokenizer encountered a character it did not recognize.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name or ALL.

expecting tracepoint name or ALL

Explanation: The user issued the CLEAR command without specifying any parameters.

System Action: The command is not executed.

User Action: Reenter the command, specifying a defined tracepoint name or ALL.

tracepoint ____ cleared

Explanation: The user entered the CLEAR command.

System Action: The command is executed.

User Action: No user action is required.

unknown symbol ____ at column ____

Explanation: The user issued the CLEAR command and specified extra parameters beyond the command.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name or ALL.

unknown tracepoint name

Explanation: The user attempted to clear a tracepoint that has not been defined.

System Action: The command is not executed.

User Action: Reenter the command, specifying either a defined tracepoint name or ALL.

Warning - tracepoint is referenced by another tracepoint

Explanation: The tracepoint the user wishes to clear has been referenced within another tracepoint subcommand.

System Action: The command is executed.

User Action: Remove all references to that tracepoint within other tracepoints.

Usage notes

DEBUG is a multi-user program, so be sure you do not clear another user's tracepoints by specifying the ALL parameter.

Examples:

1 The following command string clears tracepoint TP1:

```
>CLEAR TP1
tracepoint TP1 cleared
```

2 The following command string clears all tracepoints from the system:

```
>CLEAR ALL
tracepoint TP1 cleared
tracepoint TP2 cleared
tracepoint TP3 cleared
```

CPID

The CPID command displays the CP_ID bound as a node and terminal number formatted in the manner appropriate to the CP selector. See Chapter 2 on page 2-1 for an explanation of how to calculate node numbers and terminal numbers.

CPID	**node_number terminal_number**

Where:

node_number is the node number (range is 0 to 4095).

terminal_number is the terminal number (range is 0 to 4095).

Responses

* token error * invalid character at column ____

Explanation: The user issued the CPID command, and the tokenizer encountered a character it did not recognize.

System Action: The command is not executed.

User Action: Reenter the command, specifying a node and terminal number.

expecting node number

Explanation: The user issued the CPID command without specifying any parameters or specified an alphabetic or alphanumeric value for the node number.

System Action: The command is not executed.

User Action: Reenter the command, specifying a node number and terminal number.

expecting terminal number

Explanation: The user issued the CPID command, specifying only one parameter or specified an alphabetic or alphanumeric value for the terminal number.

System Action: The command is not executed.

User Action: Reenter the command, specifying a node number and terminal number.

The node number is out of range.

Explanation: The node number the user entered is not a valid node number.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid node number.

The terminal number out of range.

Explanation: The terminal number the user entered is not a valid terminal number.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid terminal number.

unknown symbol ____ at column ____

Explanation: The user issued the CPID command and specified extra parameters beyond the command.

System Action: The command is not executed.

User Action: Reenter the command, specifying a node and terminal number.

Usage notes

The CPID command is accepted only in offices where call processing software is loaded.

Example:

The following command string displays node #10, terminal #102 as a LEN:

```
>CPID #19 #12
LEN HOST 1 0 12 0
```

DEACTIVATE

The DEACTivate command deactivates tracepoints that have been previously defined and activated. A tracepoint is deactivated after a restart (if RESTART parameter was not specified with ACTIVATE) and once its limit is reached. For more information on tracepoint limits, refer to LIMIT on page 3-52. DEACTivate is also a tracepoint subcommand, so a tracepoint can deactivate itself or another tracepoint. For more information, refer to DEACTIVATE on page 3-71.

DEACTivate	**tracepoint_name** [**RESTART**] **ALL**

Where:

tracepoint_name	is the name of an existing tracepoint.
ALL	specifies that all tracepoints that are defined are to be deactivated.
RESTART	schedules a tracepoint to be deactivated after the next restart.

Responses

* token error * invalid character at column ____

Explanation: The user issued the DEACTivate command, and the tokenizer encountered a character it did not recognize.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name or ALL.

expecting tracepoint name

Explanation: The user issued the DEACTivate command with no parameters.

System Action: The command is not executed.

User Action: Reenter the command, specifying either a tracepoint name or ALL.

restart activation deactivated

Explanation: The user issued the DEACTivate command with RESTART as a command parameter.

System Action: The command is executed.

User Action: No user action is required.

restart activation deactivated for ____

Explanation: The user issued the DEACTivate command with ALL RESTART as command parameters.

System Action: The command is executed.

User Action: No user action is required.

tracepoint ____ deactivated

Explanation: The user issued the DEACTivate command with the tracepoint name parameter.

System Action: The command is executed.

User Action: No user action is required.

tracepoint ____ is inactive

Explanation: The user attempted to deactivate a tracepoint that was not currently active.

System Action: The command is not executed.

User Action: No user action is required.

Tracepoint ____ was not active

Explanation: The user specified the ALL parameter, and the tracepoint(s) listed in the error message are not active.

System Action: The command is not executed for any inactive tracepoints.

User Action: No user action is required.

Tracepoint ____ was not deactivated: ____

Explanation: The user issued the DEACTivate ALL command, and an internal error occurred.

System Action: The command is not executed.

User Action: This message signifies that an internal error has occurred. This message should never occur.

unknown symbol ____ at column ____

Explanation: The user issued the DEACTivate command and specified extra parameters beyond the command.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name or ALL.

UNKNOWN TRACEPOINT NAME

Explanation: The user attempted to deactivate a tracepoint that had not been defined.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint that is already active.

Examples:

1 The following command string deactivates tracepoint TP1:

```
>DEACT TP1
tracepoint TP1 deactivated
```

2 The following command string deactivates all tracepoints defined in the system and disables the restart activation:

```
>DEACT ALL RESTART
restart activation deactivated for TP1
restart activation deactivated for TP2
restart activation deactivated for TP3
```

DEFINE

The DEFine command defines tracepoints in program store. After the tracepoint is defined, DEBUG will prompt for subcommands. The subcommand input is terminated with a blank line and carriage return.

Note: Refer to Tracepoint Procedure Code Offset on page 5-10 for differences in tracepoint definition in the NT40 and SuperNode.

DEFine	**tracepoint_name address**

Where:

tracepoint_name	is the name of the tracepoint (up to 32 characters).
address	is the program store reference. Acceptable types include the following: • absolute byte address (up to six hex digits) • module_name procedure_name offset (offset defaults to beginning of procedure).

Responses

* token error * invalid character at column ____

Explanation: The user issued the DEFine command, and the tokenizer encountered a character it did not recognize.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name and address.

*** ERROR *** Unstackfrn failed.

Explanation: A Command Interpreter (CI) error occurred when DEBUG was reading the DEFine command from a DEBUG exec file ($READ command).

System Action: The command is not executed.

User Action: Reenter the command.

A tracepoint is already defined at this address
tracepoint definition failed

Explanation: A tracepoint has already been defined at the address specified in the command.

System Action: The command is not executed.

User Action: No user action is required.

address does not lie within a procedure

Explanation: The absolute byte address entered does not lie within a procedure.

System Action: The command is not executed.

User Action: Reenter the command, specifying an address within a procedure.

ALL is an invalid tracepoint name.

Explanation: The user issued the DEFine command with ALL as a parameter.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid tracepoint name and address.

attempt to redefine existing tracepoint

Explanation: The user issued the DEFine command specifying a tracepoint name that had already been defined.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name that has not yet been defined.

DIRECTORY ERROR, cannot add tracepoint

Explanation: An internal error occurred when the user issued the DEFine command.

System Action: The command is not executed.

User Action: Reenter the command.

end of subcommand list of ____ items

Explanation: The user issued the DEFine command, and the last subcommand for the tracepoint has been issued, denoted by entering a blank line.

System Action: No system action occurs; subcommand input is terminated.

User Action: No user action is required.

enter subcommands

Explanation: The tracepoint has been defined, and the user can now enter subcommands.

System Action: No system action is required.

User Action: Enter any tracepoint subcommands, ending the list by entering a null line.

enter tracepoint name or number

Explanation: The user issued the DEFine command without specifying any parameters.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid tracepoint name and address.

expecting program store address

Explanation: The user issued the DEFine command without specifying an address.

System Action: The command is not executed.

User Action: Reenter the command, specifying an address (program store reference).

expecting tracepoint name or number

Explanation: The user issued the DEFine command without specifying any parameters.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid tracepoint name and address.

failed to allocate directory for local binds

Explanation: DEBUG requested data store from the system, and the request was refused because data store on the switch was exhausted.

System Action: The command is not executed.

User Action: Wait and try the command again. (Hopefully, someone will release some data store, and the command will be executed.)

failed to allocate store for symbol table
the command will be executed.

Explanation: DEBUG requested data store from the system, and the request was refused because data store on the switch was exhausted.

System Action: The command is not executed.

User Action: Wait and try the command again. (Hopefully, someone will release some data store, and the command will be executed.)

failed to allocate store for subcommands

Explanation: DEBUG requested data store from the system, and the request was refused because data store on the switch was exhausted.

System Action: The command is not executed.

User Action: Wait and try the command again. (Hopefully, someone will release some data store, and the command will be executed.)

illegal hex number

Explanation: The address specified is a valid hex address.

System Action: The command is not executed.

User Action: Reenter the command specifying a valid hex address.

invalid module name

Explanation: The module name specified is not an actual program store reference.

System Action: The command is not executed.

User Action: Reenter the command specifying a valid module name.

invalid opcode at offset ____

Explanation: The user issued the DEFine command, and the specified address is not aligned on an operation code boundary. This is probably the result of a patch in the code.

System Action: The command is not executed.

User Action: If you are certain about the location, issue the FDEFine command. Otherwise, reissue the DEFine command, specifying a valid address.

invalid program store address

Explanation: The address specified in the command is not an actual program store reference.

System Action: The command is not executed.

User Action: Reenter the command specifying a valid address.

invalid subcode entered ____

Explanation: DEBUG disassembled an invalid SYSOP operation code (NT40 only).

System Action: The command is not executed.

User Action: This error should not occur. If it does occur, and you are certain about the location, issue the FDEFine command.

invalid tracepoint subcommand

Explanation: The subcommand entered was a valid tracepoint command but not a valid tracepoint subcommand.

System Action: The command is not executed.

User Action: Enter a subcommand included in Tracepoint Subcommands on page 3-67.

max number of tracepoints already defined

Explanation: 50 tracepoints have already been defined.

System Action: The command is not executed.

User Action: Clear existing tracepoints before attempting to add new ones.

tracepoint ____ defined

Explanation: The user entered a null line to signify the end of the subcommand list.

System Action: The command is executed.

User Action: No user action is required.

tracepoint not aligned on machine instruction
last opcode is at ____, tracepoint is at ____

Explanation: The user specified an address in the middle of an instruction.

System Action: The command is not executed.

User Action: Reenter the command, specifying an address at the beginning of an instruction or, if you are certain about the address you entered, enter the FDEFine command.

unable to allocate store for tracepoint subcommands

Explanation: DEBUG requested data store from the system, and the request was refused because data store on the switch was exhausted.

System Action: The command is not executed.

User Action: Wait and try the command again. (Hopefully, someone will release some data store, and the command will be executed.)

unknown debug command

Explanation: The subcommand entered was not a valid tracepoint subcommand.

System Action: The subcommand is not executed.

User Action: Enter a valid tracepoint subcommand from the list shown in Tracepoint Subcommands on page 3-67.

unknown procedure name

Explanation: The procedure name specified in the address is not a valid procedure name.

System Action: The subcommand is not executed.

User Action: Reenter the command, specifying a valid address.

Unknown SWBP encountered at ____

Explanation: In the process of operation code alignment, a tracepoint was encountered earlier in the procedure.

System Action: The command is not executed.

User Action: For multiple tracepoints in one procedure, place the tracepoints in reverse order. That is, define the one farthest from the beginning of the procedure first, the one second farthest from the beginning second, etc.

unknown symbol ____ at column ____

Explanation: The user issued the DEFine command and specified extra parameters beyond the command.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name and address.

Warning: non-DEBUG SWBP found at ____

Explanation: A non-DEBUG software breakpoint was encountered earlier in the procedure.

System Action: The command is not executed.

User Action: No user action is required.

Usage notes

A maximum of 50 tracepoints can be defined.

Examples:

1 The following command string defines a tracepoint at offset 3 in module LSETPRCI, procedure LINE_SETUP_PROCESSOR:

```
>DEF TP1 LSETPRCI LINE_SETUP_PROCESSOR #3
enter subcommands
```

2 The following command string defines a tracepoint at address 17459:

```
>DEF TP1 #17459
enter subcommands
```

DELBIND

The DELBIND command removes one or all BINDs defined in the system by the BIND command. It is recommended that all BINDs be removed at the end of a DEBUG session

DELBIND	**bind_name** **ALL**

Where:

bind_name
is the name of an existing bind (alphanumeric or alphabetic).

ALL specifies that all BINDs are to be cancelled.

Responses

* token error * invalid character at column ____

Explanation: The user issued the DELBIND command, and the tokenizer encountered a character it did not recognize.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid bind name or ALL.

bind cancelled: ____

Explanation: The user issued the DELBIND command with the bind name parameter.

System Action: The command is executed.

User Action: No user action is required.

command not implemented

Explanation: An internal consistency check error occurred.

System Action: The command is not executed.

User Action: This error message should never occur.

expecting bind_id

Explanation: The user issued the DELBIND command without specifying any parameters, or the user entered a hexadecimal number for a bind name.

System Action: The command is not executed.

User Action: Reenter the command, specifying a bind name or ALL.

expecting bind name or ALL

Explanation: The user issued the DELBIND command without specifying any parameters, or the user specified a numeric bind name.

System Action: The command is not executed.

User Action: Reenter the command, specifying a bind name or ALL.

unknown bind name

Explanation: The bind name entered is not a valid bind name defined using the BIND command.

System Action: The command is not executed.

User Action: No user action is required.

unknown symbol ____ at column ____

Explanation: The user issued the DELBIND command and specified extra parameters beyond the command.

System Action: The command is not executed.

User Action: Reenter the command, specifying a legal bind name or ALL.

Usage notes

At the end of a DEBUG session, a user should cancel any BINDs created during that session. However, the user must be careful not to cancel any BINDs created by other DEBUG users.

Example:

The following command string cancels bind JTG:

```
>DELBIND JTG
bind cancelled: JTG
```

DISPLAY (data store)

The DIsplay command is used to display values in data and program store. This section describes how to use the DIsplay command for displaying values in data store.

For information on displaying program store, refer to DISPLAY PS (Program Store) on page 3-35. Display is also tracepoint subcommand. For more information, refer to DISPLAY (Data Store) on page 3-74.

Note: Refer to Data Store on page 5-3 for the differences in data store on the NT40 and SuperNode.

DIsplay	**storage_reference [store_index] [size] [format]**

Where:

storage_reference

is the storage reference. Acceptable types include the following:

- absolute address (up to six hex digits), optionally preceded by # (for example, #39AE4)
- base register other than R 0 (in the format R nn; for example, R 4)
- S (expression stack top); n words below the top of the stack (for example, S.-2 or S.-#2)
- SB or L (base of current stack frame); local variable at offset nn (for example, L.nn or L.#nn or SB.nn or SB.#nn)
- bind name (as defined using the BIND command or subcommand)
- module name followed by store type.

Store type can be any of the following:

- PR.offset (for protected global variables)
- SH.offset (for shared global variables).

Offset can be up to four hex digits.

store_index can be any of the following:

- .nn (word offset on NT40 or byte offset on SuperNode)
- .nn, b, w (word offset on NT40 or byte offset on SuperNode, bit offset, bit width)

This is a reference to a bit offset within a word. If no bit width is specified, the default is the rest of the word. This accesses a bit field of width w bits, at offset nn words, b bits from the given address. Another representation for this reference is the following.

.nn OFB b W w.

- .nn@ (dereferencing a pointer at the address given)
- .nn (i) (indexing descriptors)
- .nn B (i) (indexing bit descriptors)
- .nn T x (i) (indexing a table given the stride; x defaults to 1)
- .nn BT x (i) (indexing a bit table given the stride; x defaults to 1)
- .nn TB x (i) (indexing a bit table given the stride; x defaults to 1).

Where:

- nn=word offset on NT40 or byte offset on SuperNode
- b=bit offset
- w=bit width
- x=table stride
- i=index in table.

i can be either of the following:

- &number (any numeric value)

  Note: If the "&" is omitted, the number is interpreted as an address instead of a constant.

- storage_reference (to at most one word of store; see previous explanation).

size is the number (form N=nn) of items to be displayed.

format is one of the following output formats:

- DEC (decimal output)
- HEX (hexadecimal output - this is the default output format)
- ADDR (six-digit hexadecimal address)
- DESC (address - size - stride)

- CHAR (readable character string)
- CCB (Display is formatted as Call Condense Block if the DISP_CALL_PROCS are bound in. Otherwise, it is dumped in hex. Attempting to format non-CCB data as a CCB may cause a trap.)
- CDB (Display is formatted as a Call Data Block if the DISP_CALL_PROCS are bound in. Otherwise, it is dumped in hex. Attempting to format non-CDB data as a CDB may cause a trap.)
- EXT (Display is formatted as an Extension Block if the DISP_CALL_PROCS are bound in. Otherwise, it is dumped in hex. Attempting to format non-EXT data as an EXT may cause a trap.)
- typename (for Data Dictionary format). This is mainly used to display I/O type data. All type names known by DDEDIT are allowed.

Responses

Because there are numerous system responses for the DIsplay command, they are listed here without explanations. Most of the responses are self-explanatory.

```
* token error * invalid character at column ____
cannot do word offset on bit field
desc, addr formats only apply to word items
Desc stride must be entered as D n
expecting =
expecting base register number
expecting bit offset
expecting bit width
expecting closing parenthesis
expecting decimal number
expecting hexnumber
expecting MOD:,PP,PR,SH or L,SB,R,S,& number, or addr
expecting number
expecting offset
expecting opening parenthesis
expecting PP,PR or SH
expecting storeref
failed to scan transitive uses list
illegal base register number
illegal hex number
illegal hexnumber
illegal index
```

illegal indexing
invalid base register number
invalid bit stride
invalid dec number
invalid dereference
Invalid Ecore bit descriptor syntax
invalid format or N
invalid module name
invalid perprocess reference
modifier cannot operate on one word of a register
modifier cannot operate on this register
negative address not accepted
PP only valid for perprocess module
PP reference at CI level requires module name
PR reference at CI level requires module name
protected segment does not exist
SH reference at CI level requires module name
shared segment does not exist
the first modifier must be @
too many digits
too many modifiers
typename not found
unable to allocate store
unknown bind name
unknown symbol
unknown symbol ____ at column ____

Usage notes

1 Recursive indexing or dereferencing is allowed.
2 Invalid indexing or dereferencing gives an error message.
3 Indexing or dereferencing cannot follow bit offset.
4 Further qualification of indexing or dereferencing is possible.

Example:

Refer to Chapter 4 on page 4-1 for examples of displaying data store.

DISPLAY PS (program store)

The DIsplay PS command displays program store in assembler format unless hexadecimal format is specified. The `BREAK` key can be used to stop the display. Use the `ENTER` key to continue.

Note: Refer to Program Store on page 5-1 for the differences in program store on the NT40 and SuperNode.

DIsplay PS	**start_address [size] [HEX]**

Where:

start_address	is the program store reference. Acceptable types include the following: • bind name • absolute byte address: up to six hex digits • module_name procedure_name <offset>: offset defaults to beginning of procedure.
size	is the number, in bytes (form N=nn), of items to be displayed. If not specified, the rest of the procedure is displayed.
HEX	specifies that the output is to be displayed in hexadecimal format. If not specified, the output is displayed in assembler format.

Responses

* token error * invalid character at column ____

Explanation: The user issued the DI PS command, and the tokenizer encountered a character it did not recognize.

System Action: The command is not executed.

User Action: Reenter the command, specifying valid parameter(s).

expecting procedure name

Explanation: The user issued the DIsplay PS command without specifying a start address.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid start address.

expecting program store address

Explanation: The user issued the DIsplay PS command without specifying any parameters.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid start address.

illegal hex number

Explanation: The user issued the DI PS command, specifying an address greater than 6 digits on the NT40 or 8 digits on SuperNode.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid start address.

invalid module name

Explanation: The user issued the DI PS command, specifying a module name that does not exist.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid module name.

MORE THAN 1 PROCEDURE WITH THAT NAME
CHECK SELECTED NAME
USE FULL NAME IF NECESSARY.

Explanation: The user issued the DI PS command, specifying a procedure name that is the procedure name for more than one procedure.

System Action: The command is not executed.

User Action: Reenter the command, specifying the full procedure name.

Nil procedure descriptor

Explanation: The user issued the DI PS command specifying an entry procedure name. After Initial Program Load (IPL), entry procedures can no longer be displayed.

System Action: The command is not executed.

User Action: No user action is required.

offset exceeds procedure size

Explanation: The user issued the DI PS command, specifying an offset that goes beyond the end of the procedure.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid start address.

offsets are from entered ps address

Explanation: The user issued the DI PS command, specifying an address. Offsets from the beginning of procedure cannot be displayed if the program store does not belong to a procedure. Therefore, the offsets are from the address specified.

System Action: The command is executed.

User Action: No user action is required.

unknown bind name

Explanation: The user either entered an invalid start address or none at all.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid start address.

unknown procedure name

Explanation: The user issued the DI PS command, specifying a procedure name that does not exist in the specified module.

System Action: The command is not executed.

User Action: Check the procedure name, and then reenter the command.

unknown symbol

Explanation: The user issued the DI PS command and specified extra parameters beyond the command.

System Action: The command is not executed.

User Action: Reenter the command, specifying the correct parameters.

unknown symbol ____ at column ____

Explanation: The user issued the DI PS command and specified extra parameters beyond the command.

System Action: The command is not executed.

User Action: Reenter the command, specifying the correct parameters.

Examples:

1 The following command string displays the rest of the procedure from address 17459:

```
>DI PS #17459
          017459:     0007   PUSH2        #0001 (B03)
          01745D:     000B   PUSH         #0004 (B03)
          017461:     000F   PUSHS        #0 (SB)
          017463:     0011   CALLDC       #09 (BFB)
          017466:     0014   COPY
          017467:     0015   POP          #0004 (B03)
          01746B:     0019   NE
          01746C:     001A   FEND
```

2 The following command displays 10 bytes of program store from procedure CLOSE_LINE_CHANNEL starting at offset 12 (CLOSE_LINE_CHANNEL is in module LNUTILUI):

```
>DI PS LNUTILUI CLOSE_LINE_CHANNEL #12 N=10
          3A16E0:     0012   POP          #101B (SB)
          3A16E4:     0016   JUMPS        #19
          3A16E6:     0018   PUSHSV       #06
          3A16E8:     001A   PUSHV        #02BD
```

EXTEND

The EXTend command allows the user to add subcommands to an existing tracepoint. For a list of acceptable tracepoint subcommands, refer to Tracepoint Subcommands on page 3-67. Subcommand input is terminated by a blank line followed by a carriage return.

EXTend	**tracepoint_name**

Where:

tracepoint_name is the name of an existing tracepoint.

Responses

* token error * invalid character at column ____

Explanation: The user issued the EXTend command, and the tokenizer encountered a character it did not recognize.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name.

*** ERROR *** Unstackfrn failed.

Explanation: A Command Interpreter (CI) error occurred when DEBUG was reading the EXTend command from a DEBUG exec file ($READ command).

System Action: The command is not executed.

User Action: Reenter the command.

cannot extend tracepoint which must be redefined

Explanation: The user issued the EXTEND command, and a restart occurred recently.

System Action: The command is not executed.

User Action: Clear the tracepoint and redefine it.

cannot specify ALL for EXTEND

Explanation: ALL was specified as a parameter to the EXTend command.

System Action: The command is not executed.

User Action: Reenter the command specifying a tracepoint name to be extended.

end of subcommand list of ____ items
tracepoint ____ extended

Explanation: The user issued the EXTend command, and the last subcommand of the tracepoint has been issued, denoted by entering a blank line.

System Action: No system action occurs; subcommand input is terminated.

User Action: No user action is required.

enter subcommands

Explanation: The user issued the EXTend command and can now enter subcommands.

System Action: No system action is required.

User Action: Enter any tracepoint subcommands, ending the list by entering a null line.

expecting tracepoint name

Explanation: The user issued the EXTend command without specifying any parameters.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid tracepoint name.

failed to allocate store for subcommands

Explanation: DEBUG requested data store from the system, and the request was refused because data store on the switch was exhausted.

System Action: The command is not executed.

User Action: Wait and try the command again. (Hopefully, someone will release some data store, and the command will be executed.)

invalid tracepoint subcommand

Explanation: The subcommand entered was a valid tracepoint command but not a valid tracepoint subcommand.

System Action: The command is not executed.

User Action: Enter a subcommand included in Tracepoint Subcommands on page 3-67.

unable to allocate store to extend subcommand list

Explanation: DEBUG requested data store from the system, and the request was refused because data store on the switch was exhausted.

System Action: The command is not executed.

User Action: Wait and try the command again. (Hopefully, someone will release some data store, and the command will be executed.)

unknown debug command

Explanation: The user entered an invalid tracepoint subcommand.

System Action: The command is not executed.

User Action: Enter a valid tracepoint subcommand.

unknown symbol ____ at column ____

Explanation: The user issued the EXTend command and specified extra parameters beyond the command.

System Action: The command is not executed.

User Action: Reenter the command, specifying tracepoint name.

unknown tracepoint name

Explanation: The user attempted to extend a tracepoint that had not been defined.

System Action: The command is not executed.

User Action: Define the tracepoint and then extend it.

Example:

The following command string adds tracepoint subcommand DI R 1 to tracepoint TP1:

```
>EXT TP1
enter subcommands
>DI R 1
>
end of subcommand list of 1 items
tracepoint TP1 extended
```

FDEFINE

The FDEFine command defines tracepoints in program store. Unlike the DEFine command, FDEFine will place a tracepoint in the middle of an instruction. Activation will still fail if the tracepoint is defined on an invalid operation code. This capability is mainly for the placement of tracepoints in patched code.

CAUTION

Careless placement of tracepoints can cause warm restarts.

Note: Refer to Tracepoint Procedure Code Offset on page 5-10 for differences in tracepoint definition in the NT40 and SuperNode.

FDEFine	**tracepoint_name address**

Where:

tracepoint_name	is the name of the tracepoint (up to 32 characters).
address	is the program store reference. Acceptable types include the following: • absolute byte address (up to six hex digits) • module_name procedure_name offset (offset defaults to beginning of procedure).

Responses

* token error * invalid character at column ____

Explanation: The user issued the FDEFine command, and the tokenizer encountered a character it did not recognize.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name and address.

*** ERROR *** Unstackfrn failed.

Explanation: A Command Interpreter (CI) error occurred when DEBUG was reading the FDEFine command from a DEBUG exec file ($READ command).

System Action: The command is not executed.

User Action: Reenter the command.

A tracepoint is already defined at this address
tracepoint definition failed

Explanation: A tracepoint has already been defined at the address specified in the command.

System Action: The command is not executed.

User Action: No user action is required.

address does not lie within a procedure

Explanation: The absolute byte address entered does not lie within a procedure.

System Action: The command is not executed.

User Action: Reenter the command, specifying an address within a procedure.

ALL is an invalid tracepoint name.

Explanation: The user issued the FDEFine command with ALL as a parameter.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid tracepoint name and address.

attempt to redefine existing tracepoint

Explanation: The user issued the FDEFine command specifying a tracepoint name that had already been defined.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name that has not yet been defined.

DIRECTORY ERROR, cannot add tracepoint

Explanation: An internal error occurred when the user issued the FDEFine command.

System Action: The command is not executed.

User Action: Reenter the command.

end of subcommand list of ____ items

Explanation: The user issued the FDEFine command, and the last command and the last subcommand for the tracepoint has been issued, denoted by entering a blank line.

System Action: No system action occurs; subcommand input is terminated.

User Action: No user action is required.

enter subcommands

Explanation: The tracepoint has been defined, and the user can now enter subcommands.

System Action: No system action is required.

User Action: Enter any tracepoint subcommands, ending the list by entering a null line.

enter tracepoint name or number

Explanation: The user issued the FDEFine command without specifying any parameters.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid tracepoint name and address.

expecting tracepoint name or number

Explanation: The user issued the FDEFine command without specifying any parameters.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name and address.

expecting program store address

Explanation: The user issued the FDEFine command without specifying an address.

System Action: The command is not executed.

User Action: Reenter the command, specifying a program store address.

failed to allocate directory for local binds

Explanation: DEBUG requested data store from the system, and the request was refused because data store on the switch was exhausted.

System Action: The command is not executed.

User Action: Wait and try the command again. (Hopefully, someone will release some data store, and the command will be executed.)

failed to allocate store for symbol table

Explanation: DEBUG requested data store from the system, and the request was refused because data store on the switch was exhausted.

System Action: The command is not executed.

User Action: Wait and try the command gain. (Hopefully, someone will release some data store, and the command will be executed.)

failed to allocate store for subcommands

Explanation: DEBUG requested data store from the system, and the request was refused because data store on the switch was exhausted.

System Action: The command is not executed.

User Action: Wait and try the command again. (Hopefully, someone will release some data store, and the command will be executed.)

illegal hex number
invalid module name

Explanation: The user issued the FDEFine command, specifying an illegal address or module name.

System Action: The command is not executed.

System Action: The subcommand is not executed.

User Action: Try the subcommand again.

unknown symbol ____ at column ____

Explanation: The user issued the TRACEBACK subcommand and specified extra parameters beyond the command.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying correct parameters.

User Action: Reenter the command, specifying a tracepoint name and address.

invalid program store address

Explanation: The address specified in the command is not an actual program store reference.

System Action: The command is not executed.

User Action: Reenter the command specifying a valid ps_reference.

invalid tracepoint subcommand

Explanation: The subcommand entered was not a valid tracepoint subcommand.

System Action: The command is not executed.

User Action: Enter a subcommand included in Tracepoint Subcommands on page 3-67.

max number of tracepoints already defined

Explanation: 50 tracepoints have already been defined.

System Action: The command is not executed.

User Action: Delete any tracepoints you are not using, and then reenter the command.

tracepoint ____ defined

Explanation: The user entered a null line to signify the end of the subcommand list.

System Action: The command is executed.

User Action: No user action is required.

unable to allocate store for tracepoint subcommands

Explanation: DEBUG requested data store from the system, and the request was refused because data store on the switch was exhausted.

System Action: The command is not executed.

User Action: Wait and try the command again. (Hopefully, someone will release some data store, and the command will be executed.)

unknown symbol ____ at column ____

Explanation: The user issued the FDEFine command and specified extra parameters beyond the command.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name and address.

Usage notes

A maximum of 50 tracepoints can be defined.

Examples:

1 The following command string defines a tracepoint at offset 3 in module LSETPRCI, procedure LINE_SETUP_PROCESSOR:

```
>FDEF TP1 LSETPRCI LINE_SETUP_PROCESSOR #3
```

2 The following command string defines a tracepoint at address 17459:

```
>FDEF TP1 #17459
```

GO

The GO command exits from the DEBUG utility. All tracepoint information collected will be kept, and active tracepoints will continue collecting information until they reach specified limits. QUIT is a synonym for GO.

GO	

Responses

* token error * invalid character at column ____

Explanation: The user issued the GO command, and the tokenizer encountered a character it did not recognize.

System Action: The command is not executed.

User Action: Reenter the command without specifying any parameters.

unknown symbol ____ at column ____

Explanation: The user issued the GO command and specified extra parameters beyond the command.

System Action: The command is not executed.

User Action: Reenter the command without specifying any parameters.

HELP

The HELP command gives syntax and usage notes for the specified command or subcommand. If no command or subcommand is specified, a general help panel is displayed.

HELP	[**command_name** **subcommand_name**]

Where:

command_name	is one of the following: • ACTIVATE • ALLOCATE • BIND • CLEAR • CPID • DEACTIVATE • DEFINE • DELBIND • DISPLAY • DISPLAY PS • EXTEND • HELP • LISTBIND • PRINT • QUIT • RESET • STATUS
subcommand_name	is one of the following: • ACTIVATE • BIND • DEACTIVATE • DISPLAY • EXITIF • SAVECCB • SAVECDB • SAVEEXT

- TIMESTAMP
- TRACEBACK

Responses

* token error * invalid character at column ____

Explanation: The user issued the HELP command, and the tokenizer encountered a character it did not recognize.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid command or subcommand name.

unknown debug command

Explanation: The user issued the HELP command and specified an unknown command or subcommand name.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid command or subcommand name.

unknown symbol ____ at column ____

Explanation: The user issued the HELP command and specified extra parameters beyond the command.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid command or subcommand name.

Examples:

1 The following command displays the general help screen:

```
> HELP
  HELP - valid at CI level
      HELP shows syntax and usage notes for a command
      or subcommand.  Commands may be executed at CI level;
      subcommands may be included in tracepoint definitions.
      This display is shown if no parameter is given.
      Parameters:  [ <command_name> | <subcommand_name> ]
      <command_name> is:  QUIT| HELP| DISPLAY| DISPLAY PS
         | DEFINE| EXTEND| ACTIVATE| DEACTIVATE| CLEAR
         | STATUS| BIND| PRINT| RESET| LISTBIND| DELBIND
         | LIMIT| ALLOC| CPID
      <subcommand_name> is:  DISPLAY| ACTIVATE| DEACTIVATE
         | TRACEBACK| BIND| EXITIF| TIMESTAMP| SAVECCB
         | SAVECDB| SAVEEXT
```

2 The following command displays the help screen for the DIsplay command:

```
>HELP DISPLAY
DIsplay storage _reference [FORMAT]  [N = nn]
storage_reference is:  start_address  [modifiers ......]

start_address is:  absolute address   (up to 6 hex digits)
                  module_name:PR| PP| SH (headsegments)
                  L| SB                      (local variables)
                  S                     (expression stack top)
                  R nn                       (base register nn)

modifiers are:   offset          (in words, up to 4 hex digits)

                 offset [ ,s [ ,w ] ]         (bit field width w,
                                  offset s bits from start of word)
               @    (dereference a pointer at address given)
                (index)| B (index) (index a desc or bit desc)
                     T n (index )| BT n (index)                  )
                          (index a table or bit table of stride n)
index is: &constant | storage_reference                          )
formats are DEC| HEX| ADDR| DESC| CHAR| typename
command valid at CI level and in tracepoint definition
```

LIMIT

The LIMIT command changes the maximum number of times a tracepoint can be hit. After a tracepoint is hit the maximum number of times, it is deactivated. The default limit is 25 times. The maximum number of times a tracepoint can be hit is 32767.

CAUTION

Numerous passes with large amounts of data being collected may result in the data first collected being overwritten.

LIMIT	**tracepoint_name [number_of_passes]**

Where:

tracepoint_name	is the name of an existing tracepoint.
number_of_passes	is a number between 1 and 32767.

Responses

* token error * invalid character at column ____

Explanation: The user issued the LIMIT command, and the tokenizer encountered a character it did not recognize.

System Action: The command is not executed.

User Action: Reenter the command, specifying valid parameter(s).

cannot set LIMIT for all tracepoints

Explanation: The user specified ALL as a parameter to the LIMIT command.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name and number of passes.

expecting positive number

Explanation: A negative number was specified for the number_of_passes parameter of the LIMIT command.

System Action: The command is not executed.

User Action: Reenter the command, specifying a positive number for the number of passes.

expecting tracepoint name

Explanation: The user issued the LIMIT command without specifying any parameters.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid tracepoint name.

old pass limit: ____ new pass limit: ____

Explanation: The user issued the LIMIT command. If no number_of_passes parameter was specified, the old pass limit and new pass limit are the same.

System Action: The command is executed.

User Action: No user action is required.

unknown symbol ____ at column ____

Explanation: The user issued the LIMIT command and specified extra parameters beyond the command.

System Action: The command is not executed.

User Action: Reenter the command, specifying valid parameter(s).

unknown tracepoint name

Explanation: The user attempted to change the maximum number of passes for a tracepoint that has not been defined.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name that has been defined.

Example:

The following command string limits the number of times tracepoint TP1 is hit to 75 times:

```
>LIMIT TP1 75
old pass limit:  25        new pass limit:  75
```

LISTBIND

The LISTBIND command displays all global BINDs that are defined in the system, as well as their data storage references. The output is in such a format that a command file can be created.

LISTBIND	

Responses

* token error * invalid character at column ____

Explanation: The user issued the LISTBIND command, and the tokenizer encountered a character it did not recognize.

System Action: The command is not executed.

User Action: Reenter the command without specifying any parameters.

unknown symbol ____ at column ____

Explanation: The user issued the LISTBIND command and specified a parameter.

System Action: The command is not executed.

User Action: Reenter the command without specifying any parameters.

Example:

The following command displays all global BINDS defined in the system:

```
>LISTBIND
BIND JTG TO OAUUI1:PR.60
```

PRINT

The PRINT command displays all data collected by the specified tracepoint or all tracepoints. If no parameter is specified, all tracepoints are displayed, along with the number of passes each tracepoint has made.

```
PRINT   [ tracepoint_name ]
        [ ALL             ]
```

Where:

tracepoint_name	is the name of an existing tracepoint.
ALL	specifies that all data collected by all tracepoints is to be displayed.

Responses

* token error * invalid character at column ____

Explanation: The user issued the PRINT command, and the tokenizer encountered a character it did not recognize.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name, ALL, or no parameters.

____: ____ passes made: ____ passes allowed

Explanation: The user issued the PRINT command specifying no parameters.

System Action: The command is executed.

User Action: No user action is required.

____: no data found, ____ passes made: ____ passes allowed

Explanation: The user issued the PRINT command without specifying any parameters. No tracepoint data has been collected for this session.

System Action: The command is executed.

User Action: No user action is required.

all data for this tracepoint has been overwritten

Explanation: The user issued the PRINT command with no parameters. Since the data buffer is common to all users, data is eventually overwritten.

System Action: The command is executed.

User Action: No user action is required.

earlier data has been overwritten

Explanation: The user issued the PRINT command with no parameters. Since the data buffer is common to all users, data is eventually overwritten.

System Action: The command is executed.

User Action: No user action is required.

The data collected for this
pass exceeds 1K limit.

Explanation: There is a limit of 1 Kbyte of data that can be collected for each subcommand.

System Action: The command is not executed.

User Action: Break the data collection into two or more tracepoints.

there is no saved data to print

Explanation: The user issued the PRINT command with no parameters. No tracepoint data has been collected for this session.

System Action: The command is executed.

User Action: Wait for the tracepoint to be hit and data collected.

unknown symbol ____ at column ____

Explanation: The user issued the PRINT command and specified extra parameters beyond the command.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name, ALL, or no parameters.

unknown tracepoint id

Explanation: Tracepoint name specified is not a defined tracepoint.

System Action: The command is not executed.

User Action: Reenter the command, specifying a defined tracepoint.

unknown tracepoint name

Explanation: Tracepoint name specified is not a defined tracepoint.

System Action: The command is not executed.

User Action: Reenter the command, specifying a defined tracepoint.

Examples:

1 The following command string prints all data collected by tracepoint TP1:

```
>PRINT TP1
TRACEPOINT TP1 AT 3A14C3:  LNUTILUI ALLOCATE OFFSET #0003
PASS 1
DI R 1

76D2B5:8051
```

2 The following command string prints all tracepoints and the number of passes each tracepoint made:

```
>PRINT
TP2:93 passes made:  100 passes allowed
TP1: 1 passes made:  100 passes allowed
```

QUIT

The QUIT command exits from the DEBUG utility. All tracepoint information collected will be kept, and active tracepoints will continue collecting information until they reach specified limits. GO is a synonym for QUIT.

QUIT	

Responses

* token error * invalid character at column ____

Explanation: The user issued the QUIT command, and the tokenizer encountered a character it did not recognize.

System Action: The command is not executed.

User Action: Reenter the command without specifying any parameters.

unknown symbol ____ at column ____

Explanation: The user issued the QUIT command and specified extra parameters beyond the command.

System Action: The command is not executed.

User Action: Reenter the command without specifying any parameters.

RESET

The RESET command resets the tracepoint counter for the specified tracepoint to zero. All data previously collected by the tracepoint is lost. The tracepoint will not change state (for example, from active to not active). If ALL is specified, the tracepoint counters for all tracepoints are reset.

RESET	**tracepoint_name** **ALL**

Where:

tracepoint_name	is the name of an existing tracepoint.
ALL	specifies the counters for all tracepoints are to be set to zero, and all previously collected data is lost.

Responses

* token error * invalid character at column ____

Explanation: The user issued the RESET command, and the tokenizer encountered a character it did not recognize.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name or ALL.

expecting tracepoint name or ALL

Explanation: The user issued the RESET command without specifying any parameters.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name or ALL.

unknown symbol ____ at column ____

Explanation: The user issued the RESET command and specified extra parameters beyond the command.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name or ALL.

Unknown tracepoint name

Explanation: The user issued the RESET command, specifying an undefined tracepoint name.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid tracepoint name or ALL.

Usage notes

The tracepoint remains in its current state (active or not active) following a RESET.

Examples:

1 The following command string resets the counter for tracepoint TP1:

```
>RESET TP1
```

2 The following command string resets the counter for all tracepoints defined:

```
>RESET ALL
```

SHELP

The SHELP command displays syntax and usage notes for commands and subcommands which are not for general use and not mentioned by the HELP command. Commands can be executed at the CI level of the MAP; subcommands can be included in tracepoint definitions.

SHELP	[**command_name** **subcommand_name**]

Where:

command_name	is one of the following:

- ACTIVATE
- ALLOCATE
- BIND
- CLEAR
- CPID
- DEACTIVATE
- DEFINE
- DELBIND
- DISPLAY
- DISPLAY PS
- EXTEND
- FDEFINE
- HELP
- LISTBIND
- PRINT
- QUIT
- RESET
- SHELP
- STATUS

subcommand_name	is one of the following:

- ACTIVATE
- BIND
- DEACTIVATE
- DISPLAY
- EXITIF

- SAVECCB
- SAVECDB
- SAVEEXT
- TIMESTAMP
- TRACEBACK

Responses

* token error * invalid character at column ____

Explanation: The user issued the SHELP command, and the tokenizer encountered a character it did not recognize.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid command or subcommand name.

unknown debug command

Explanation: The user issued the SHELP command and specified an unknown command or subcommand name.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid command or subcommand name.

unknown symbol ____ at column ____

Explanation: The user issued the SHELP command and specified extra parameters beyond the command.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid command or subcommand name.

Example:

The following command displays the general secret help screen:

```
> SHELP
SHELP - secret
     SHELP shows syntax and usage notes for a command
     or subcommand.  Commands may be executed at CI level;
     subcommands may be included in tracepoint definitions.
     Secret commands are undocumented and not to be publicized
     This display is shown if no parameter is given.
     Parameters:  [ <command_name> | <subcommand_name>|
                                                  <secret_name> ]
     <command_name> is:  QUIT| HELP| DISPLAY| DISPLAY PS
        | DEFINE| EXTEND| ACTIVATE| DEACTIVATE| CLEAR
        | STATUS| BIND| PRINT| RESET| LISTBIND| DELBIND
        | LIMIT| ALLOC| CPID
     <subcommand_name> is:  DISPLAY| ACTIVATE| DEACTIVATE
        | TRACEBACK| BIND| EXITIF| TIMESTAMP| SAVECCB
        | SAVECDB| SAVEEXT
     <secret_name> is:  SHELP| FDEFINE
```

STATUS

The STATUS command displays the name, location, activation state, and subcommand list of one or all tracepoints. The output is in such a format that a DEBUG exec can be created. A blank line is needed in the DBUG exec to terminate tracepoint subcommand definitions. [For more information on these execs, see DEBUG EXECS ($READ COMMAND) on page 3-2.] If no parameter is specified, the number of defined, active, and restart active tracepoints is displayed as well as the current tracepoint data and subcommand buffer sizes.

```
STATUS  [ tracepoint_name ]
        [ ALL             ]
```

Where:

tracepoint_name	is the name of an existing tracepoint.
ALL	specifies that the definition (including all subcommands) is to be displayed for all tracepoints.

Responses

* token error * invalid character at column ____

Explanation: The user issued the STATUS command, and the tokenizer encountered a character it did not recognize.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name, ALL, or no parameters.

active

Explanation: The user issued the STATUS command with the tracepoint name parameter, and the specified tracepoint is active.

System Action: The command is executed.

User Action: No user action is required.

deactivated: maximum passes made

Explanation: The user issued the STATUS command with the tracepoint name parameter, and the specified tracepoint is deactivated.

System Action: The command is executed.

User Action: No user action is required.

defined tracepoints ____; active ____; restart activate ____
Data buffer size: ____ words.
Subcommand buffer size: ____ subcommands.

Explanation: The user issued the STATUS command without specifying any parameters.

System Action: The command is executed.

User Action: No user action is required.

inactive

Explanation: The user issued the STATUS command with the tracepoint name parameter, and the specified tracepoint has been defined but not activated.

System Action: The command is executed.

User Action: No user action is required.

restart activation

Explanation: The user issued the STATUS command with the tracepoint name parameter, and the specified tracepoint is to be activated upon restart.

System Action: The command is executed.

User Action: No user action is required.

unknown symbol ____ at column ____

Explanation: The user issued the STATUS command and specified extra parameters beyond the command.

System Action: The command is not executed.

User Action: Reenter the command, specifying a tracepoint name, ALL, or no parameters.

unknown tracepoint name

Explanation: The tracepoint for which the user has requested the status is not defined in the system.

System Action: The command is not executed.

User Action: Reenter the command, specifying a valid tracepoint name.

Example:

The following command string displays the status of tracepoint TP1:

```
>STATUS TP1
define TP1 LNUTILUI CLOSE_LI 003   % at 3A16D1 inactive
DI R 40 N=120
```

Tracepoint subcommands

The subcommands in this section can be entered after a tracepoint has been defined. To add subcommands to a tracepoint that has already been defined, use the EXTend command.

ACTIVATE

The ACTivate subcommand activates a previously defined tracepoint so that it can begin collecting data. All tracepoints are deactivated following a restart.

ACTivate	**tracepoint_name**

Where:

tracepoint_name is the name of an existing tracepoint.

Responses

* token error * invalid character at column ____

Explanation: The user issued the ACTivate subcommand, and the tokenizer encountered a character it did not recognize.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying a valid tracepoint name.

ACT/DEACT ALL is an invalid subcommand

Explanation: The user issued the ACTivate subcommand, specifying ALL as a parameter.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying a defined tracepoint name.

expecting tracepoint name

Explanation: The user issued the ACTivate subcommand with no parameters.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying a tracepoint name.

unknown symbol ____ at column ____

Explanation: The user issued the ACTivate subcommand and specified extra parameters beyond the command.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying a tracepoint name.

unknown tracepoint

Explanation: The user attempted to activate a tracepoint that had not been defined.

System Action: The subcommand is not executed.

User Action: Define the tracepoint and then activate it, or activate a tracepoint that is already defined.

unknown tracepoint name

Explanation: The user attempted to activate a tracepoint that had not been defined.

System Action: The subcommand is not executed.

User Action: Define the tracepoint and then activate it, or activate a tracepoint that is already defined.

Usage notes

1. All tracepoints are deactivated following a restart.
2. ACTivate is both a tracepoint command and a subcommand, which means that a tracepoint can activate another tracepoint when the first tracepoint gets hit. This is useful if a tracepoint gets hit often, and the user is interested only when another piece of code is executed first.

Example:

The following command string within a tracepoint definition activates tracepoint TP1:

```
ACT TP1
```

BIND

The BIND subcommand assigns a symbolic name (bind name) to the specified storage reference. The name can now be used in further storage references but is only known within the tracepoint. Illegal bind names include names which can be interpreted as hex numbers or storage reference elements (for example, R, SH, or L). A tracepoint BIND is local to that tracepoint, but a BIND created at the CI level is global.

BIND	**bind_name [TO] storage_reference**

Where:

bind_name	is a string of up to eight characters, excluding certain values reserved for DEBUG, such as hex numbers and storage reference elements (1, 2, 3, ... D, E, F, L, PP, PR, R, S, SB, SH).
storage_reference	is the storage reference. Acceptable types include the following:

- absolute address
- base register
- S (expression stack top)
- SB or L (base of current stack frame)
- bind name
- module name followed by store type.

Store type can be any of the following:

- PR.offset (for protected global variables)
- PP .offset (for private global variables)
- SH.offset (for shared global variables).

Responses

* token error * invalid character at column ____

Explanation: The user issued the BIND subcommand, and the tokenizer encountered a character it did not recognize.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying a bind name and storage reference.

expecting bind_id

Explanation: The user entered the BIND subcommand without specifying any parameters, or the user entered a hexadecimal number for a bind name.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying a bind name and storage reference.

L,S,R,SB,SH,PR,PP are illegal bind names

Explanation: The user attempted to BIND using an illegal bind name or entered BIND and a storage reference without entering a bind name.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying a legal bind name.

unknown symbol ____ at column ____

Explanation: The user issued the BIND subcommand and specified extra parameters beyond the command.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying a bind name and storage reference.

Usage notes

When BIND is used as a tracepoint subcommand, the BIND is only known inside the tracepoint. If the same bind name is used both inside the tracepoint (local) and outside the tracepoint (global), the locally defined one has priority.

Example:

The following command string within a tracepoint definition binds storage reference OAUUI1:PR.60 to bindname JTG:

```
BIND JTG TO OAUUI1:PR.60
```

This results in an alternative reference for OAUUI1:PR.60. Compare the following two equivalent DIsplay commands:

```
> DI OAUUI1:PR.60 BT 4 (&0)
  08354D:   0008

> DI JTG BT 4 (&0)
  08354D:   0008
```

DEACTIVATE

The DEACTivate command deactivates tracepoints that have been previously defined and activated. A tracepoint is deactivated after a restart and once its limit is reached. For more information on tracepoint limits, refer to LIMIT on page 3-52.

DEACTivate	**tracepoint_name**

Where:

tracepoint_name is the name of an existing tracepoint to be deactivated.

Responses

* token error * invalid character at column ____

Explanation: The user issued the DEACTivate subcommand, and the tokenizer encountered a character it did not recognize.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying a tracepoint name.

ACT/DEACT ALL is invalid subcommand

Explanation: The user issued the DEACTivate subcommand, specifying ALL as a parameter.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying a tracepoint name.

expecting tracepoint name

Explanation: The user issued the DEACTivate subcommand with no parameters.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying a tracepoint name.

unknown symbol ____ at column ____

Explanation: The user issued the DEACTivate subcommand and specified extra parameters beyond the command.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying a tracepoint name.

UNKNOWN TRACEPOINT NAME

Explanation: The user attempted to deactivate a tracepoint that had not been defined.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying a tracepoint that is already active.

Example:

The following command string within a tracepoint definition deactivates tracepoint TP1:

```
DEACT TP1
```

DISPLAY CCBPTR, ECCBPTR, CDBPTR, and RUNPPTR

CCBPTR, ECCBPTR, CDBPTR, and RUNPPTR (Runningproc) are pointers to their respective blocks. To access a field in the block, the pointer must first be dereferenced.

DIsplay	**pointer_name@**

Where:

pointer_name
is one of the following pointers:

- CCBPTR
- ECCBPTR
- CDBPTR
- RUNPPTR

Responses

* token error * invalid character at column ____

Explanation: The user issued the DI pointer subcommand, and the tokenizer encountered a character it did not recognize.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying a pointer name.

unknown symbol ____ at column ____

Explanation: The user issued the DI pointer subcommand and specified extra parameters beyond the command.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying valid pointer name.

Example:

The following command string within a tracepoint definition displays the contents of the CPMBPTR field of the CCB:

```
DI CCBPTR@.0 ADDR
476F1C:  48821C
```

DISPLAY (data store)

The DIsplay command is used to display values in data and program store. This section describes how to use the DIsplay command for displaying values in data store. For information on displaying program store, refer to DISPLAY PS (Program Store) on page 3-35.

Note: Refer to Data Store on page 5-3 for the differences in data store on the NT40 and SuperNode.

DIsplay	**storage_reference** [**store_index**] [**size**] [**format**]

Where:

storage_reference

is the storage reference. Acceptable types include the following:

- absolute address (up to six hex digits), optionally preceded by # (for example, #39AE4)
- base register other than R 0 (in the format R nn; for example, R 4)

  This option is especially useful at the tracepoint level.
- S (expression stack top); n words below the top of the stack (for example, S.-2 or S.-#2)
- SB or L (base of current stack frame); local variable at offset nn (for example, L.nn or L.#nn or SB.nn or SB.#nn)
- bind name (as defined using the BIND command or subcommand)
- module name followed by store type.

Store type can be any of the following:

- PR.offset (for protected global variables)
- PP .offset (for private global variables)
- SH.offset (for shared global variables).

Offset can be up to four hex digits.

store_index can be any of the following:

- .nn (word offset on NT40 or byte offset on SuperNode)
- .nn, b, w (word offset on NT40 or byte offset on SuperNode, bit offset, bit width)

This is a reference to a bit offset within a word. If no bit width is specified, the default is the rest of the word. This accesses a bit field of width w bits, at offset nn words, b bits from the given address. Another representation for this reference is the following.

.nn OFB b W w.

- .nn@ (dereferencing a pointer at the address given)
- .nn (i) (indexing descriptors)
- .nn B (i) (indexing bit descriptors)
- .nn T x (i) (indexing a table given the stride; x defaults to 1)
- .nn BT x (i) (indexing a bit table given the stride; x defaults to 1)
- .nn TB x (i) (indexing a bit table given the stride; x defaults to 1).

Where:

- nn=word offset on NT40 or byte offset on SuperNode
- b=bit offset
- w=bit width
- x=table stride
- i=index in table.

i can be either of the following:

- &number (any numeric value)
- storage_reference (to at most one word of store; see previous explanation).

size	is the number, in bytes (form N=nn), of items to be displayed. If not specified, the rest of the procedure is displayed.
format	is one of the following output formats:

- DEC (decimal output)
- HEX (hexadecimal output)
- ADDR (six-digit hexadecimal address)
- DESC (address - size - stride)
- CHAR (readable character string)

- CCB (Display is formatted as Call Condense Block if the DISP_CALL_PROCS are bound in. Otherwise, it is dumped in hex. Attempting to format non-CCB data as a CCB may cause a trap.)
- CDB (Display is formatted as a Call Data Block if the DISP_CALL_PROCS are bound in. Otherwise, it is dumped in hex. Attempting to format non-CDB data as a CDB may cause a trap.)
- EXT (Display is formatted as an Extension Block if the DISP_CALL_PROCS are bound in. Otherwise, it is dumped in hex. Attempting to format non-EXT data as an EXT may cause a trap.)
- typename (for Data Dictionary format). This is mainly used to display I/O type data. All type names known by DDEDIT are allowed.

Responses

Because there are numerous system responses for the DIsplay command, they are listed here without explanations. Most of the responses are self-explanatory.

```
* token error * invalid character at column ____
cannot do word offset on bit field
data is too large for tracepoint buffer
desc, addr formats only apply to word items
Desc stride must be entered as D n
expecting =
expecting base register number
expecting bit offset
expecting bit width
expecting closing parenthesis
expecting decimal number
expecting hexnumber
expecting MOD:,PP,PR,SH or L,SB,R,S,& number, or addr
expecting number
expecting offset
expecting opening parenthesis
expecting PP,PR or SH
expecting storeref
failed to scan transitive uses list
illegal base register number
illegal hex number
illegal hexnumber
illegal index
```

illegal indexing
invalid base register number
invalid bit stride
invalid dec number
invalid dereference
Invalid Ecore bit descriptor syntax
invalid format or N
invalid module name
invalid perprocess reference
modifier cannot operate on one word of a register
modifier cannot operate on this register
negative address not accepted
PP only valid for perprocess module
PP reference at CI level requires module name
PR reference at CI level requires module name
protected segment does not exist
SH reference at CI level requires module name
shared segment does not exist
the first modifier must be @
too many digits
too many modifiers
typename not found
unable to allocate store
unknown bind name
unknown symbol
unknown symbol ____ at column ____

Usage notes

1. Recursive indexing or dereferencing is allowed.
2. Invalid indexing or dereferencing gives an error message.
3. Indexing or dereferencing cannot follow bit offset.
4. Further qualification of indexing or dereferencing is possible.

Example:

Refer to Chapter 4 on page 4-1 for examples of displaying data store.

EXITIF

The EXITIF subcommand exits tracepoint processing if the specified relationship is true. By putting IF statements in the tracepoint, this subcommand helps the user to reduce the collected data to only those conditions in which he is interested.

> ***Note:*** No error message appears if the evaluation of either argument fails. The subcommand is simply ignored.

> ***Note:*** Storage references used as arguments may not be larger than one word. This includes long word registers on SuperNode. Refer to EXITIFs on page 5-11 for differences in the EXITIF command on the NT40 and SuperNode.

EXITIF	**argument operator argument**

Where:

argument	is each storage reference or number to be compared. Acceptable types include the following: • number • absolute address • base register • S (expression stack top) • SB or L (base of current stack frame) • bind name • module name followed by store type. Store type can be any of the following: - PR.offset (for protected global variables) - PP .offset (for private global variables) - SH.offset (for shared global variables).
operator	is one of the following relational operators: • = (equal to) • < (less than) • > (greater than) • ≠ (not equal to) • <= (less than or equal to) • >= (greater than or equal to) • +> (unsigned greater than)

- +< (unsigned less than).

Responses

* token error * invalid character at column ____

Explanation: The user issued the EXITIF subcommand, and the tokenizer encountered a character it did not recognize.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying valid parameters.

expecting relational operator

Explanation: The user failed to specify a relational operator.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying two arguments, separated by a relational operator.

expecting MOD:,PP,PR,SH or L,SB,R,S,& number, or addr

Explanation: The user entered the EXITIF subcommand without specifying two arguments and a relational operator.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying two arguments, separated by a relational operator.

expecting storeref

Explanation: The user entered the EXITIF subcommand, specifying one argument and a relational operator.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying two arguments, separated by a relational operator.

invalid relational operator

Explanation: The user entered the EXITIF subcommand, specifying an invalid relational operator or no relational operator.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying two arguments, separated by a valid relational operator.

unknown symbol ____ at column ____

Explanation: The user issued the EXITIF subcommand and specified extra parameters beyond the command.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying correct parameters.

Usage notes

If one of the storage references is constant, it must be prefaced with an ampersand.

Example:

The following command string exits tracepoint if the storage reference L.#1C0 is greater than zero:

```
EXITIF L.#1C0 > &0
```

SAVECCB

The SAVECCB subcommand saves a copy of the Call Condense Block (CCB) for the active call. The CCB is displayed in symbolic form or in hex form (if DISPCALL is absent).

SAVECCB	

Responses

* token error * invalid character at column ____

Explanation: The user issued the SAVECCB subcommand, and the tokenizer encountered a character it did not recognize.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand without specifying any parameters.

unknown symbol ____ at column ____

Explanation: The user issued the SAVECCB subcommand and specified extra parameters beyond the command.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand without specifying any parameters.

Usage notes

1. This subcommand is only accepted in offices where call processing software is loaded. If base register 4 is NIL when the tracepoint is hit, an error message will be displayed when the data is printed. Note also that base register 4 can point to data other than CCBs when not in a call processing environment.
2. Only four different CCBs and CDBs can be saved before the oldest one is overwritten.
3. This subcommand is only valid in the context of a call process (CALLP).

Example:

The following is printout of a SAVECCB subcommand. Because of the print format, the lines that are too long to be printed are split into two lines.

```
SAVECCB
CPTLB :
  LINK/CPMBPTR   =   48821C    MYINDEX              =    27  00
```

```
                                                    PROCQD          =  N
 STATE            =  LINKED    AUDIT                =      0000
                                                  LINKCOUNT     =  0001
 LETTERCOUNT    =      0003    LETTERC              =      C102
                                                  WAKEID          =  FFFF
 LETTERQ          =  FFFFFF    CCBTIMEQ.SUCC      =  FFFFFF
                                               CCBTIMEQ.PREV  =  FFFFFF
 FASTECCBINDEX =       0000    UP_OVER_WARM       =  N
                                                 ECCBINDEX      =  00   00
EXTPTR            =  FFFFFF    SEQNO            =  0000    CS   =
                                                 ORIGIN XBITS      =  0000
FORCEUNAVAIL  =  N             IBN              =  N       SA    =
                                          N          FASTSEQNO       =  0020
CMI               =      0000   CTRLPORT        =  0000    CCBFC  =
                                                                     0000
RECEIVER          =  <NIL>
CCBFA  :
 0008   0010

PORT1PERM  :
 AGENT     =   <NIL>
 PATHEND  :
  CHNL   =   0000    PORT  =   0000    NM_PAIR       =   0000    NM_CHNL
                                           =   0000       LOGICAL  =   0001
  GAIN   =   0000    LOSS  =   0000    INTEG_VAL     =   00FF
                                       PREFERRED_PL   =   EVEN_PLANE
PMCHNL  =          200     PORT        =   01    CFWBIT          =   N
                                                  AGENT_SUSPECT   =   N
THREAD  =        00C7     FMTCODE   =   05    UTR_AVAILABLE    =   N
TID          :  NODE_NO    =   044       TRMNL_NO_MSN  =    0
                                         TRMNL_NO_LSB   =    09   TSI   =   0
FMTAREA:
 EAE1  6F00  EAE0  6F00  4ABF  6E00  FE08  FFFF  FFFF  FFFF
MBUFFPTR  =  0007B8

PORT2PERM  :
 AGENT     =   <NIL>
 PATHEND  :
  CHNL   =   0000    PORT  =   0000    NM_PAIR       =   0000    NM_CHNL
                                           =   0000       LOGICAL  =   0001
  GAIN   =   0000    LOSS  =   0000    INTEG_VAL     =   00FF
                                       PREFERRED_PL   =   EVEN_PLANE
PMCHNL  =          400     PORT        =   02    CFWBIT          =   N
                                                  AGENT_SUSPECT   =   N
THREAD  =         0000     FMTCODE   =   00    UTR_AVAILABLE    =   N
TID          :  NODE_NO    =   000       TRMNL_NO_MSN  =    0
```

```
                                     TRMNL_NO_LSB  =   00   TSI  =  0
FMTAREA:
 FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF
MBUFFPTR  =  FFFFFF

CHB  :
 XLAB:
  RC               =   XLA_MO        TXSEL          =      0000
                                             TX_TRMT          =
  IBN_TRMT_SET  =   N                MIN_DIGITS    =      0000
                                             MAX_DIGITS      =   0012
  PREFIX_FENCE  =       0000         DATA_VER      =   N
                                             OC                =   LCL
  TRAN_SYS         =   NA            TYPECALL      =   NP
                                             NPA_ADDED       =   N
  TX_POS            =   NONE         TXROUTE/TERM_AGENT  =  <NIL>
  CALLED_DR        =
  RTE                 =   N           POS              =   N
                                             RETRANSLAT      =   N

  CALLING_DR/AUTH_CODE_DR  =
  SCREENING_IN  =   N                CAN_LCS       =   N
                                             CAN_SCRN        =   Y
  XLASTAGE        =   INITIA         ADP              =   N
                                             BLK_OVLP        =   N
  LONGHAUL        =   N              HTRP             =       0
                                             SNPA              =   518
  XLT_FROM         =       0         HTRC             =   N
                                             VALID_/PRIVL    =   N
  ANI_INFO  :
   ONI    =   N      HOT     =   N       TDN    =   N      COIN    =   N
   RSP    =   N    ANI_FAIL =   N    SPARE2  =   N      SPARE3   =   N
  AMADATA  :
   0000   0000   0000   0000   0000   0000   0000   0000

  SOURCEPARMS  :
   VALID_SOURCEPARMS  =   N          NCOS             =      0000
                                       DESTIN                  =   NIL_DE
   CUSTGRP                  =          SUBGRP          =      0000
                                       CALL_CHARACTER    =      0000
   DGCOLL_TABLE          =      0000   SOURCE          =   NIL_SO
                                       OWAT_ZONE             =      0000
   SOURCE_TRC            =      0000   SMDR              =   N
                                       SMDRB                  =   N
   ACR                         =   N   INTRAGROUP  =   N
                                       ENABLE_CRL           =   N
   CRL_REQUIRED          =   N         ATTDNDOV      =   N
```

```
                                        DNDPREEMPT          =   N
  CALLED_DR_SHIFTED  =   N              TDN            =   N
                                        DOD_DESTIN          =   N
  GROUP_INTERCOM     =   N              ATTX           =   N
                                        LINE_HAS_LNR        =   N

 RTEB:
  ROUTE          =   FDFDFD   RC                    =   RTR_IN
                                   RTE_TYPE         =   IBN_RT
  ROUTE_CHAIN    =   IN_CHA   PREV_SATELLITE    =   N
                                   CHOICE           =   DIRECT
  OHQT_APPLIED   =   N        QUEUEING_ACTIVE  =   N
                                   OHQ              =   N
  CANCSTDL       =   N        INHIBIT_QUEUEING =   N
                                   HUNT             =   Y

 TRMT          =   UNDT      POS                =      0000
                                        TFR                 =   NIL_TF
 ANI_SPILL     =   N         ANI_SPILL_9       =   N
                                        RECORDING_REGD      =   N

 TS_OMREG       =   N      SEIZE_FAILURE        =   N
                                        AC_FTR_IN_EFFECT    =   N
RESELECT        =   N
                                        DIST_RING           =   N
 FTR_IN_EFFECT  =   N
 LCO_CALL       =   N      CHARGE_TREATMENT =   CHGA
 FORCE_CC_TIMI  =   N      AC_ATTACHED          =   N
 EAEO_DATA  :
  EA_CARRIER  =   ATT   EA_LOCAL    =   N    EA_OCS_NEEDED  =   N
  CALL_EVENT  =   INITIAL_EVENT

 ECHOSUP    =   <NIL>
 OVLPD:
  REALCMI    =   0     DRCOUNT   =   0      OLSTATE  =   OLFIRST
 PSTATE  :  PROCESSOR    =   NIL_PRCR  STATE    =   0000
 MBI           =    0000   CHARGE      =   N       DATE_CALL  =   N
                                        TIMESTAMP      =     0021  AD74
 ORIGDISP   =    0000   TERMDISP    =    0000  OCC_CALL   =   N
                                        OCC_INCOM      =     N
 NO_INTRAS =  N          EA_CALL     =   N       PIC_CALL    =   N
                                        LATA_CALL      =     INTRALATA
 STATE_CAL =  INTRAS   TRD_TIMIN    =   NO_TRD  TPS_CP    =   N
```

SAVECDB

The SAVECDB subcommand saves a copy of the Call Data Block (CDB) for the active call. The CDB is displayed in symbolic form or in hex form (if DISPCALL is absent).

SAVECDB	

Responses

* token error * invalid character at column ____

Explanation: The user issued the SAVECDB subcommand, and the tokenizer encountered a character it did not recognize.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand without specifying any parameters.

unknown symbol ____ at column ____

Explanation: The user issued the SAVECDB subcommand and specified extra parameters beyond the command.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand without specifying any parameters.

Usage notes

1. This subcommand is only accepted in offices where call processing software is loaded. If base register 5 is NIL when the tracepoint is hit, an error message will be displayed when the data is printed. Note also that base register 5 can point to data other than CDBs when not in a call processing environment.
2. Only four different CCBs and CDBs can be saved before the oldest one is overwritten.
3. This subcommand is only valid in the context of a call process (CALLP).

Example:

The following is a printout of a SAVECDB subcommand. Because of the print format, the lines that are too long to be printed are split into two lines.

```
SAVECDB
CPMB:
  LINK/CPTLBPTR    =   3E1248
  MAILBOX:
    MBQLINK         =   FFFFFF   MBPROCQD     =   N    CLONED =   N
                                                       MBINDEX     =      0000
    MBLETTERCOUN =      0002    MBPROCPTR    =   4BEDA9
                                                  MBLETTERQ     =   FFFFFF
  STATE   =   CPMBLINKED     MILTIWAIT   =   N
STATUS   =   PROCESSING

MESSAGE  :
  MT   =   000C   -   ORIGINATION_MSG
  BODY:
    C000   0000   D800   0000   4C20   1400   0000   D61E   0454   C94A
    0554   7364   D618   0000   0000   0000   0000   0000   0000   5553
    4342   4359   454C   2020   4215
  TID   =   NODE_NO   =   044      TRMNL_NO_MSN   =   0
                                                 TRMNL_NO_LSB        =   09
TEMPTIMESTAMP   =   010   AD74    NETWORK_CONNECTION_MADE  =   N
                                                       NCCLS        =   0

CDBRTEB:
 GROUP_CPID         = HUNT   GRP     200   MEM      3:
                          LEN   REM2   02   0   15   12     DN   6212057
 TERMINAL_CPID      = LEN   REM2   02   0   15   12     DN6212057
 CONNECTION_TYPE=  D   REROUTE_ELEMENT  =  N  CST_L                =N
 CST_H              = N   RECREATE_CDBRTEB   =  Y   ERWT_ACCEPTED
=N
 WAIT_FOR_COT_TO=  N   GRPSC                   =  Y
FAST     PROC   =   0
```

SAVEEXT

The SAVEEXT subcommand saves up to ten extension blocks of the given type (or all types) for the active call. The EXT is displayed in symbolic form unless hex is specified or no symbolic procedure is built into the load.

SAVEEXT	**format_code** [**HEX**] **ALL**

Where:

format_code	is the name of an extension block type.
ALL	specifies that all types of extension blocks are to be displayed.
HEX	specifies that the extension block is to be displayed in hexadecimal form.

Responses

* token error * invalid character at column ____

Explanation: The user issued the SAVEEXT subcommand, and the tokenizer encountered a character it did not recognize.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying a format code or ALL.

<format_code> is: ____

Explanation: The user issued the SAVEEXT subcommand, specifying the format code parameter.

System Action: The subcommand is executed.

User Action: No user action is required.

<format_code> is the name of an extension block type.

Explanation: The user issued the SAVEEXT subcommand and specified an invalid format code.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying a valid format code.

expecting extension block parameters

Explanation: The user entered the SAVEEXT subcommand without specifying any parameters.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying a format code or ALL.

Only one saveext subcommand per tracepoint.

Explanation: The user attempt to issue two SAVEEXT subcommands for a single tracepoint.

System Action: The subcommand is not executed.

User Action: No user action is required.

unknown symbol ____ at column ____

Explanation: The user issued the SAVEEXT subcommand and specified extra parameters beyond the command.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying a valid format code or ALL.

Usage notes

1. This subcommand is only accepted in offices where call processing software is loaded. If base register 6 is NIL when the tracepoint is hit, an error message will be displayed when the data is printed. Note also that base register 6 can point to data other than EXTs when not in a call processing environment.
2. Only four different EXTs can be saved before the oldest one is overwritten.
3. This subcommand is only valid in the context of a call process (CALLP).

TIMESTAMP

The TIMESTAMP subcommand displays the current time when the tracepoint is executed. This is often the first subcommand entered. The time printed can be used to correlate logs with a tracepoint hit.

TIMESTAMP	

Responses

* token error * invalid character at column ____

Explanation: The user issued the TIMESTAMP subcommand, and the tokenizer encountered a character it did not recognize.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand without specifying any parameters.

unknown symbol ____ at column ____

Explanation: The user issued the TIMESTAMP subcommand and specified extra parameters beyond the command.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand without specifying any parameters.

Example:

Following is an example of output from the TIMESTAMP subcommand.

```
TIMESTAMP
TIME 1988/04/08 16:24:31.498
```

TRACEBACK

The TRACEBACK subcommand saves a specified number of procedure calls, saving a specified number of words of data at each call.

TRACEBACK	[**number_of_levels** [**words_per_level**]]

Where:

number_of_levels
is the number of procedure calls to be saved. The default is 4.

words_per_level level is the number of words of data to be saved at each call. The default is 16.

Responses

* token error * invalid character at column ____

Explanation: The user issued the TRACEBACK subcommand, and the tokenizer encountered a character it did not recognize.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying the correct parameter(s) or no parameters.

expecting positive number

Explanation: The user entered values for number_of_levels and/or words_per_level that were less than one.

System Action: The subcommand is not executed.

User Action: Reenter the subcommand, specifying values greater than or equal to one for number_of_levels and/or words_per_level.

Procedure not found at ____

Explanation: This response occurs when there is a stack frame error or when there are no more calling procedures.

System Action: The subcommand is not executed.

User Action: Try the subcommand again.

Traceback terminated.

Explanation: This response occurs when there is a stack frame error or when there are no more calling procedures.

Usage notes

The user will get x + 1 (number_of_levels + 1) stack frames (one for the procedure and x calling procedures). If, for a stack frame, the words_per_level is larger than the number of parameters and locals, then the stack frame saved will be the correct size. In other words, it is possible to get a partial stack frame, but it is not possible to get more than a full stack frame.

Example:

The following is a printout of sample output from the TRACEBACK subcommand without any parameters:

```
TRACEBACK

LNUTILUI:ALLOCATE+#0003   IC=3A14C3   SB=000E56
   3C09   28FF   0001   1B79   FFFF   006E   005A   001F
   1AA4   003C   0030   0011   0001

XLALCZI:XL_ORIG_+#0119   IC=3C1B79   SB=000E41
   0001   3C09   28FF   125C   3E00   1264   3E00   0000
   0000   0001   BF06   4800   6E00   0000   001F   001C

KSETALCI:KSET_ORI+#0344   IC=6EF21A   SB=000E1E
   0001   3C09   28FF   125C   3E00   3C19   2808   001C
   0E19   9ED0   6E00   009C   001A   0000   0001   0008

LSETPRCI:LINE_SETUP_P+#019F   IC=3B2813   SB=000E06
   3C09   28FF   00C7   A03B   3E00   0001   0000   0000
   AFDA   00C7   0000   1264   3E00   2910   4A00   EAE0

LNSTART:LINE_STA+#049A   IC=3B1D32   SB=000DF8
   3C09   28FF   0001   3C09   28FF   125C   3E00   0001
   0000   3E00
```

Displaying data store

The DISPLAY command is used to display data. This part describes several methods for displaying data. First, the more simple forms of data display are explained. Next, how to get friendly output, followed by methods to display complex data structures are explained.

All addresses, offsets, field widths, and base register numbers are interpreted as hexadecimal numbers. However, they may optionally be preceded by an octothorpe (#). Constants and repetition values are interpreted as decimal numbers unless they are preceded by an octothorpe (#). Strides are also interpreted as decimal numbers unless preceded by an octothorpe (#). A stride is the width of an element of a table.

Note: Refer to Data store on page 5-3 for a description of the differences in data store in the NT40 and SuperNode.

Format of DEBUG output from DISPLAY command

This section describes the format of DEBUG output following the DIsplay data store command.

When a DIsplay command is used, the output format is HEX, unless otherwise specified. The following are possible output formats for the DIsplay command:

DEC:	decimal output, eight items per line, preceded by the hex address of the first item. Items are printed in ascending address order.
HEX:	hexadecimal output, eight items per line, preceded by the hex address of the first item. Items are printed in ascending address order.
ADDR:	six hex digits per line, preceded by the hex address of the first item. Items are printed in ascending address order.
DESC:	three-word items (address-size-stride), one item per line, preceded by the hex address of the descriptor. Items are printed in ascending address order.

CHAR:	continuous readable character string of up to 45 characters per line, preceded by the hex address of the first character. Characters are printed in ascending address order.

Multiword table items:

- Stride of eight or less: single item per line, preceded by the hex address of the first item on each line. Items are printed in ascending address order.
- Stride greater than eight: eight words per line, with a new line for the start of each item. Each line is preceded by the hex address of the first item on each line. Items are printed in ascending address order.

CCB:	display is formatted as a Call Condense Block if the DISP_CALL_PROCS are bound in. Otherwise, it is dumped in hex. Attempting to format non-CCB data as a CCB may cause a trap.
CDB:	display is formatted as a Call Data Block if the DISP_CALL_PROCS are bound in. Otherwise, it is dumped in hex. Attempting to format non-CDB data as a CDB may cause a trap.
EXT:	display is formatted as an Extension Block if the DISP_CALL_PROCS are bound in. Otherwise, it is dumped in hex. Attempting to format non-EXT data as an EXT may cause a trap.
typename:	used for Data Dictionary format. This is mainly used to display I/O type data. All type names known by DDEDIT are allowed.

Note: It is possible to add types to the data dictionary using DDEDIT. From DDEDIT, type HELP ADDTYPE.

In all cases the length of the output line is less than the width of the MAP conversation area. An example of each output format is given in Figure 4-1 on page 4-3 .

Refer to Figure Figure 4-8 on page 4-11 for additional examples of displaying data using different output formats.

Figure 4-1xxx
Using different output formats

- No output format specified (default is HEX):

```
>DI LOGS:
PR.3A    016F48:  5645
```

- Output format DEC specified:

```
>DI LOGS:
PR.2D DEC   01701B:  -515
```

- Output format ADDR specified:

```
>DI LOGS:
PR.3A ADDR   016F48:  005645
```

- Output format DESC specified:

```
>DI LOGS: PR.3C DESC
016F4A:   address   6F4F4C   stride  4C   size   0108
```

- Output format CHAR specified:

```
>DI 083523 N=3 CHAR
083523:  OAU
```

- Output format TYPENAME (CARD_CODE) specified:

```
>DI OAUUI1:
SH.A CARD_CODE   0617BC:    2X57AA
```

- CCB format is the same as the format of the SAVECCB command output (see SAVECCB on page 3-81).

- CDB format is the same as the format of the SAVECDB command output (see SAVECDB on page 3-85).

- EXT format is the same as the format of the SAVEEXT command output (see SAVEEXT on page 3-87).

Specifying an offset in a module

The LCOPY listing of a module on NT40 (or LISTER on SuperNode) gives the offsets of declarations in private, shared, and protected store defined in that module. To use this offset in order to display the data of those declarations, type one of the following command strings.

DI <module_name>:PR.offset
for protected global variables
DI <module_name>:PP.offset
for private global variables
DI <module_name>:SH.offset
for shared global variables

It is also possible to enter successive offsets, as in the following command string: DI module_name:

PR.offset.offset.offset

Note: Offsets are words on the NT40 and bytes on SuperNode.

Because private data is specific to a process, the use of: PP references is only allowed in tracepoints. Moreover, only modules which are in the transitive uses list of the tracepointed module may be referenced, so that DEBUG can be certain that the private segment located is the one in context. (This check is done at the time the tracepoint is defined.) A private segment is in context only if it can be reached by following the links which the loader inserts at the top of each private segment, beginning with the one currently addressed by the private segment register (base register 3, or A3 on SuperNode).

The context private segment for a module is usually (but not always) the segment addressed by the latest occurrence of the module in the Perprocess Segment Table (PST). Many of the private segments addressed by the PST will not be accessible using DEBUG because the modules to which they belong are not transitively used by the tracepointed module. If you want to reference the latest PST entry for a module, and DEBUG will not let you refer to it using: PP, you must use the QUERY CIBINCOM to find the private segment address, then use absolute address format for the reference.

No private data references are allowed in tracepoints at which the private segment register does not address the tracepointed module’s private segment (for example, between a PRECALL instruction and the following POSTCALL instruction). (This check is done at the time the tracepoint is hit.)

The following response is from module TRKDUI.

```
PROTECTED   nil_trunk_index   trunk_index;                      001.0
%%          -----------

PROTECTED   trunk_nwm_perm_data DESC
             (|trunk_group_number|) OF nwm_area;           001.0 003.0

PROTECTED   trunk_group_reservedlist_data DESC
        (|trunk_group_number|) OF
               trunk_group_reservedlist_data_type;         004.0 003.0

PROTECTED ktnildesc DESC OF kt_member_data;                007.0 003.0

PROTECTED   mem_var_area_type_id,                          010.0 001.0
            tm2_member_type_id,                            011.0 001.0
            tm4_member_type_id,                            012.0 001.0
```

All of the parameters from the previous listing are defined in protected store. The offset and the length are given in the last two columns at the right hand side of the listing. The commands in Figure 4-2 on page 4-5 are used to display parameters.

Figure 4-2xxx
Displaying data using offsets

- Enter the following command string to display trunk_nwm_perm_data:

  ```
  >DI TRKDUI:
  PR.1 N=3  09713C:  01F4  630A  3ED2
  ```

- Enter the following command string to display trunk_group_reservedlist_data:

  ```
  >DI TRKDUI:
  PR.4 N=3   09713F:  01F4  4F03  6A98
  ```

- Enter the following command string to display mem_var_area_type_id, tm2_member_type_id and tm4_member_type_id:

  ```
  >DI TRKDUI:
  PR.10 N=3   09714B:  0389  FDFD  FDFD
  ```

Refer to Displaying a field in the CCB on page 4-25 for another example using successive offsets.

Displaying contents of an absolute address

The following command string displays the contents of an absolute address:

DI absolute address

Figure 4-3 on page 4-6 provides examples of displaying the contents of an absolute address.

Figure 4-3xxx
Displaying contents of an absolute address

```
•  The following command string displays the
   contents of location 5C5400:

   >DI #5C5400
   5C5400:  01CA

•  The following command string displays the
   contents of location 5C5400 plus the next
   19 (decimal) words:

   >DI 5C5400 N=20
   5C5400:  01CA  9083  0015  136B  0012  19A4  0033  2610
   5C5408:  01CA  9083  01CA  9083  01CA  9083  0014  0BCD
   5C5410:  01CA  9083  0009  1971
```

Displaying a base register

The following command string displays the contents of the address where a register is pointing:

DI R n

where n is any hex number of a base register, except 0.

> ***Note:*** DI R 0 is invalid, because R0 always points to the current stack base. In this case, use DI SB instead. (See Displaying local variables and parameters on page 4-8.)

The register may be pointing to a data block (for example, CDB or CCB). It is often pointing to a module head segment. Using a base register for displaying data is practiced mainly in tracepoint subcommands. See Figure 4-4 on page 4-7 for examples of command usage.

Figure 4-4xxx
Displaying data using a base register

- The following command string displays the contents of the memory pointed to by the address contained in base register 4:

```
>DI R 4
76D2B5:  8051
```

- The following command string displays the contents of a data block to which base register 4 is pointing:

```
>DI R 4 N=3
76D2B5:  8051   8054   8057
```

In the following list, registers are shown pointing to their corresponding data. The current head segments are the head segments of the running module unless they have been loaded in preparation for a procedure call by a LDREG or PRECALL instruction.

R 1 --->	current protected head segment
R 2 --->	current shared head segment
R 3 --->	current private head segment
R 4 --->	the CCB if the process running is CALLP
R 5 --->	the CDB if the process running is CALLP
R 40 --->	the process control block of the currently running process

Note: Refer to Display CCBPTR, ECCBPTR, CDBPTR, and RUNPPTR on page 3-73 for more information on displaying the contents of R 4, R 5, and R 40.

Note: Registers in the NT40 have different names than registers in SuperNode. For SuperNode register names, refer to Registers on page 5-4.

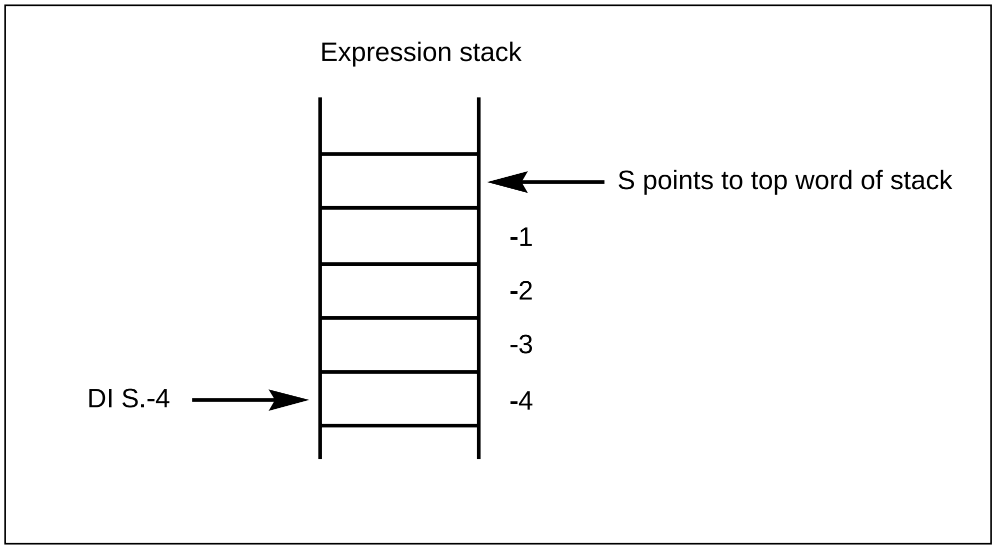
Displaying the expression stack

To display values at the expressions stack, use the DISPLAY command with the S parameter, as follows:

DI S

The display command DI S points to the top of the stack (see Figure 4-5 on page 4-8).

Figure 4-5xxx
Displaying the expression stack



To display the stack, use a negative value, as shown in Figure 4-6 on page 4-8.

Figure 4-6xxx
Displaying the expression stack

- `The following command string displays the top word of the stack:`

  ```
  >DI S
  000DEF:   0006
  ```

- `The following command string displays the last 5 words that are pushed on the stack:`

  ```
  >DI S.-4 N=5
  000DEB:  0DD2    CB7D   0000    0000    0006
  ```

Note: Refer to Expression stack on page 5-3 for differences in displaying the expression stack in the NT40 and SuperNode.

Displaying local variables and parameters

Local variables and parameters are stored on the call stack. Either of the following commands will display values at the stack:

DI L
DI SB

These commands are most useful in a tracepoint subcommand because at that time the register is pointing to that part of the call stack which has the most interest.

The following listing is from module IOLOG:

```
DCL fill_device_name PROC (nodeno  node_number,                    001.0
   UPDATES devclass  TABLE (|0 TO 7|) OF char,             001.0   004.0
          UPDATES devnum    int) IS                        003.0   001.0
BLOCK
 DCL space       TABLE (|0 TO 15|) OF char,                005.0   008.0
   device_name  device_names;                              00d.0   004.0
```

> ***Note:*** REF and UPDATES parameters are always two words long. REF and UPDATES create an address on the stack that points to the item’s data area. The size shown in the listing (far right column) is the size of the item’s data area referred to by the address on the stack.

This listing indicates the following:

- parameter NODENO is defined at offset 0 and is 1 word long
- update parameter DEVCLASS is stored at offset 1 as a pointer to a data area of 4 words long
- update parameter DEVNUM is stored at offset 3 as a pointer to a data area of 1 word long
- local variable SPACE is a table at offset 5 and 8 words long
- local variable DEVICE_NAME is stored at offset #D and is 4 words long.

To display these local variables it is necessary to set a tracepoint in this procedure at the point where the variables are expected to contain interesting data, and issue the DIsplay subcommands. See Figure 4-7 on page 4-10 for examples. See Displaying update parameters on page 4-23 for an example displaying the updates parameters DEVCLASS and DEVNUM.

Figure 4-7xxx
Displaying local variables and parameters

- The following command string displays parameter NODENO:

```
>DI SB
77BBC5:  BBAA
```

- The following command string displays local variable SPACE:

```
>DI L.5 N=8
77BBCA:  5446   4749   4541   4342   4541   4549   4342   4544
```

- The following command string displays local variable DEVICE_NAME:

```
>DI L.D N=4
77BBD2:  DB34   DAFE   DE34   DE54
```

Note: Refer to Local variables on page 5-3 for differences in displaying the local variables in the NT40 and SuperNode.

Output of multiple items

As stated before, every DI command can optionally be followed by:

<size> <format>

Note: The parameters <size> and <format> can be interchanged. So <format> <size> is the same as the format shown here. For simplicity only the <size> <format> syntax is used in this document.

If a DIsplay command is followed by N=10, then decimal 10 items will be displayed. The item size is determined by what is displayed, according to the following order of priority:

1 the output format, if it describes items not of word size (for example, ADDR)
2 the table or DESC stride, if an indexing operation was performed
3 the bit field width, if a bit field was specified.

Some examples to clarify this order of priority are shown in Figure 4-8 on page 4-11.

The same addresses are used in the examples in Figure 4-8 on page 4-11 in order to compare the different output types.

Figure 4-8xxx
Displaying multiple items

- No output format is specified. The item size is 1 word.

```
>DI TRKDUI: SH.3 N=20
063928:  0EBE  0012  1368  0033  2C28  01CA  9083  01CA
063930:  9083  01CA  9083  0014  0D35  01CA  9083  0009
063938:  0FDB  01CA  9083  003B
```

- The output format is ADDR. The item size is 2 words (3 bytes shown).

```
>DI TRKDUI: SH.3 ADDR N=4
063928:  000EBE  001368  012C28  019083
```

- The output format is DESC. The item size is 3 words.

```
>DI TRKDUI: SH.3 DESC N=3
063928:  address  001368  stride 12  size 0EBE
06392B:  address  2C01CA  stride 28  size 0033
06392E:  address  019083  stride CA  size 9083
```

- The output format is a bit field. The bit field starts at bit 0, and the width is 4 bits.

```
>DI TRKDUI: SH.3,0,4 N=10
063928:  000E  000B  000E  0000  0002  0001  0000  0000
06392A:  0008  0006
```

Using the dereference operator @

To display the contents of a location pointed to by a pointer, use dereferencing. Figure 4-9 on page 4-12 shows a pointer pointing to the contents of a location.

Only a data location containing a valid address can be dereferenced. An invalid dereference will not cause a trap, but a message is displayed. Multiple dereferences are possible.

The following example of dereferencing shows a display of module TRKDUI, the fourth word of shared store for 10 words.

```
>DI TRKDUI: SH.3 N=10
5C5403:  136B  0012  19A4  0033  2610  01CA  9083  01CA
5C540B:  9083  01CA
```

Another example of dereferencing is given in Displaying queue problems on page 4-24.

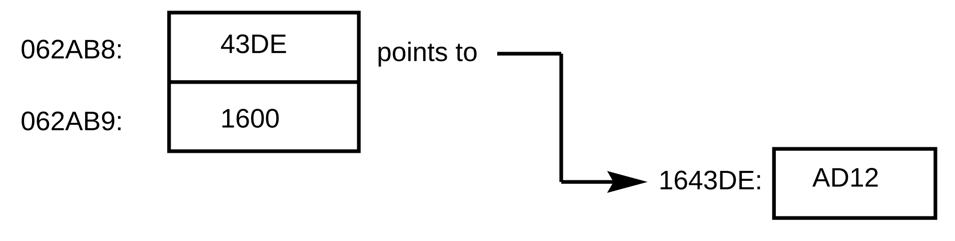
Assume that the fourth and fifth words of shared store of module TRKDUI are a pointer. The following command string displays the contents of the word (plus the next 19 words) where this pointer is pointing:

```
>DI TRKDUI: SH.3@ N=20
00136B:  2031   2020   0002   4B4F   2020   4157   2049   2020
001373:  2020   2020   1413   3811   D211   E0CC   7D06   190C
00137B:  0000   0000   5652   3231
```

Next assume that the fourth word of shared store of module TRKDUI is a pointer which points to a pointer. The following command string displays the contents of the word where the final pointer is pointing:

```
>DI TRKDUI: SH.3@@
202031:  0004
```

Figure 4-9xxx
Displaying the contents of a location



- The following command string displays the contents of the pointer:

```
>DI 62AB8 ADDR
062AB8:   1643DE
```

- To display the contents of the address to which the pointer points, use the dereference operator with the following command string:

```
>DI 62AB8@
1643DE:   AD12
```

Displaying fields in a structure

The offsets in a structure, as given in an LCOPY listing, can be used to display data of the items in that structure using one of the following command strings:

```
DI <module_name>: PR.offset % protected global variables
DI <module_name>: PP.offset % private global variables
DI <module_name>: SH.offset % shared global variables
```

It is also possible to enter successive offsets, as in the following command string:

```
DI module_name: PR.offset.offset.offset
```

The following display is part of the listing of TRKDUI:

```
TYPE trunk_members_prot                                     005.0
STRUCT
     tmid trmnl_id,                                         001.8
     digdbrecv        digital_db_level,             001.8   000.4
     digdbtrans       digital_db_level,             001.c   000.4
     digtrk           BOOL,                         002.0   000.1
     sgrp trunk_subgroup_number,                    002.1   000.1
     in_only          BOOL,                         002.2   000.1
     data_present     BOOL,                         002.3   000.1
     offline          BOOL,                         002.4   000.1
     halfpmc          pm_channels,                  002.5   000.9
     name external_trunk_name,                      003.0   000.e
     suppl_data       BOOL,                         003.e   000.1
     intrasw_chnl     BOOL,                         003.f   000.1
     halfpe           int                           004.0   001.0
ENDSTRUCT;
```

Assume that this TYPE is declared in TRKDUI at offset 4 in protected store. TRUNK_MEMBERS_PROT can be displayed as in Figure 4-10 on page 4-14. Note that the BOOL names are not written in. See Figure 4-11 on page 4-14 for examples of displaying data using an offset.

Figure 4-10xxx
TRUNK_MEMBERS_PROT display

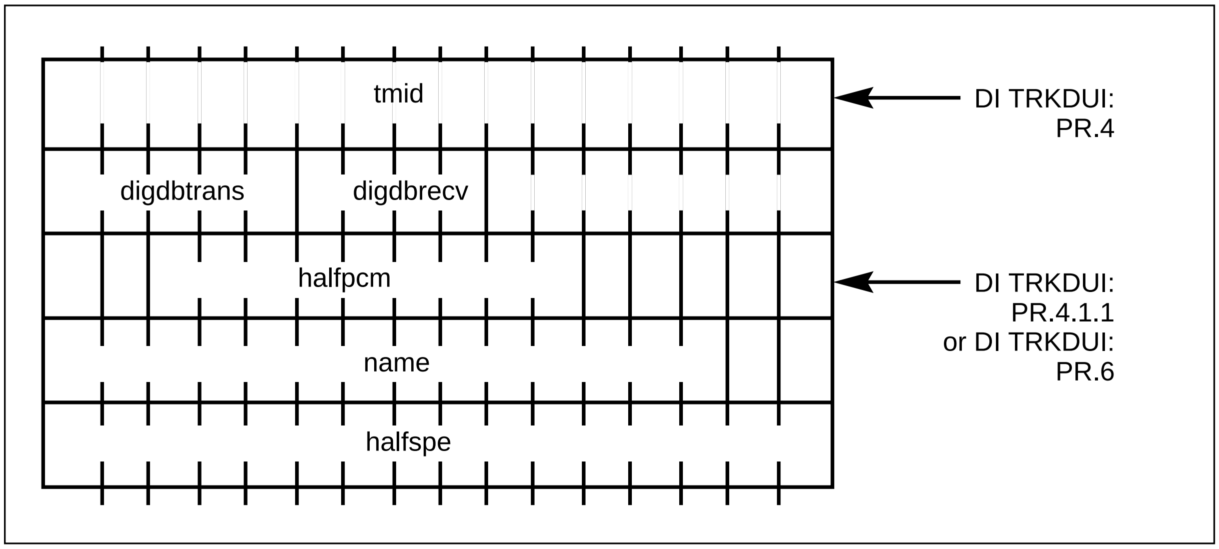


Figure 4-11xxx
Displaying fields in a structure

- The following command string displays the first word of trunk_members_prot:

  >DI TRKDUI: PR.4
  6D4FD5: 4219

- The following command string displays the third word of trunk_members_prot by using successive offsets:

  >DI TRKDUI: PR.4.1.1
  6D4FD8: 03E8

- The following command string displays the complete data block trunk_members_prot:

  >DI TRKDUI: PR.4 N=5
  6D4FD5: 4219 80C0 03E8 2909 851C

Bit field selection

When selecting a bit field, the user can specify the word offset, bit offset, and bit width, as in the following command string:

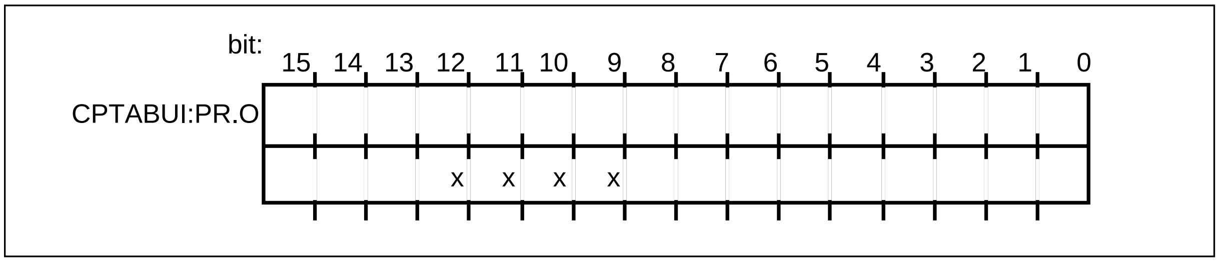
DI storage_reference .nn,b,w

Where:

nn	is the word offset on NT40 or byte offset on SuperNode,
b	is the bit offset (defaults to 1),
w	is the width (defaults to rest of word).

Figure 4-12 on page 4-15 shows how bits 9 - 12 of the second word of protected store are stored in module CPTABUI.

Figure 4-12xxx
Bits 9-12 of Second Word of Protected Store



The following command string displays the contents of the bits marked with an x in Figure 4-12 on page 4-15:

```
>DI CPTABUI: PR.1,9,4
```

The following listing from module OAULI will be used in an example showing how to display a boolean.

```
TYPE oau_sd_group_data
00a.1
        STRUCT
                tid             trmnl_id,                               001.8
                cos             card_code,                      001.8   000.8
                sd_pts          TABLE(|ext_sd_point|)
                                  OF PACK(16) oau_sd_funct_desig,       002.0

                data_present    BOOL                            00a.0   000.1
        ENDSTRUCT;
```

This structure is in store at address 3DF69F.

Because the boolean data_present must be displayed, use bit field selection, as shown in Figure 4-13 on page 4-16.

Figure 4-13xxx
Displaying data using bit selection

```
•  First, use bit field selection.  The word offset is A,
   the bit offset 0, and the bit width is 1 bit.

   >DI 3DF69F.A,0,1
   3DF6A9:   0001

•  To determine if this boolean is false or true,
   use the type name BOOL, which gives:

   >DI 3DF69F.A,0,1 BOOL
   3DF6A9:   Y
```

Indexing a table

To index a table, the stride must be known and input in decimal. It is possible to index word size or multiword size and bit tables.

Note: Refer to Tables on page 5-5 for differences in displaying the tables on the NT40 and SuperNode.

Word size table indexing

The following command string indexes a word size table:

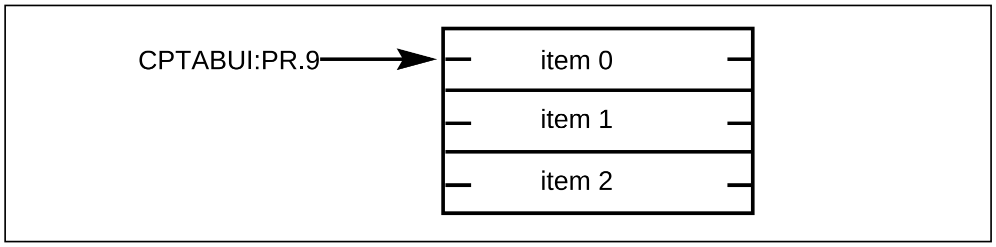
DI storage_reference .nn T x (i)

Where:

nn	is the word offset.
T	indicates word size table indexing.
x	is the size of the stride in DECIMAL (defaults to 1).
i	is the index of the desired element in the table. The index can be a constant or a storage reference (for example, (S.-2)). If the index is a constant, it is displayed as (&decimal value) or (&#hex value). If the index is a storage reference to more than one word of store, it is an error. If no index is specified (for example, if () is entered), an index of 0 is assumed.

Refer to the example from CPTABUI shown in Figure 4-14 on page 4-17. The stride of the table is two words.

Figure 4-14
Word size table indexing



The following command displays the contents of item 1:

```
>DI CPTABUI: PR.9 T 2 (&1)
```

The following example shows how to use word size table indexing. In module IOLI the following statements are found:

```
SHARED roundtripinfo TABLE(|0 TO 16|)
                    OF unsignedint;                    122.0    011.0
%%      ----------
%% used to store distribution of round trip message times
```

The stride is one word. The following command string displays the third item in this table:

```
>DI IOUI: SH.122 T 1 (&#3)
00035A:   0000
```

Bit table indexing

The following command string indexes a bit table:

```
DI storage_reference .nn BT x (i)
DI storage_reference .nn TB x (i)
```

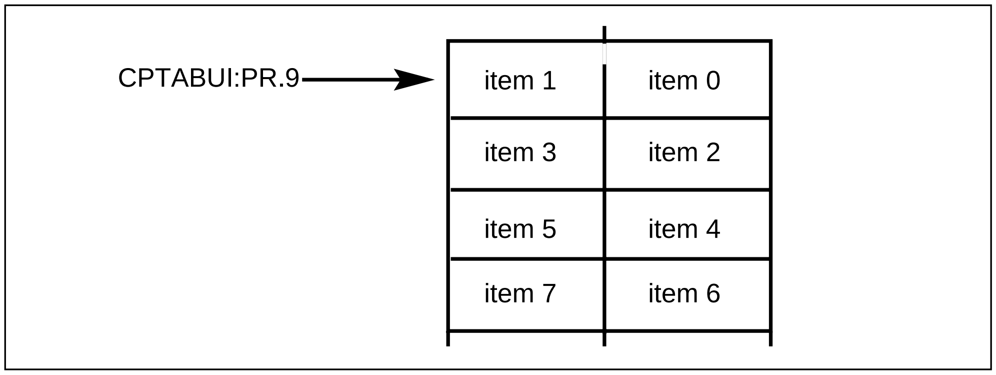
Where:

nn	is the word offset.
BT or TB	indicates bit table indexing.
x	is the size of the stride in DECIMAL (defaults to 1).

i is the index of the desired element in the table. The index can be a constant or a storage reference (for example, (S.-2)). If the index is a constant, it is displayed as (&decimal value) or (&#hex value). If the index is a storage reference to more than one word of store, it is an error. If no index is specified (for example, if () is entered), an index of 0 is assumed.

Refer to the example from CPTABUI shown in Figure 4-15 on page 4-18. The stride of the table is 8 bits.

Figure 4-15xxx
Bit Table Indexing



The following command displays the contents of item 5 of Figure 4-15 on page 4-18:

```
>DI CPTABUI: PR.9 BT 8 (&5)
057253:  0000
```

In module OAUUI1 the following declaration in protected store is found:

```
PROTECTED
  ncpalarm_reset_table
                        TABLE (|0 TO 6|) OF byte,              05d.0    0004.0
  ncpalarm_2x57aa_table
                        TABLE (|0 TO 19|) OF byte,             061.0    000a.0
  ncpalarm_table
                        TABLE (|0 TO 19|) OF byte,             06b.0    000a.0
  dead_system_alarm_table
                        TABLE (|0 TO 19|) OF byte,             075.0    000a.0
  dead_system_alarm_2x57aa_table
                        TABLE (|0 TO 19|) OF byte;              07f.0    000a.0
```

One of the following command strings will display four elements in table NCPALARM_RESET_TABLE, in which the offset in protected store is #5D and the stride is eight bits:

```
>DI OAUUI1: PR.5D BT 8 (&0) N=4
08354A:   0002    0008    0000    0000

or:

>DI OAUUI1: PR.5D TB 8 (&0) N=4
08354A:   0002    0008    0000    0000
```

Indexing a descriptor

DEBUG easily displays store referenced by a descriptor. When indexing the descriptor, the stride does not need to be given, because DEBUG reads it from the descriptor. (This is true for the NT40 but not for SuperNode.)

> ***Note:*** Refer to Descriptors on page 5-6 for differences in displaying the tables on the NT40 and SuperNode.

It is possible to display both word size tables and bit size descriptors.

Indexing a word descriptor

The following command string indexes a word descriptor:

```
DI storage_reference .nn (i)
```

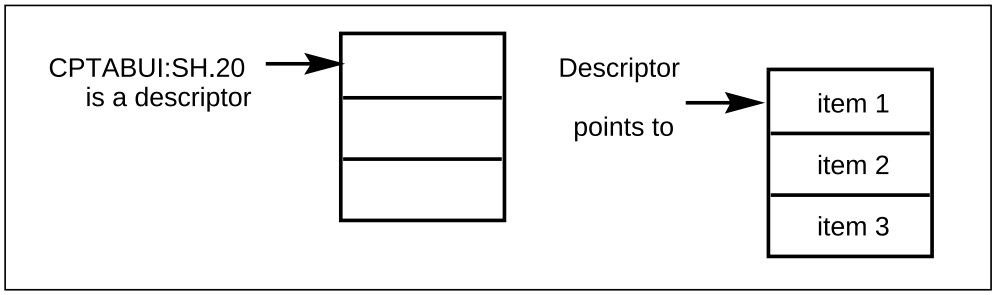
Where:

nn is the word offset.

i is the index of the desired element in the table. The index can be a constant or a storage reference (for example, (S.-2)). If the index is a constant, it is displayed as (&decimal value) or (&#hex value). If the index is a storage reference to more than one word of store, it is an error. If no index is specified (for example, if () is entered), an index of 0 is assumed.

Refer to the table example in Figure 4-16 on page 4-20. The descriptor is in shared store at offset 20 in module CPTABUI.

Figure 4-16
Displaying the third item of table



The following command displays the contents of the third item of the table shown in Figure 4-16 on page 4-20:

```
>DI CPTABUI: SH.20 (&2)
000D40:  0D34   000C   0D24   000F   2CD8   0019   00B9   0008
         FFFF   34BE   1770   1568   7FFE   1D4D   0008   0008
         0D0B   34BF   4901   0769   0000   5ADE   0000   0000
         000C   0000   000F   00B9   0008   0008   0D0B
```

Descriptors can be used recursively. For example, a descriptor can point to a descriptor of descriptors. In the preceding example, if item 1 is a descriptor, then the following command will display the fourth entry of the table pointed to by item 1.

```
>DI CPTABUI: SH.20 (&0) (&3)
000003:  0D00
```

Observe the following code from module OAUUI1:

```
TYPE oau_sd_group_data
00a.1
     STRUCT
          tid            trmnl_id,                                001.8
          cos            card_code,                       001.8   000.8
          sd_pts         TABLE(|ext_sd_point|) OF
                          PACK(16) oau_sd_funct_desig,    002.0   008.0
     data_present        BOOL                             00a.0   000.1
  ENDSTRUCT;

PROTECTED
     oau_sd_grp          DESC OF DESC OF
                                 oau_sd_group_data,       042.0   003.0
     oau_sdgp_unprot     DESC OF
                                 oau_sdgp_unprot_data,    045.0   003.0
     oau_sd_point        DESC OF DESC OF
                                 oau_sd_point_data,       048.0   003.0
     oau_sd_counts       DESC OF DESC OF
                                 int;                     04b.0   003.0
```

The following procedure steps through the declaration for OAU_SD_GRP.

1 At offset #42 in protected store a DESC OF DESC OF oau_sd_group_data is defined. Observe the following command:

```
>DI OAUUI1: PR.42 DESC
084975:   address   386D46   stride  03   size   0004
```

This indicates that the descriptor at offset #42 of module OAUUI1 in protected store points to a table with four elements in it, each element three words long. The stride must be three words, because the descriptor at offset #42 points to a table of descriptors.

2 Now display the first entry in this table of descriptors, as follows:

```
>DI OAUUI1: PR.42 (&0) DESC
386D46:   address   3B7085   stride  0B   size   0040
```

This entry points to a table with #40 elements, each element #B words in size. This table is of type OAU_SD_GROUPDATA. The declaration of this type is given in the previous example of code from OAUUI1.

3 The TID in type OAU_SD_GROUP_DATA can easily be displayed by using the descriptors and type name TRMNL_ID. In both tables the first entry is used, as follows:

```
>DI OAUUI1: PR.42 (&0) (&0) TRMNL_ID
3B7085:   36  0   1
```

4 Also the card code can be displayed (using bit field selection) as follows:

```
>DI OAUUI1: PR.42 (&0) (&0).1,8,8 CARD_CODE
3B7086:    3X82AB
```

In the following example a storage reference is used to index a descriptor. The storage reference S has the value 6, so the second value is 6.

```
>DI S 000DEF:  0006

>DI IOUI: PR.2E9 (&42) (S ) CP_ID
0E899B:                  CKT      RCVRATD    5
```

In this example a storage reference (the top of stack) is used as the second index. In this case the second index is 6.

Indexing a bit descriptor

The following command string indexes a bit descriptor:

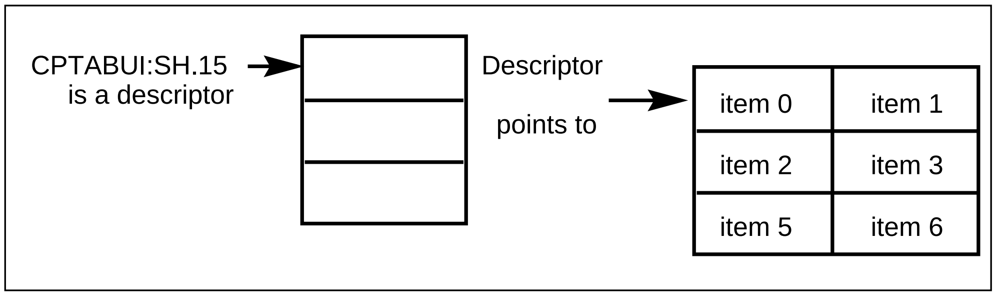
DI storage_reference .nn B (i)

Where:

nn	is the word offset.
B	indicates a bit descriptor is indexed.
i	is the index of the desired element in the table. The index can be a constant or a storage reference, for example, (S.-2). If the index is a constant, it is displayed as (&decimal value) or (&#hex value). If the index is a storage reference to more than one word of store, it is an error. If no index is specified (for example, if () is entered), an index of 0 is assumed.

Refer to the bit table example in Figure 4-17 on page 4-22. The descriptor is in shared store at offset 15 in module CPTABUI.

Figure 4-17xxx
Displaying the third item of bit table



The following command displays the contents of item 2 in Figure 4-17 on page 4-22:

```
>DI CPTABUI: SH.15 B (&2)
```

The following listing is from module IOLI:

```
TYPE alias_index (|0 TO max_alias_index|);                        000.8

PROTECTED node_aliases DESC OF alias_index;               321.0   003.0
  %% allocated [0 to max_node_no] in perm store
```

node_aliases is a DESC OF type alias_index, which has a size of 8 bits. The stride can be verified with the following command:

```
>DI IOUI: PR.321 DESC
02197E:   address   207E44   stride  08   size   0080
```

The following command uses a bit descriptor and displays 10 items starting at item 3.

```
>DI IOUI: PR.321 B (&3) N=#A
207E45:   0000    0000    0000    0000    0000    0000    0000    0000
207E49:   0000    0000
```

Examples of displaying data

Displaying update parameters

The following listing is from module IOLOG:

```
DCL fill_device_name PROC (nodeno  node_number,                   001.0
    UPDATES devclass  TABLE (|0 TO 7|) OF char,           001.0   004.0
             UPDATES devnum    int) IS                    003.0   001.0
BLOCK
  DCL space       TABLE (|0 TO 15|) OF char,              005.0   008.0
     device_name  device_names;                           00d.0   004.0
```

REF and UPDATES parameters are always two words long. REF and UPDATES create an address on the stack that points to the item’s data area. The size shown in the listing is the size of the item’s data area referred to by the address on the stack.

Updates parameter DEVCLASS points to a data area four bytes long. The following command displays this data area:

```
>DI SB.1@ N=4
43523A:   D289    7600    B321    80FF
```

Updates parameter DEVNUM points to a data area of one byte. The following command displays this byte:

```
>DI SB.3@
7894BC:   0023
```

Displaying queue problems

This is an example of recursive dereferencing.

In modules QUEUES, IOUI and IOLI the following declarations are found:

```
Module QUEUES:
TYPE q1way PTR TO q1way;                                   002.0

Module IOUI:
TYPE cmciobuffer_q q1way;                                  002.0

Module IOLI:
TYPE cmciobuffer_q_data                                    007.0
  STRUCT
    cbd          DESC OF cmciobuffer,                      003.0
    cbh          PTR TO cmciobuffer_q,             003.0   002.0
    owner        id                                005.0   002.0
  ENDSTRUCT;

SHARED cmciobuffer_qs DESC OF cmciobuffer_q_data; 065.0 003.0
```

Assume that the user wants to display the contents of the address where CBH, defined in type CMCIOBUFFER_Q_DATA, is pointing. This can be accomplished by displaying offset #65 in IOUI using the descriptor. Offset this value by 3, because CBH is at offset 3 in CMCIOBUFFER_Q_DATA, and then dereference this value using the following command string:

```
 >DI IOUI: SH.65 (&1).3   @
 01D347:  F4E6
```

From the preceding listings, note that CBH is a pointer of type CMCIOBUFFER_Q, which is of type Q1WAY, which, in turn, is a pointer of type Q1WAY. Hence, a pointer to a pointer is created. Recursive dereferencing can now be used to follow the pointers, as shown in Figure 4-18 on page 4-25.

Figure 4-18xxx
Recursive dereferencing

```
>DI IOUI:SH.65 (&1).3 @@ ADDR
10F4E6:   10F357

>DI IOUI:SH.65 (&1).3 @@@ ADDR
10F357:   10F3DC

>DI IOUI:SH.65 (&1).3 @@@@ ADDR
10F3DC:   10F461

>DI IOUI:SH.65 (&1).3 @@@@@ ADDR
10F461:   10F4E6

(Now we are back at the same address where we started.)

>DI IOUI:SH.65 (&1).3@ @ @ @ @ @ ADDR
10F4E6:   10F357
```

Displaying a field in the CCB

Assume the user wants to know the contents of a particular field in the CCB. By using successive offsets, any field can be displayed. The following example shows how the NCOS field is displayed. First, all the offsets are collected from module CPDATAUI.

```
TYPE  call_condense_block
  chb                    call_handler_block,                  03c.0   02c.8

TYPE call_handler_block
%%     --------------
  STRUCT
  xlab                   translation_block,                           024.0

TYPE translation_block   PACK (36*16) STRUCT
  sourceparms            source_parms_area                    01f.0   005.0

TYPE source_parms_area
%%     ---------------
UNRESTRICTED AREA (5*16)
  valid_sourceparms BOOL,                                             000.1
  ncos                   network_class_of_service,            000.1   000.8
```

From this example the following information can be extracted:

- CHB is at offset #3c in the CCB.

- XLAB is at offset 0 in the CHB
- Sourceparms is at offset #1f in the XLAB.
- Field NCOS is a bit field in sourceparms at offset 0, bit offset 1 and bit size 8.
- Register 4 points to the CCB during call processing.

The example in Figure 4-19 on page 4-26 shows how the NCOS was displayed after a tracepoint was hit in the line error processor.

Figure 4-19xxx
Displaying a field of the CCB

```
>DEFINE E1 LLERPRCI LL_ERROR 0
enter subcommands
>DI R 4.3C.0.1F.0,1,8           %NCOS

end of subcommand list of 1 items
tracepoint E1 defined
>ACT E1
tracepoint E1 activated
>PRINT E1
TRACEPOINT E1 AT 3D8003:
                LLERPRCI LL_ERROR OFFSET #0003
PASS 1
di r 4.3c.0.1f.0,1,8  %ncos
685B02:     0000
```

Differences between DEBUG on the NT40 and SuperNode

This part describes the differences between DEBUG on the NT40 and the version of DEBUG on SuperNode.

On the SuperNode, DEBUG can run on any Software Operating System (SOS) node.

Program store

On the NT40, program store is byte-addressable and byte-accessible. On the SuperNode, program store is byte-addressable and word-accessible.

Because PROTEL was written for the architecture of the NT40, SuperNode requires more memory than the NT40 for the same amount of work. However, SuperNode is faster than the NT40. See Figure 5-1 on page 5-2 for an example of the same procedure for the NT40 and SuperNode.

Figure 5-1xxx
Displaying program store on NT40 versus SuperNode

```
NT40:

>di ps     loctabui     change_block_id
           127BBA:      0000BPROC             L=#02 S=#04
           127BBD:      0003PUSH2             #000C(B04)4
           127BC1:      0007SAVE2L            O=#02
           127BC3:      0009NILP
           127BC4:      000ATRUES             #1A
           127BC6:      000CPUSHAS            #2(SB)
           127BC8:      000EPUSHS             #0(SB)
           127BCA:      0010PUSHS             #1(SB)
           127BCC:      0012PUSHB@@L          O1=#2 O2=#2 E=#0 W=#7
           127BCF:      0015SHIFT             E=#F W=#0 (LEFT #1)
           127BD1:      0017PUSHT2            #0025(B01) L=#0100
           127BD7:      001DCALL@
           127BD8:      001ELDREG             SSTO=#04AA
           127BDB:      0021PUSH2@@L          O1=#2 O2=#0
           127BDD:      0023JUMPSB            #1D
           127BDF:      0025PEND

SuperNode:

>di ps     loctabui     change_block_id
082AEA1C.0000:          LINK.W    A5,#-$C
082AEA20.0004:          CMPA.L    ($40F000).L,SP
082AEA26.000A:          TRAPCS
082AEA28.000C:          MOVE.W  D0,-$6(A5)
082AEA2C.0010:          MOVE.W  D1,-$2(A5)
082AEA30.0014:          MOVEA.L D7,A6
082AEA32.0016:          MOVE.L    $18(A6),-$C(A5)
082AEA38.001C:          MOVEA.L -$C(A5),A6
082AEA3C.0020:          CMPA.L    #-$10000.L,A6
082AEA42.0026:          BEQ.B     $82AEA72
082AEA44.0028:          LEA -$C(A5),A0
082AEA48.002C:          MOVE.W  -$2(A5),D1
082AEA4C.0030:          MOVE.W  -$6(A5),D0
082AEA50.0034:          BFEXTU   $4(A6){0:6},D6
082AEA56.003A:          MOVEA.L ($4E7304).L,A4
082AEA5C.0040:          CHK.W     #$3F,D6
082AEA60.0044:          MOVEA.L ($F34,A4,D6.W*4),A6
082AEA66.004A:          JSR       (A6)
082AEA68.004C:          MOVEA.L -$C(A5),A6
082AEA6C.0050:          MOVE.L    (A6),-$C(A5)
082AEA70.0054:          BRA.B     $82AEA38
082AEA72.0056:          UNLK      A5
082AEA74.0058:          RTS
082AEA76.005A:          NOOP
```

Data store

On the NT40, data store is word-addressable, word-accessible, and word-indexible. On the SuperNode, data store is byte-addressable, word-accessible, and word-indexible.

Compare the DIsplay commands in Figure 5-2 on page 5-3 to see the differences between NT40 and SuperNode.

Figure 5-2xxx
Displaying data store on NT40 versus SuperNode

NT40:

```
>DI TRKDUI:PR.1 N=3
```

SuperNode:

```
>DI TRKDUI:PR.2 N=3    % Notice the byte addressing
```

Note: **In both cases, N=3 is referring to 3 words.**

There is one exception to SuperNode data store being byte-addressable. When displaying bit fields (less than 16 bits), data store becomes word-addressable. You must convert the byte offset to a word offset by dividing the byte offset by two and use it (word offset) as the data store offset into which you give the starting bit and number of bits to display. See Figure 5-3 on page 5-3 for an example.

Figure 5-3xxx
Displaying data store (bit fields) on SuperNode

- To look at byte A, bits 4 through 6 of module CPTABUI’s protected data store, enter the following command string:

```
>DI CPTABUI:PR.5,4,3    % The 5 is A divided by 2.
```

Expression stack

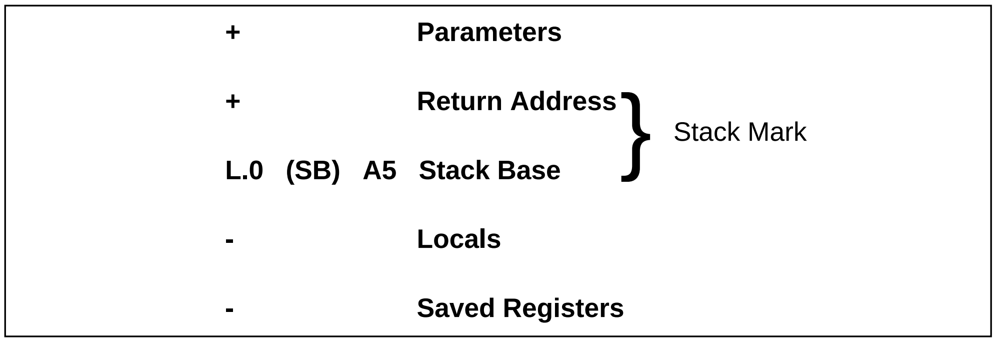
The SuperNode expression stack grows from high address to low, whereas NT40 does the opposite. Thus, offsets from the expression stack for SuperNode (for example, ‘S.2’) should be positive.

Local variables

On the NT40, local variables are obtained from compiled listings. L.0 is the first passed parameter or local. The local variables always have a positive offset.

On SuperNode, local variables are sometimes kept in registers instead of (or in addition to) being on the stack. The SuperNode stack is described in Figure 5-4 on page 5-4.

Figure 5-4xxx
Locals in the SuperNode stack



Locals are at negative offsets from the stack base, whereas passed parameters are at positive offsets. Local variable offsets given by listings may therefore be misleading. The user is advised to disassemble the code using DIsplay PS if in doubt.

The following commands show how register A5 is used to display local variables. The two commands provide the same result.

```
DI L.-4 N=20
```

```
DI R A5@.-4 N=20
```

Registers

The NT40 has base registers. Each base register is designed for a specific purpose, and most can be examined outside of a tracepoint.

On SuperNode, DEBUG registers are named D0 to D7 (data registers) and A0 to A7 (address registers). The SP (stack pointer) and SR (status register) may also be displayed. All registers may be displayed at once using DI R ALL. These registers are 32-bit hardware registers (except for the SR, which is 16 bits) which are used to hold data similar to the way the stack does on the NT40. In order to display the contents of a register in SuperNode, the user must be in a tracepoint.

When the user issues the following command with SuperNode,

```
DI R n
```

n can take on any of the following values:

- A0
- A1

• A2	• A3
• A4	• A5
• A6	• A7
• D0	• D1
• D2	• D3
• D4	• D5
• D6	• D7
• SR	• SP
• ALL	

The upper or lower word of a data or address register may be selected by suffixing the register name with U or L (for example, D5L).

Table 5-1 on page 5-5 shows register equivalencies between NT40 and SuperNode.

Table 5-1xxx
Important register equivalencies

Data block	Data name in CPUDEFUI	NT40 base register	SuperNode register	SuperNode address	Common commands
CCB	DATAREG0	R 4	R D7@	410800@	CCBPTR@
CDB	DATAREG1	R 5		410804@	CDBPTR@
EXT	DATAREG2	R 6		410808@	ECCBPTR@
PCB	BASEPCBRE G	R 40		41080C@	RUNPPTR@

Note: In SuperNode, the stack base (SB) pointer is in R A5, and the stack pointer (SP) is in R A7.

Tables

Both the NT40 and SuperNode use word strides for tables, although data store in the SuperNode is byte-addressable.

The NT40 uses word addresses and bit strides for bit tables. SuperNode uses byte addresses and bit strides, unless a bit field is specified, in which case it uses a word address and bit stride. See Figure 5-5 on page 5-6 for examples of displaying word and bit tables.

Figure 5-5xxx
Displaying tables on the NT40 and SuperNode

- Assume that CPTABUI:PR.8 is a table with a stride of 2 words. To look at the fourth through eighth entries of this table, enter the following command string:

  NT40:

  >DI CPTABUI: PR.8 T 2 (&3) N=5

  SuperNode:

  >DI CPTABUI: PR.16 T 2 (&3) N=5

- Assume that CPTABUI:PR.8 is a bit table with a bit stride of 8. To look at the fifth entry of this bit table, enter the following command string:

  NT40:

  >DI CPTABUI: PR.8 BT 8 (&4)

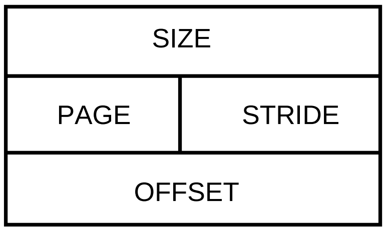
  SuperNode:

  >DI CPTABUI: PR.16 BT 8 (&4)

Descriptors

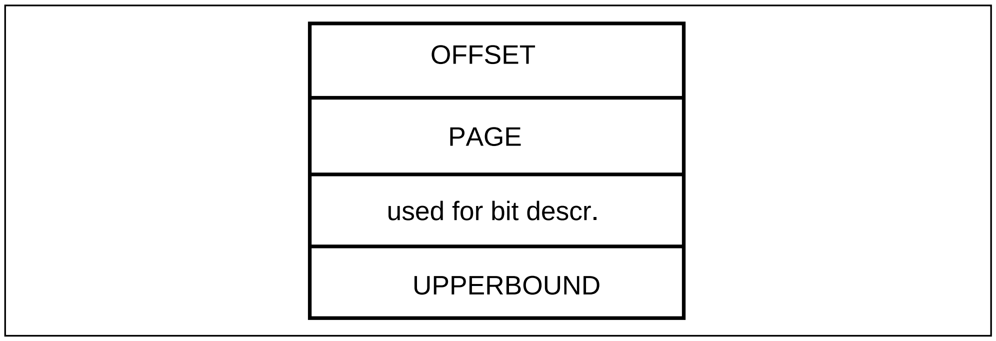
The NT40 descriptor is three words long, made up of a page, address (offset), size and stride as shown in Figure 5-6 on page 5-6.

Figure 5-6xxx
Descriptor layout on the NT40



The SuperNode descriptor is four words long, made up of a page, address (offset), size, and upperbound as shown in Figure 5-7 on page 5-7.

Figure 5-7xxx
Descriptor layout on SuperNode



For the NT40 version of DEBUG no stride has to be specified when using a descriptor, since DEBUG can read it from the data structure. This is not possible on the SuperNode version. The display command that uses descriptors has an additional parameter D followed by the stride.

Enter the following command string to display data using a word descriptor on the SuperNode:

DI storage-reference .nn D x (i)

And for a bit descriptor:

DI storage_reference .nn BD x (i)

Where:

nn	is the word offset.
D	indicates the SuperNode version is used.
BD	indicates a bit descriptor is indexed using the SuperNode version.
DB	indicates a bit descriptor is indexed using the SuperNode version.
x	is the size of the stride in decimal. The stride is a word stride for word descriptors and a bit stride for bit descriptors.
i	is the index of the desired element in the table. The index can be a constant or a storage reference (for example, (S.-2)). If the index is a constant, it is displayed as (&decimal value) or (&#hex value).

See Figure 5-8 on page 5-8 for an example of an equivalent commands entered on the NT40 and SuperNode.

Figure 5-8
Displaying descriptors on NT40 and SuperNode

- Assume that CPTABUI:SH.20 is a descriptor with a stride of 4. To look at the third entry of where this descriptor points, enter the following command string:

  NT40:

  >DI CPTABUI: SH.20 (&2)

  SuperNode:

  >DI CPTABUI: SH.20 D 4 (&2)

- Assume that CPTABUI:SH.15 is a bit descriptor with a bit stride of 4. To look at the second entry of where this descriptor points, enter the following command string:

  NT40:

  >DI CPTABUI: SH.15 B (&1)

  SuperNode:

  >DI CPTABUI: SH.15 BD 4 (&1)

  Note: Because of byte addressing on the SuperNode, offsets will be different on the SuperNode than on the NT40.

See Figure 5-9 on page 5-9 for an example of type declarations for a descriptor on the NT40 and SuperNode.

Figure 5-9
Descriptors on NT40 versus SuperNode

```
NT40:

TYPE $DESCRIPTOR PACK (DESCSIZE)
   STRUCT
      SIZE              POSINT, %% ACTUALLY GOES UP TO 65K
      OVLY {0 TO 1}
      {0}:
         STRIDE         {0 TO #FF}
      {1}:
         BITSTRIDE      {0 TO 15},
         BITOFFSET      {0 TO 15}
      ENDOVLY,
      PAGE              ADDRPAGE,
      OFFSET            ADDROFFSET
   ENDSTRUCT;

SuperNode:

TYPE $DESCRIPTOR
   STRUCT
      ADDRESS                $UNIVERSAL_PTR,
      OVLY {0 TO 2}
      {0}: UNUSED            POSINT   %L% FOR ALIGNED DESCRIPTORS.
      {1}: BITOFFSET         POSINT
      {2}: CHAROFFSET        POSINT
      ENDOVLY,
      UPPERBOUND             POSINT   %L% THIS IS MORE CONVENIENT FOR
                                      %L% DESC MANIPULATION.
  ENDSTRUCT;
```

On SuperNode the upperbound is the size minus one. Therefore an upperbound of -1 on SuperNode and a size of 0 on the NT40 are equivalent and are NIL descriptors.

Pointers and addresses

On the SuperNode, addresses and pointers are eight hex digits long (32 bits) as opposed to six on the NT40 (24 bits).

The NT40 pointer is made up of the offset (one word) and page (8 bits). The SuperNode pointer is made up of the address (one word) and the page (one word). Figure 5-10 on page 5-10 gives an example of a type declaration for a pointer on the NT40 and on SuperNode.

Figure 5-10
Pointers on NT40 versus SuperNode

```
NT40:

TYPE $POINTER PACK (PTRSIZE)
   STRUCT
      OFFSET          ADDROFFSET,
      FILL            {0 to #FF},
      PAGE            ADDRPAGE
   ENDSTRUCT;

SuperNode:

TYPE $POINTER PACK (PTRSIZE)
%       --------
   STRUCT
      OVLY   {0 to 1}
      {0}:    ADDRESS      $UNIVERSAL_PTR
      {1}:    PAGE         $INT,
              OFFSET       $INT
      ENDOVLY
   ENDSTRUCT;
```

Tracepoint procedure code offset

Since all procedures start with a 3-byte linkage instruction on the NT40, DEBUG assumes an offset of #3 bytes if none is given in the DEFine command. On SuperNode, the default is 0 bytes. This may not give the desired result, since procedure linkage has not yet been done at this point. For the majority of procedures, linkage is complete before offset #10 bytes, so the tracepoint can safely be placed at offset #10 or greater. However, it is best to display program store first to be certain. Offset #10 is usually a valid offset, but not always. The problem with the zero byte default is most apparent when setting a traceback in the tracepoint. If the user places a tracepoint before the following statement,

```
LINK.W A5, $#-xx
```

the caller of the procedure will not be printed in the traceback. This is because the LINK statement sets up the stack base for this procedure.

The best solution for this problem is to not set up a tracepoint at the default offset of a procedure. Instead, take the time to display some program store, and set the tracepoint after the 'LINK.W A5, $#-xx' statement.

EXITIFs

EXITIFs work essentially the same on the NT40 and SuperNode, except for TID_MAP comparison. TID_MAP comparison is one of the most widely used ways of saving call data on a certain Terminal Identifier (TID). In SuperNode, the user cannot compare more than one word per EXITIF. See Figure 5-11 on page 5-11 for an example of an EXITIF on the NT40 and SuperNode.

Figure 5-11
EXITIFs on NT40 versus SuperNode

```
NT40:

EXITIF R 4.14 ^= IOUI:PR.234 (&nn)(&tn)

SuperNode:

EXITIF R D7@.28 ^= IOUI:PR.505 D 4(&nn) D 2 (&tn)
EXITIF R D7@.2A ^= IOUI:PR.505 D 4(&nn) D 2 (&tn).2
```

The user can, however, EXITIF on all 32 bits of a register, as is done in the following command:

```
EXITIF R D2 = &#12345678
```

List of terms

Absolute Address

An address in a computer language that identifies a storage location or a device without the use of any intermediate reference. An address that is permanently assigned by the machine designer to a storage location. Synonymous with Explicit Address, Machine Address, and Specific Address.

Buffer

A storage device used to compensate for a difference in rate of data flow or time of occurrence of an event when transmitting data from one device to another.

Call Condense Block (CCB)

A data block that is associated with a call from initiation through completion. The CCB contains enough information to describe a basic call, and can be extended for calls that require more data.

Call Condensing

Consists of a) condensing the data stored in the Call Data Block (CDB) so that only the information necessary to handle further functions associated with the call is retained; and b) idling the process previously associated with the call and freeing the CDB.

Call Data Block (CDB)

A data block that is associated with a call only while it is being processed. The CDB has the capability of being extended if required.

Call Processing

The software system that handles the processes involved in setting up connections through the DMS-100 Family network between calling and called parties.

Call Processing Identifier (CPID)

Uniquely identifies a call processing agent. The CPID consists of two parts a) a call processing selector, which is a number that identifies a particular call processing agency b) an agent identifier that identifies a specific agent within the agency.

CC

Central Control

CCB

Call Condense Block

CDB

Call Data Block

Central Control (CC)

Comprises the data processing functions of the DMS-100 Family, with associated data store and program store.

CI

Command Interpreter

CLLI

Common Language Location Identifier

Common Language Location Identifier (CLLI)

A standard identification method for trunk groups in the form:

aaaa bb xx yyyy

Where:

aaaa	is City Code.
bb	is Province/State Code.
xx	is Trunk Group Identity.
yyyy	is Trunk Number.

Command Interpreter (CI)

A support operating system component that functions as the main interface between machine and user. Its principal roles are;

1 to read lines entered by a terminal user
2 to break each line into recognizable units
3 to analyze the units
4 to recognize command input-numbers on the input lines
5 to invoke these commands.

CPID

Call Processing Identifier

Data Store

The main data memory for the DMS-100 Family system. The data store is part of the Central Control Complex and contains transient information on a per-call basis as well as customer data and office parameters.

Directory Number (DN)

The full complement of digits required to designate a subscriber’s station within one NPA - usually a three-digit Central Office code followed by a four-digit station number.

DN

Directory Number

DS

Data Store

DISPCALL

Display Call

Display Call (DISPCALL)

A debugging tool that captures data associated with death of a call, a call being held for trouble analysis, or a call in progress. The formatted output displays the Call Condense Block (CCB) and the Call Data Block (CDB).

EXT

Extension Block

Extension Block (EXT)

In DMS-100 Call Processing, used to provide additional data space for a Call Condense Block (CCB) or a Call Data Block (CDB).

Index

1) In DMS, a piece of information by which a particular tuple in an internal schema table is identified. There is no ordering associated with an index and no concept of a used or unused index. (All indices in the index range are valid.) 2) A 15-bit field containing the two-part descriptor index of the Call Condense Block.

Line Equipment Number (LEN)

Composed of the site, frame number, unit number, drawer number, and circuit number. For example, the LEN HOST 00 0 05 08 has the site HOST, frame number 00, unit number 0, drawer number 05, and circuit number 08.

LEN

Line Equipment Number

Maintenance and Administration Position (MAP)

A group of components that provide a Man-Machine Interface between operating company personnel and the DMS-100 Family systems. A MAP consists of a Visual Display Unit and keyboard, a voice communications module, test facilities, and MAP furniture. MAP is a trademark of Northern Telecom.

MAP

Maintenance and Administration Position

Node

Any unit that can accept or originate messages.

Node Number

A system assigned number unique to a node.

PCB

Process Control Block

Peripheral Module (PM)

A generic term referring to all hardware modules of the DMS-100 Family systems that provide interfaces with external line, trunk, or service facilities. PMs contain Peripheral Processors which perform local routines, thus relieving the load on the Central Processing Unit.

Peripheral Module Intercept System Test (PMIST)

A debugging tool that traces messages between the Peripheral Modules.

PM

Peripheral Module

PMIST

Peripheral Module Intercept System Test

Procedure

In DMS, a block of PROTEL statements with a single entry and single exit.

Process Control Block (PCB)

A block of data used by the operating system to keep track of the status of a process. There is a unique PCB for each process in the system.

Program Store (PS)

In DMS, programmed instructions for the various procedures required to perform processing, administration, and maintenance.

PROT

Protected Store

Protected Store (PROT)

In DMS, data store type that must be explicitly unprotected prior to any write-operation and protected again afterwards. This type of store remains allocated over all restarts. PROT is used to hold the office data base and translation data equipment configurations.

PS

Program Store

Terminal

1) Refers to both the interface circuit on a circuit card mounted in a PM unit and the device it is connected to. Devices include telephone sets, trunk circuits, and data links. 2) An external connection to the DMS system.

Terminal Identifier (TID)

The node number and the terminal number.

Terminal Number

A number given to a specific terminal attached to a node.

TID

Terminal Identifier

Tracepoint

An interruption in the flow of a software program to collect information. A tracepoint causes a program to branch to system code to collect information and then return to the original program to continue execution.

User

A person, organization, or other group that uses the services of a DMS switch.

DMS-100 Family

DEBUG

Technical Assistance Manual

Publication number: TAM-1001-008
Product release: BCS36 and up
Document release: Standard 03.02
Date: December 1993

Printed in the United States of America

northern
telecom