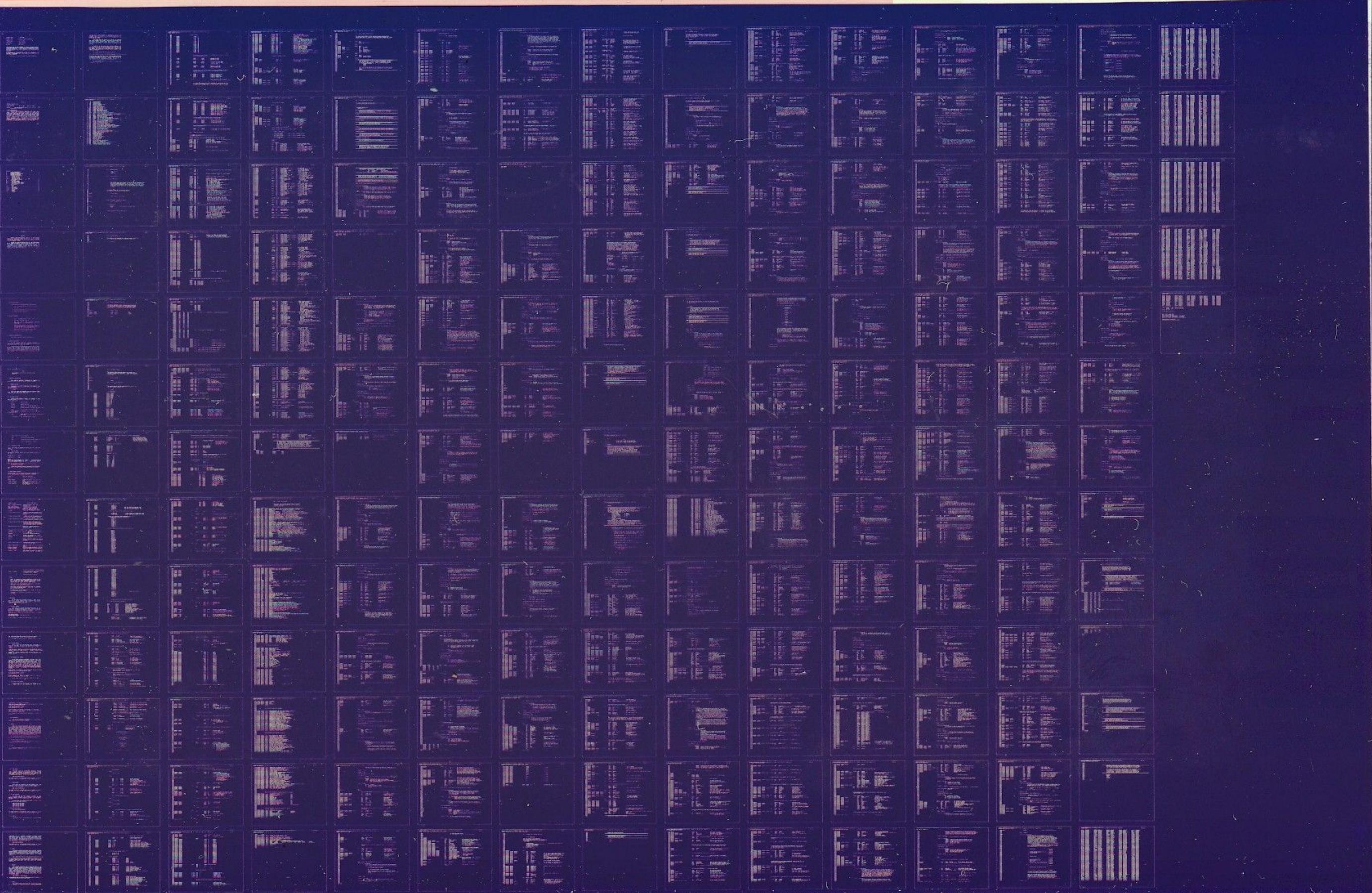


**DEUNA
DELUA**

**DEUNA NI EXER
CZUACC0**

**AH-T228C-MC
1 OF 1 OCT 1985
COPYRIGHT© 1981-85**

**digital
MADE IN USA**



e :
m
A ?:

SEQ 1

PRODUCT CODE: AC-T227C-MC
PRODUCT NAME: CZUACCO DEUNA NI EXERCISER
PRODUCT DATE: JULY 3, 1985
MAINTAINER: JAMES CRITSER
AUTHOR: GARY MCCOY

THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION. DIGITAL EQUIPMENT CORPORATION ASSUMES NO RESPONSIBILITY FOR ANY ERRORS THAT MAY APPEAR IN THIS DOCUMENT.

NO RESPONSIBILITY IS ASSUMED FOR THE USE OR RELIABILITY OF SOFTWARE ON EQUIPMENT THAT IS NOT SUPPLIED BY DIGITAL OR ITS AFFILIATED COMPANIES.

COPYRIGHT (C) 1985 BY DIGITAL EQUIPMENT CORPORATION

HISTORY

ORIGINAL RELEASE: 1981

FIRST REVISION: JULY 3, 1985 Dennis R. Racca

REASON: The NIE functional specification has been significantly enhanced.

CHANGES/ENHANCEMENTS:

The NIE listen and bounce commands, both new, were added. Nearly all routines were modified in some way to either clean them up or make them conform to the new NIE functional specification. Also, a set of routines was added that will allow the NIE to make use of extended memory made available to it by the advent of new releases of the XDP+ monitor. These routines let the NIE drive the PDP-11's memory management unit. The addition of more memory has eased limitations imposed by memory size while allowing the enlargement of NIE data structures. More available memory allows future enhancements to this version of the NIE.

CONTENTS

ABSTRACT

1	SYSTEM REQUIREMENTS	6
2	RELATED DOCUMENTS	6
3	DIAGNOSTIC PREREQUISITES	6
4	PROGRAM ASSUMPTIONS	6
5	OPERATING INSTRUCTIONS	7
5.1	LOADING THE NIE	7
5.2	NIE AND THE DRS	7
5.2.1	STARTING THE NIE	8
5.3	NIE COMMAND LANGUAGE	8
6	NIE ERRORS	10
6.1	NIE SOFT ERRORS	10
6.2	NIE HARD ERRORS	11
6.3	NIE DEVICE FATAL ERRORS	11
6.4	NIE SYSTEM FATAL ERRORS	11
7	TEST SUMMARIES	12
7.1	BUILD	12
7.2	RUN	12
7.2.1	RUN DIRECT	13
7.2.2	RUN PATTERN	13
7.2.3	RUN LOOPPAIR	13
7.2.4	RUN ALL	13
7.3	BOUNCE	14
7.4	IDENTIFY	14
7.5	LISTEN	14

ABSTRACT

CZUACC is the XXDP+ monitor version of the Network Interconnect Exerciser (NIE) written to use the Digital Ethernet LSI Unibus Adapter (DELUA) or the Digital Ethernet to Unibus adapter (DEUNA).

The NIE is a tool designed to aid in the maintenance of an Ethernet network. Its functions are twofold. First, and foremost, the NIE verifies the connectivity (or lack of) of nodes on the network by testing their ability to communicate with one another. Second, the NIE provides a network monitoring capability that allows a user to get a sampling of the traffic on the NI.

1 SYSTEM REQUIREMENTS

The NIE has the following hardware requirements:

- o PDP-11/24,34A,44,70,84 with functioning clock
- o 256K RAM
- o DELUA or DEUNA Unibus Ethernet Controller
- o H4000 Ethernet Transceiver

2 RELATED DOCUMENTS

1. PDP-11 DIAGNOSTIC DESIGN GUIDE (EL-ENDIA-11)
2. NIE Functional Specification
3. DEC STD 134-0, The Digital Ethernet Specification, A-DS-EL00134-0-0, Rev. A, 6-Mar-1984
4. DECnet Digital Network Architecture, Phase 4, Maintenance Operations Functional Specification, AA-X436A-TK, Ver. 3.0.0, December 1983
5. DEUNA User's Guide, EK-DEUNA-UG-001, 1983
6. DELUA User's Guide, EK-DELUA-UG-PRE

3 DIAGNOSTIC PREREQUISITES

There are no prerequisites for the NIE to run.

4 PROGRAM ASSUMPTIONS

The NIE assumes that all required hardware is functioning correctly, with the exception of the Ethernet controller which it will check for errors.

This version of the NIE must be run with V2.0 or later of the XXDP+ monitor. The extended memory features of the NIE make use of capabilities afforded it by using the extended XXDP+ system, labeled XXDPXM.SYS on XXDP+ system media. All processors supported by this version of the NIE come equipped with the necessary memory required by the NIE

and the extended monitor.

NOTE

THIS VERSION OF THE NIE WILL NOT WORK WITHOUT
XXDPXM.SYS

5 OPERATING INSTRUCTIONS

This section contains information on loading and starting the NIE, as well as the NIE command language.

5.1 LOADING THE NIE

You must have an XXDP+ system media that contains the file CZUACC.BIN. Boot the media and at the XXDP+ prompt, type the following:

.R CZUACC

This will cause the Diagnostic Run-Time Services (DRS) along with the NIE to be loaded into PDP-11 memory. XXDP+ will then pass control over to the DRS.

5.2 NIE AND THE DRS

Though the DRS offers a number of commands to the user, when running the NIE only a subset are relevant. These are the following:

STArt	- Start the NIE
REStart	- restart the NIE
CONTinue	- continue running the NIE after a tC
DISplay	- display contents of hardware parameter table
EXIT	- exit the DRS to the XXDP+ monitor

START, RESTART, and CONTINUE may be used with the following switches:

/NOR	- tells the DRS to not perform checksums after DRS traps
/FLA:flaglist	- sets all DRS flags in flaglist

those flags that may be used are:

- | | |
|-----|--|
| IER | - inhibit all error reports |
| IBE | - inhibit all error reports except first level |
| IXE | - inhibit extended error reports |

5.2.1 STARTING THE NIE -

After XXDP+ has passed control to the DRS, the DRS issues its prompt and waits for instructions. To start the NIE type:

DR>START/NOR

The following dialogue should take place between the DRS and the user:

Change HW (L) ? ...type Y

UNITS (D) ? ... type 1

unit 0
 WHAT IS THE PCSRO ADDRESS? (0) 174510 ? ... type PCSRO address
 WHAT IS THE VECTOR ADDRESS? (0) 120 ? ... type vector address
 WHAT IS THE PRIORITY LEVEL? (0) 5 ? ... type priority level

NOTE: for the last three questions a return will cause the default to be used.

After this dialogue control is passed to the NIE which will print an identification message and give its prompt --
 NIE>

5.3 NIE COMMAND LANGUAGE

COMMAND SUMMARY FOR THE NETWORK INTERCONNECT EXERCISER (NIE)
 (it is only necessary to type the letters in brackets)

- | | |
|-------------------|---|
| [H]elp or ? | - type this help text. |
| [E]xit | - return to the supervisor. |
| [SH]ow [N]odes | - prints information in current node table. |
| [SH]ow [M]essage | - prints selected message type, size, and copies. |
| [SH]ow [C]ounters | - prints the low level counters of the HOST NODE. |

[S]how [L]isten - print listen data

[R]un [L]oopair/[P]ass=nn - runs the looppair test, pass defaults to 1

[R]un [A]ll/[P]ass=nn - runs the node-to-node test

[R]un [D]irect/[P]ass=nn - runs the direct loop test

[B]ounce /<addr list> - allows the user to select a path for loopforwarding a packet.

[L]isten - listen for all packets on the NI.

[L]isten [P]rotocol/nnnn - listen to the NI for packets using protocol type nnnn and display those packets.

[L]isten [S]ource/<addr> - listen to the NI for packets which have the source address indicated.

[L]isten [D]estination/<addr> - listen to the NI for packets which have the destination address indicated.

[L]isten [S]ource/<addr>/[D]estination/<addr>/[P]rotocol/nnnn - listen to the NI for packets which have source and destination addresses and the protocol type as indicated.

[M]essage/[T]ype=a/[S]ize=n/[C]opies=m - allows the user to modify the default message type, size and copy count

[M]essage /[T]xt =\$<hex data string> - input user defined hex data

[M]essage /[T]xt ="<ascii data string> - input user defined ascii data

[M]essage - sets default message parameters

[NOD]es /<addr list> - enters 1 or more physical address into the node table.

[SU]mmary - prints a summary of the test results.

[B]uild - builds a table of remote node physical addresses by listening to ID messages on the NI.

[C]lear [N]ode/<addr list> - removes nodes listed in the address list from the node table.

[C]lear [N]ode/[A]ll - clears all nodes from the current node table.

[C]lear [M]essage - sets all message parameters to default.

[C]lear [L]isten - clears the accumulated listen data.

[C]lear [S]ummary - clears the table of summary test data.

[I]dentify <addr> - uses request ID function to identify a remote node on the NI. The address may

be either a physical or logical address.

- | | |
|---------------------|---|
| [SA]ve <filespec> | - writes the current node table into the file specified by filespec. |
| [U]NSAVE <filespec> | - updates the current node table from the file specified by filespec. |

Notes:

1. <addr> is a physical or logical address of a node on the NI. The physical address consists of a string of 12 hex digits which may have embedded spaces and dashes. Logical addresses range from N1 to N2000 (Octal)
2. <addr list> is a list of physical and logical addresses. Addresses must be separated by commas.
3. Pass count, optionally specified within the run command, is a positive decimal number. Specifying -1 causes the test to loop indefinitely.
4. A protocol type is described by 4 hex digits which may have embedded spaces or dashes.
5. <filespec> is a character string specifying a valid XXDP+ file name.

6 NIE ERRORS

The DRS offers four classes of errors: soft errors, hard errors, device fatal errors, and system fatal errors. (For a detailed explanation of each, refer to the PDP-11 Diagnostic Design Guide, section 7.5.7)

6.1 NIE SOFT ERRORS

Soft errors for the NIE are those errors that do not hinder the further operation of the NIE. These errors will generally be caused by the inability of nodes to communicate on the NI. An example of a soft error follows:

CZUAC soft error 00034 on unit 00 test 001 sub 000 PC: 050264

LOOP DIRECT FAILED
FAILING NODE ADDRESS: AA-00-03-01-07-42
DATA PATTERN: ASCII

In this example, an attempt was made to loop a packet with

the given data through the node with the given address. The node did not respond, so the failure was duly noted.

The NIE will always continue operation from a soft error.

6.2 NIE HARD ERRORS

There is only one error that has been classified as hard for the NIE. It occurs when the NIE has attempted to transmit a packet three times on the NI without success; it follows:

CZUAC hard error 00015 on unit 00 test 001 sub 000 PC: 032714
TRANSMIT FAILED AFTER THREE ATTEMPTS -- ETHERNET EXTREMELY LOADED

The NIE will continue from this error, but the fact that the network is very busy should be taken into consideration for further testing.

6.3 NIE DEVICE FATAL ERRORS

Device fatal errors are hardware failures that will inhibit further successful operation of the NIE. There are two pieces of hardware that will cause a device fatal error upon failure, the DEUNA or DELUA and the system clock. Since the DEUNA or DELUA is the hardware used to communicate over the NI, its failure will, of course, have drastic consequences for the NIE. The system clock is used by the NIE to time operations, such as timeouts for pending packet receptions. If it fails, the NIE quite possibly will hang-up waiting for events. An example of a device fatal error follows:

CZUAC DVC FTL error 00011 on unit 00 test 001 sub 001 PC: 032014
DEUNA/DELUA WILL NOT READ DESCRIPTOR RINGS

PC OF CALLING ROUTINE = 032324
pass aborted for this unit

In this example, the DEUNA or DELUA could not read the descriptor presented to it by the NIE.

Device fatal errors will cause a return to the DRS.

6.4 NIE SYSTEM FATAL ERRORS

A system fatal error for the NIE is an attempt by the NIE to report when it has sustained an error due to

inaccuracies in software. For example:

CZUAC SYS FTL error 00014 on unit 00 test 001 sub 000 PC: 032702
TRANSMIT RING BOOKKEEPING ERROR

PC OF CALLING ROUTINE = 32324
pass aborted for this unit

In this example, the NIE has encountered an inaccuracy in what it believes the transmit ring looks like and what the device believes it looks like.

These are very severe errors resulting in a return to the DRS.

7 TEST SUMMARIES

This section contains information on different NIE tests as well as the NIE BUILD command.

7.1 BUILD

Before any node testing can be done a table of nodes to test must be created. The BUILD command is the method by which this is done. When BUILD is issued, the NIE listens for system IDs of nodes on the NI. As nodes are heard from they are added to the node table. The node table contains a node's current physical address, its default physical address, its DECnet address (if it has one), a logical node number by which the node may be addressed, and the type of Ethernet controller at that node (e.g. DEQNA). The BUILD continues until one of the following conditions occurs:

1. 40 minutes have passed since the beginning of the BUILD
2. No node has been heard from in the past 10 minutes, or
3. the user types a control-C

The SHOW NODES command may be used to display the information contained in the node table.

7.2 RUN

RUN will invoke one of the following four tests:
DIRECT, PATTERN, LOOPPAIR, or ALL.

7.2.1 RUN DIRECT -

This test uses the Maintenance Operation Protocol (MOP) loopback protocol to loop packets from the host node (the one on which the NIE is running) to each node in the node table. This verifies the ability of the node under test to communicate on the NI. To run this test type:

NIE> RUN DIRECT/PASS=N

The /PASS qualifier indicates the number of times to invoke the test. If it is not specified it will default to one.

7.2.2 RUN PATTERN -

This test is identical to RUN DIRECT with the exception that it will loop a packet of each message type to each node in the node table. To run this test type:

NIE> RUN PATTERN/PASS=N

The /PASS qualifier indicates the number of times to invoke the test. If it is not specified it will default to one.

7.2.3 RUN LOOPPAIR -

This test uses the MOP loopback protocol to loop packets between adjacent pairs of nodes in the node table. It tests nodes' ability to communicate with other nodes on the NI.

If there were four nodes in the table -- N1-N4 -- then the series of loop tests would be:

HOST->N1->N2->N1->HOST
HOST->N2->N3->N2->HOST
HOST->N3->N4->N3->HOST
HOST->N4->N1->N4->HOST

To run this test type:

NIE> RUN LOOPPAIR/PASS=N

The /PASS qualifier indicates the number of times to invoke the test. If it is not specified it will default to one.

7.2.4 RUN ALL -

The RUN ALL test is a two part test. First the DIRECT

loop test is run. Second, a packet is looped, via MOP loopback protocol, to each pair of nodes in the node table. The second part is only run if all nodes respond in the direct loop test. The function of the test is to verify that the two nodes on the farthest ends of the NI can communicate with each other. To run this test type:

NIE> RUN ALL/PASS=N

The /PASS qualifier indicates the number of times to invoke the test. If it is not specified it will default to one.

7.3 BOUNCE

The bounce command also makes use of the MOP loopback protocol packet. It will allow the user to specify a path on which a loopback packet will travel. It allows the user the flexibility of testing explicit communications paths between nodes without the overhead of the RUN command. An example follows:

NIE> BOUNCE/NO,AA-00-04-00-08-10,N37,AA-00-04-00-27-10,N12

If this command were given then the NIE would attempt to loop a packet along the path specified. Note the mixing of logical node names (from the node table) and Ethernet addresses.

7.4 IDENTIFY

This command allows the user to identify nodes on the NI. When issued, the NIE will send a request ID to the node specified in the command line and, if the node replies to the request, displays the information contained in the node's reply. Some, but not all, of this information would be the nodes current physical address, its default physical address, the type of controller attached to that node, and the maintenance operations it is capable of performing. To use this command type:

NIE> IDENTIFY <node-address>

<node-address> may be either an Ethernet physical address or a logical node name from the node table.

7.5 LISTEN

The LISTEN command allows the user to passively listen to a sampling of traffic on the NI. For this command the

user may specify packet filters for destination, source, and protocol type. If a packet is successfully received and it passes the user specified filters, it will be added to a log maintained by the NIE.

This listen log will contain 30 entries of packets that have passed the filters. Each entry will contain the destination, source, protocol type, and character count of the packet that passed the filter, along with a count of the number of times a packet with those exact characteristics was received.

In addition to the listen log a source address list will be maintained by the NIE that contains up to 30 entries. Each entry will contain a source address from a packet that has passed the specified filters along with a count of the number of times that packets with that source address have passed the filters.

The LISTEN command has the following format:

NIE> LISTEN SOURCE/<src-adr>/DESTINATION/<dest-adr>/PROTOCOL/<prot-type>

where <src-adr> and <dest-adr> may be Ethernet node addresses or logical node names and <prot-type> is a hexadecimal string representing a protocol type (e.g. 90-00). Any or all of the filters may be included or excluded. The only way to terminate the listen command is by typing control-C.

The SHOW LISTEN command may be used to display the information in the logs.

2-	28	PROGRAM HEADER
2-	60	Program Macros
3-	450	DISPATCH TABLE
4-	466	DEFAULT HARDWARE P-TABLE
5-	494	SOFTWARE P-TABLE
5-	515	GLOBAL EQUATES SECTION
6-	983	GLOBAL DATA SECTION
6-	1686	COMMAND LINE ACTION TREE
7-	1963	GLOBAL TEXT SECTION
8-	2265	GLOBAL ERROR REPORT SECTION
9-	2312	GLOBAL SUBROUTINES SECTION
9-	2407	CLKSET Clock Setup Subroutine
10-	2451	CLKINT Clock Interrupt Service Routine
10-	2514	PREG14 Preserve Registers 1 through 4 across subroutine calls
11-	2575	WAIT Wait For DEUNA/DELUA Interrupt with Timeout
11-	2621	ERROR Handle UNA interrupt errors
11-	2786	UNAINI Initialize the UNA
11-	2890	unaistr una interrupt service routine
11-	2984	COMMAND Subr to issue a DELUA/DEUNA port command
11-	3022	FUNCT subr to perform a DELUA/DEUNA Port Function
11-	3068	XMIT Transmit DELUA/DEUNA frames
11-	3175	RECEIVE Receive DELUA/DEUNA ring buffers
11-	3318	EDPACK ETHERNET DATA PACKING ROUTINE
11-	3378	HXFORM HEX FORMAT ROUTINE
11-	3444	HEXBIN HEX TO BINARY CONVERSION
11-	3518	BINHEX Binary to Hex Conversion Procedure
11-	3572	BLDLD Build loop direct data buffers for transmit.
12-	3644	BLDFAS Build frame for full assist transmission.
13-	3762	BLDREQ Build Request ID Frames for transmit.
13-	3819	GET?NX Get next transmit or receive ring entry
13-	3862	BLDBUF Build Message Buffers
14-	3946	DATCMP Compare data buffers
14-	4015	WRITES Write data onto summary table
15-	4113	BINDEC Convert a 32 bit binary number to decimal
16-	4194	COMMAND LINE TRAVERSE ROUTINES
16-	4525	REPORT CODING SECTION
17-	4566	PROTECTION TABLE
18-	4595	INITIALIZE SECTION
19-	4896	AUTODROP SECTION
20-	4916	CLEANUP CODING SECTION
21-	5015	DROP UNIT SECTION
22-	5051	ADD UNIT SECTION
22-	5089	TEST 1: NIE
22-	5208	CLI ACTION TABLE AND ROUTINES
23-	7430	READ LINE OF OPENED FILE
23-	8032	GETIDA get the address of a system id field
23-	8088	PRTTYP print the device type
24-	8950	HARDWARE PARAMETER CODING SECTION
25-	8992	SOFTWARE PARAMETER CODING SECTION

```
28          .SBTTL PROGRAM HEADER
54
55          :      .ENABL ABS,AMA
56          :      .= 2000
57          :      .ENABL AMA
58
59
60          .SBTTL Program Macros
61
62
63          :I$STACK macro
64          :-----
65
66          :***+
67          ;The I$STACK macro facilitates initializing the R6 (hardware) stack
68          ;and the R5 (parameter) stack. R5 is set to the stack low limit
69          ;(STAKLO) and the parameter stack grows upward. R6 is set to the
70          ;stack high limit (STAKHI) and the hardware stack grows downward.
71          ;If there is a stack over-run, it will be detected by the PREG14
72          ;routine.
73          :---
74
366          :++
367          ; THE PROGRAM HEADER IS THE INTERFACE BETWEEN
368          ; THE DIAGNOSTIC PROGRAM AND THE SUPERVISOR.
369          :--
370
371 000000          POINTER BGNRPT
372
389
390 000000          HEADER CZUAC,C,0,0,1,PRI07
391
402
403
404          ; NAMES OF DEVICES SUPPORTED BY PROGRAM
405
406 000122          :DEVTYP <DEUNA,DELUA>
407
413
414          ; TEST DESCRIPTION
415
416 000136          :DESCRIPT      <CZUAC DEUNA,DELUA NI EXERCISER>
417          .EVEN
418
425
426
427          ; FORMAT STATEMENTS USED IN PRINT CALLS
428
429
440
441
```

450 .SBTTL DISPATCH TABLE
451
452 :++
453 ; THE DISPATCH TABLE CONTAINS THE STARTING ADDRESS OF EACH TEST.
454 ; IT IS USED BY THE SUPERVISOR TO DISPATCH TO EACH TEST.
455 :--
456
457 000176
458 DISPATCH 1

```
466          .SBTTL DEFAULT HARDWARE P-TABLE
467
468          ;+
469          : THE DEFAULT HARDWARE P-TABLE CONTAINS DEFAULT VALUES OF
470          : THE TEST-DEVICE PARAMETERS. THE STRUCTURE OF THIS TABLE
471          : IS IDENTICAL TO THE STRUCTURE OF THE HARDWARE P-TABLES,
472          : AND IS USED AS A "TEMPLATE" FOR BUILDING THE P-TABLES.
473          ;-
474
475 000202          BGNHW  DFPTBL
476
477 000204 174510          .WORD  174510          ; CSR
478 000206 000120          .WORD  120           ; VECTOR
479 000210 000240          .WORD  PRI05         ; PRIORITY
480
490
491 000212          ENDHW
```

```
493
494          .SBTTL SOFTWARE P-TABLE
495
496          ;++
497          ; THE SOFTWARE TABLE CONTAINS VARIOUS DATA USED BY THE
498          ; PROGRAM AS OPERATIONAL PARAMETERS. THESE PARAMETERS ARE
499          ; SET UP AT ASSEMBLY TIME AND MAY BE VARIED BY THE OPERATOR
500          ; AT RUN TIME.
501          ;-
502
503 000212          BGNSW SFPTBL
504
512
513 000214          ENDSW
514
515          .SBTTL GLOBAL EQUATES SECTION
516
526
527
528          ;++
529          ; THE GLOBAL EQUATES SECTION CONTAINS PROGRAM EQUATES THAT
530          ; ARE USED IN MORE THAN ONE TEST.;-
531
546
547 000214          EQUALS
; BIT DEFINITIONS
;
100000          BIT15== 100000
040000          BIT14== 40000
020000          BIT13== 20000
010000          BIT12== 10000
004000          BIT11== 4000
002000          BIT10== 2000
001000          BIT09== 1000
000400          BIT08== 400
000200          BIT07== 200
000100          BIT06== 100
000040          BIT05== 40
000020          BIT04== 20
000010          BIT03== 10
000004          BIT02== 4
000002          BIT01== 2
000001          BIT00== 1
;
001000          BIT9== BIT09
000400          BIT8== BIT08
000200          BIT7== BIT07
000100          BIT6== BIT06
000040          BIT5== BIT05
000020          BIT4== BIT04
000010          BIT3== BIT03
000004          BIT2== BIT02
000002          BIT1== BIT01
000001          BIT0== BIT00
;
; EVENT FLAG DEFINITIONS
```

```
: EF32:EF17 RESERVED FOR SUPERVISOR TO PROGRAM COMMUNICATION
:
000040    EF.START==      32.          : START COMMAND WAS ISSUED
000037    EF.RESTART==   31.          : RESTART COMMAND WAS ISSUED
000036    EF.CONTINUE==  30.          : CONTINUE COMMAND WAS ISSUED
000035    EF.NEW==       29.          : A NEW PASS HAS BEEN STARTED
000034    EF.PWR==       28.          : A POWER-FAIL/POWER-UP OCCURRED
:
:
: PRIORITY LEVEL DEFINITIONS
:
000340    PRI07== 340
000300    PRI06== 300
000240    PRI05== 240
000200    PRI04== 200
000140    PRI03== 140
000100    PRI02== 100
000040    PRI01== 40
000000    PRI00== 0
:
:OPERATOR FLAG BITS
:
000004    EVL==        4
000010    LOT==        10
000020    ADR==        20
000040    IDU==        40
000100    ISR==       100
000200    UAM==       200
000400    BOE==       400
001000    PNT==      1000
002000    PRI==      2000
004000    IXE==      4000
010000    IBE==     10000
020000    IER==     20000
040000    LOE==     40000
100000    HOE==    100000
```

549
550 :::EQUATES FOR FLAG WORD:::::
551
552 000000 CTARGT==0
553 000001 CASIST==1
554 000002 CSHCTR==2 ;ARG TYPE FOR 'SHOW COUNTERS' CMD
555 000004 CCLNAD==4 ;ARG TYPE FOR 'CLEAR NODE/ADR' CMD
556 000010 CCLNAL==8. ;ARG TYPE FOR 'CLEAR NODE/ALL' CMD
557 000020 CEXIT==16.
558
559 :::CLOCK ENABLE VALUES TO BE LOADED IN CLK'S CSR:::
560
561 000100 LCLKEN==100 ; L-Clock CSR value to enable the clock
562 000111 PCLKEN==111 ; P-Clock CSR value to enable the clock
563 001600 PCLKCT==1600 ; P-Clock count set register for counter
564
565 : SPECIAL CLI CODES FOR "CHAR" ARGUMENT IN CLI CALLS
566 : (COMMAND LINE INTERPRETER DEFINITIONS)
567 000000 CLIERR= 0
568 000001 CLIEXI= 1
569 000002 CLIBR = 2
570 000003 CLIBIF= 3
571 000004 CLISPA= 4
572 000005 CLINUM= 5
573 000006 CLIALP= 6
574 000010 CLI OCT= 8.
575 000011 CLIDEC= 9.
576 000012 CLISTR= 10.
577
578 :DEFS FOR COMMAND LINE INTERPRETATION ACTION VALUES
579
580 000000 NULL=0
581 000001 HELP=1
582 000002 NODE=2
583 000003 BUILD=3
584 000004 CRUN=4
585 000005 CPATRN=5
586 000006 CSAVE=6
587 000007 SUMMRY=7
588 000010 IDENT=10
589 000011 EXIT=11
590 000012 NOTNUF=12
591 000013 CEXADR=13
592 000014 CSAVR4=14
593 000015 CNODE=15
594 000016 CALPHA=16
595 000017 CONES=17
596 000020 CZERO\$=20
597 000021 C1ALT=21
598 000022 COALT=22
599 000023 CCCITT=23
600 000024 COPRSL=24
601 000025 CTYPE=25
602 000026 CSIZE=26
603 000027 CCPY\$=27
604 000030 CNDADR=30
605 000031 CNODAL=31

```

606      000032      CRNALL=32
607      000033      CLUPPR=33
608      000034      CSHMSG=34
609      000035      CCLMSG=35
610      000036      CCNTR=36
611      000037      CNDLOG=37
612      000040      CFUNCT=40
613      000041      CUNSAV=41
614      000042      CCLSUM=42
615      000043      CDIR=43
616      000044      CDEFLT=44
617      000045      CUNSVF=45
618      000046      SETQIK=46
619      000047      CLRQIK=47
620      000050      NCMPAR=50
621      000051      INIBNC=51
622      000052      BOUNCE=52
623      000053      BNCLOG=53
624      000054      SOUADR=54
625      000055      DESADR=55
626      000056      CEXPRO=56
627      000057      LISTEN=57
628      000060      CSLIST=60
629      000061      CCLIST=61
630
631      000000      ALPHA==0          ;MESSAGE TYPE VALUES
632      000001      ONES==1
633      000002      ZEROS==2
634      000003      ONEALT==3
635      000004      ZROALT==4
636      000005      CCITT==5
637      000006      OPRSEL==6
638
639      ;           GLOBAL EQUATES FOR THE DEUNA/DELUA DRIVER
640      ;           :Port Control and Status Register 0
641
642      ;           SERI      ==      BIT15      : STATUS ERROR INTERRUPT
643      ;           PCEI      ==      BIT14      : PORT COMMAND ERROR INTERRUPT
644      ;           RXI       ==      BIT13      : RECEIVE RING INTERRUPT
645      100000      RXI       ==      BIT13      : RECEIVE RING INTERRUPT
646      040000      TXI       ==      BIT12      : TRANSMIT RING INTERRUPT
647      020000      TXI       ==      BIT12      : TRANSMIT RING INTERRUPT
648      010000      DNI       ==      BIT11      : DONE INTERRUPT
649      004000      RCBI      ==      BIT10      : RECEIVE BUFFER UNAVAILABLE
650      002000      USCI      ==      BIT08      : UNSOLICITED STATE CHANGE INTERRUPT
651      000400      FATI       ==      BIT08      : FATAL ERROR INTERERUPT
652      000400      INTR       ==      BIT07      : INTERRUPT SUMMARY <15:08>
653      000200      INTE       ==      BIT06      : INTERRUPT ENABLE
654      000100      RSET       ==      BIT05      : DEUNA/DELUA RESET
655      000040
656
657      ;           PORT COMMANDS in bit 3 to bit 0
658      ;   -----
659
660      000001      GETPCB == bit00      ; Get Address of Port Control Block
661      000002      GETFNT == bit01      ; Get Command in Port Control Block
662      000003      PNOP == bit00!bit01  ; No operation performed

```

663 000004 STRT == bit02 : Enable XMIT and RCVR
664 000005 BCOT == bit02!bit00 : Boot , -> Prim load state,
665 : initiate downline load
666
667 000010 PDMD == bit03 : polling demand/wake up bit
668 000011 TMRO == bit03!bit00 : sanity timer enable (=1 its on)
669 000012 TMRF == bit03!bit01 : Sanity Timer Off
670 000015 RSTT == bit03!bit02!bit00 : reset sanity timer
671 000017 STOP == bit03!bit02!bit01!bit00 ; Suspend DEUNA/DELUA operation
672
673
674
675 :Port Control and Status Register 1
676
677
678 100000 XPWR == bit15 : transceiver power ok
679 040000 ICAB == bit14 : port to link cable ok
680
681 000200 ; self test error code in bit 13 to bit 08
682 PCTO == bit07 : port command timeout
683
684 000010 RMTC == bit03 : remote console reserved (=1)
685
686 : port state in bit 2 to bit 0
687
688 000000 RESET == 0 : 000 reset state
689 000001 PRIMLD== bit00 : 001 primary load state
690 000002 READY== bit01 : 010 ready state
691 000003 RUN == bit01!bit00 : 011 running state
692
693 000005 UNIHLT == bit02!bit00 : 101 unibus halted state
694 000006 NIHLT == bit02!bit01 : 110 ni halted state
695 000007 NIUNI == bit02!bit01!bit00 : 111 ni and unibus halted state
696
697
698
699 :Port Control and Status Register 2
700
701 : lower 16 address bits of the port control block base
702 : address pointer in bit 15 to bit 0
703
704 :Port Control and Status Register 3
705
706 : upper 2 address bits of the port control block base
707 : address pointer in bit 1 to bit 0
708
709 :Port Functions
710
711 : function codes are as follows
712
713 000000 PFNOP == 0 : no operation performed
714 000002 RDDEFA == bit01 : read default physical address
715
716 000004 RDPHYA == bit02 : read physical address
717 000005 WDPHYA == bit02!bit00 : write physical address
718
719 000006 RDMULA == bit02!bit01 : read list of multicast addresses

720 000007 WDMULA == bit02!bit01!bit00 ; write list of multicast addresses
721 000010 RDRNGS == bit03 ; read both the rcvr and xmit rings
722 000011 WDRNGS == bit03!bit00 ; write both the rcvr and xmit rings
723 000012 RDCNTS == bit03!bit01 ; read counters
724 000013 CLRCNTS == bit03!bit01!bit00 ; read and clear counters
725 000014 RD MODE == bit03!bit02 ; read internal link mode register
726 000015 WDMODE == bit03!bit02!bit00 ; write internal link mode register
727 000016 RD STA == bit03!bit02!bit01 ; read port status
728 000017 CLRSTA == bit03!bit02!bit01!bit00 ; read and clear port status
729 000018 DMPMEM == bit04 ; dump internal memory
730 000019 LDMEM == bit04!bit00 ; load internal memory
731 000020 RD SYS == bit04!bit01 ; read system id parameters
732 000021 WDSYS == bit04!bit01!bit00 ; write system id parameters
733 000022
734 000023
735 000024
736 000025
737 000026
738 000027
739 000028
740 000029
741 000030
742 000031
743 000032
744 000033
745 000034
746 000035 header == 14. ; offset (size) to end of header in bytes
747 000036 destin == 0 ; destination address
748 000037 source == 6 ; source address
749 000038 protoT == 12. ; protocol type field
750 000039
751 000040
752 000041
753 000042
754 000043
755 000044
756 000045
757 000046
758 000047
759 000048
760 000049
761 000050
762 000051
763 000052
764 000053
765 000054
766 000055
767 000056
768 000057
769 000058
770 000059
771 000060
772 000061
773 000062
774 000063
775 000064
776 000065
777 000066
778 000067
779 000068
780 000069
781 000070
782 000071
783 000072
784 000073
785 000074
786 000075
787 000076
788 000077
789 000078
790 000079
791 000080
792 000081
793 000082
794 000083
795 000084
796 000085
797 000086
798 000087
799 000088
800 000089
801 000090
802 000091
803 000092
804 000093
805 000094
806 000095
807 000096
808 000097
809 000098
810 000099
811 000100
812 000101
813 000102
814 000103
815 000104
816 000105
817 000106
818 000107
819 000108
820 000109
821 000110
822 000111
823 000112
824 000113
825 000114
826 000115
827 000116
828 000117
829 000118
830 000119
831 000120
832 000121
833 000122
834 000123
835 000124
836 000125
837 000126
838 000127
839 000128
840 000129
841 000130
842 000131
843 000132
844 000133
845 000134
846 000135
847 000136
848 000137
849 000138
850 000139
851 000140
852 000141
853 000142
854 000143
855 000144
856 000145
857 000146
858 000147
859 000148
860 000149
861 000150
862 000151
863 000152
864 000153
865 000154
866 000155
867 000156
868 000157
869 000158
870 000159
871 000160
872 000161
873 000162
874 000163
875 000164
876 000165
877 000166
878 000167
879 000168
880 000169
881 000170
882 000171
883 000172
884 000173
885 000174
886 000175
887 000176
888 000177
889 000178
890 000179
891 000180
892 000181
893 000182
894 000183
895 000184
896 000185
897 000186
898 000187
899 000188
900 000189
901 000190
902 000191
903 000192
904 000193
905 000194
906 000195
907 000196
908 000197
909 000198
910 000199
911 000200
912 000201
913 000202
914 000203
915 000204
916 000205
917 000206
918 000207
919 000208
920 000209
921 000210
922 000211
923 000212
924 000213
925 000214
926 000215
927 000216
928 000217
929 000218
930 000219
931 000220
932 000221
933 000222
934 000223
935 000224
936 000225
937 000226
938 000227
939 000228
940 000229
941 000230
942 000231
943 000232
944 000233
945 000234
946 000235
947 000236
948 000237
949 000238
950 000239
951 000240
952 000241
953 000242
954 000243
955 000244
956 000245
957 000246
958 000247
959 000248
960 000249
961 000250
962 000251
963 000252
964 000253
965 000254
966 000255
967 000256
968 000257
969 000258
970 000259
971 000260
972 000261
973 000262
974 000263
975 000264
976 000265
977 000266
978 000267
979 000268
980 000269
981 000270
982 000271
983 000272
984 000273
985 000274
986 000275
987 000276
988 000277
989 000278
990 000279
991 000280
992 000281
993 000282
994 000283
995 000284
996 000285
997 000286
998 000287
999 000288
1000 000289
1001 000290
1002 000291
1003 000292
1004 000293
1005 000294
1006 000295
1007 000296
1008 000297
1009 000298
1010 000299
1011 000300
1012 000301
1013 000302
1014 000303
1015 000304
1016 000305
1017 000306
1018 000307
1019 000308
1020 000309
1021 000310
1022 000311
1023 000312
1024 000313
1025 000314
1026 000315
1027 000316
1028 000317
1029 000318
1030 000319
1031 000320
1032 000321
1033 000322
1034 000323
1035 000324
1036 000325
1037 000326
1038 000327
1039 000328
1040 000329
1041 000330
1042 000331
1043 000332
1044 000333
1045 000334
1046 000335
1047 000336
1048 000337
1049 000338
1050 000339
1051 000340
1052 000341
1053 000342
1054 000343
1055 000344
1056 000345
1057 000346
1058 000347
1059 000348
1060 000349
1061 000350
1062 000351
1063 000352
1064 000353
1065 000354
1066 000355
1067 000356
1068 000357
1069 000358
1070 000359
1071 000360
1072 000361
1073 000362
1074 000363
1075 000364
1076 000365
1077 000366
1078 000367
1079 000368
1080 000369
1081 000370
1082 000371
1083 000372
1084 000373
1085 000374
1086 000375
1087 000376
1088 000377
1089 000378
1090 000379
1091 000380
1092 000381
1093 000382
1094 000383
1095 000384
1096 000385
1097 000386
1098 000387
1099 000388
1100 000389
1101 000390
1102 000391
1103 000392
1104 000393
1105 000394
1106 000395
1107 000396
1108 000397
1109 000398
1110 000399
1111 000400
1112 000401
1113 000402
1114 000403
1115 000404
1116 000405
1117 000406
1118 000407
1119 000408
1120 000409
1121 000410
1122 000411
1123 000412
1124 000413
1125 000414
1126 000415
1127 000416
1128 000417
1129 000418
1130 000419
1131 000420
1132 000421
1133 000422
1134 000423
1135 000424
1136 000425
1137 000426
1138 000427
1139 000428
1140 000429
1141 000430
1142 000431
1143 000432
1144 000433
1145 000434
1146 000435
1147 000436
1148 000437
1149 000438
1150 000439
1151 000440
1152 000441
1153 000442
1154 000443
1155 000444
1156 000445
1157 000446
1158 000447
1159 000448
1160 000449
1161 000450
1162 000451
1163 000452
1164 000453
1165 000454
1166 000455
1167 000456
1168 000457
1169 000458
1170 000459
1171 000460
1172 000461
1173 000462
1174 000463
1175 000464
1176 000465
1177 000466
1178 000467
1179 000468
1180 000469
1181 000470
1182 000471
1183 000472
1184 000473
1185 000474
1186 000475
1187 000476
1188 000477
1189 000478
1190 000479
1191 000480
1192 000481
1193 000482
1194 000483
1195 000484
1196 000485
1197 000486
1198 000487
1199 000488
1200 000489
1201 000490
1202 000491
1203 000492
1204 000493
1205 000494
1206 000495
1207 000496
1208 000497
1209 000498
1210 000499
1211 000500
1212 000501
1213 000502
1214 000503
1215 000504
1216 000505
1217 000506
1218 000507
1219 000508
1220 000509
1221 000510
1222 000511
1223 000512
1224 000513
1225 000514
1226 000515
1227 000516
1228 000517
1229 000518
1230 000519
1231 000520
1232 000521
1233 000522
1234 000523
1235 000524
1236 000525
1237 000526
1238 000527
1239 000528
1240 000529
1241 000530
1242 000531
1243 000532
1244 000533
1245 000534
1246 000535
1247 000536
1248 000537
1249 000538
1250 000539
1251 000540
1252 000541
1253 000542
1254 000543
1255 000544
1256 000545
1257 000546
1258 000547
1259 000548
1260 000549
1261 000550
1262 000551
1263 000552
1264 000553
1265 000554
1266 000555
1267 000556
1268 000557
1269 000558
1270 000559
1271 000560
1272 000561
1273 000562
1274 000563
1275 000564
1276 000565
1277 000566
1278 000567
1279 000568
1280 000569
1281 000570
1282 000571
1283 000572
1284 000573
1285 000574
1286 000575
1287 000576
1288 000577
1289 000578
1290 000579
1291 000580
1292 000581
1293 000582
1294 000583
1295 000584
1296 000585
1297 000586
1298 000587
1299 000588
1300 000589
1301 000590
1302 000591
1303 000592
1304 000593
13

```
777          ; nothing needed
778
779          ; TDRB+2
780
781          ; nothing needed
782
783
784          ; TDRB+4
785
786          ;
787
788      000400    enp    ==    bit08    ; end of frame flag
789      001000    stp    ==    bit09    ; stop of frame flag
790      002000    def    ==    bit10    ; deffering frame flag
791      004000    one    ==    bit11    ; xmit successful after one retry
792      010000    more   ==    bit12    ; xmit successful after more than
793                  ; one retry
794      040000    errs   ==    bit14    ; ERROR SUMMARY BIT
795      100000    own    ==    bit15    ; ownership bit (=1 DEUNA/DELUA, =0 host)
796
797          ; TDRB+6
798
799      002000    rtry   ==    bit10    ; retry error bit
800      004000    lcar   ==    bit11    ; lost carrier error bit
801      010000    lcol   ==    bit12    ; late collision error bit
802
803      040000    ubto   ==    bit14    ; unibus timeout error bit
804      100000    bufl   ==    bit15    ; buffer length error bit
805
806          ;+
807          ; Rcvr ring descriptor definitions
808          ;-
809
810          ; RDRB+0
811
812          ; nothing needed
813
814          ; RDRB+2
815
816          ; nothing needed
817
818
819          ; RDRB+4
820
821          ;
822          ; --> indicates same as for transmit ring descriptor base
823
824      004000    crc    ==    bit11    ; crc error in received frame
825      010000    oflo   ==    bit12    ; message overflow
826      020000    fram   ==    bit13    ; framing error
827
828          ;errs   ==    bit14    ; ERROR SUMMARY BIT
829          ;own   ==    bit15    ; ownership bit (=1 DEUNA/DELUA, =0 host)
830
831          ; RDRB+6
832
833      020000    nchn   ==    bit13    ; set to indicate DEUNA/DELUA in no
```

834 ; buffer chain on rcvr mode
835 ;
836 ;ubto == bit14 : unibus timeout error bit
837 ;bufl == bit15 : buffer length error bit
838 ;
839 002756 xpklen == 1518. : transmit frame length
840 002756 rpklen == 1518. : receive frame length
841 000004 no.ntr == 4 : number of entries in xmit rings
842 000010 no.nrr == 8. : number of entries in receive rings
843 000016 LBCOU == 16 : offset to byte count for this frame type
844 000020 LISCOU == 20 : offset to count for listen log entry
845 000022 LISENT == 22 : length of one entry in listen log
846 000006 ADRCOU == 6 : offset to count for address list entry
847 000010 ADRENT == 10 : length of one entry in address list
848 ;
849 ;
850 ; System ID reply message offsets
851 ;
852 000022 sircpt == 22
853 000024 siffid == 24
854 000016 siccou == 16
855 ;
856 ; Device type defs
857 ;
858 000001 IDTUNA == 1 : DEUNA
859 000003 IDTCNA == 3 : DECNA
860 000005 IDTQNA == 5 : DEQNA
861 000011 IDTLUA == 11 : DELUA
862 000013 IDTCSA == 13 : DECSA - PLUTO
863 000021 IDTSRV == 21 : DSRVA - POSEIDON
864 ;
865 ; Loop Direct Offsets
866 ;
867 000016 ldskip == 16 : offset to skip count
868 000020 ldfct1 == 20 : offset to forward function code
869 000022 ldedr1 == 22 : offset to forward address
870 000030 ldfct2 == 30 : offset to reply function code
871 000032 ldedr2 == 32 : offset to reply address
872 300022 ldata == 22 : number of bytes of data buffer occupied by
873 ; loop header
874 ;
875 ;
876 ; Full Assist Offsets
877 ;
878 000016 faskip == 16 : offset to skip count
879 000020fafct1 == 20 : offset to first forward function code
880 000022faaddr1 == 22 : offset to first forward address
881 000030fafct2 == 30 : offset to second forward function code
882 000032faaddr2 == 32 : offset to second forward address
883 000040fafct3 == 40 : offset to third forward function code
884 000042faaddr3 == 42 : offset to third forward address
885 000050fafct4 == 50 : offset to reply function code
886 000052faaddr4 == 52 : offset to reply address
887 000032fdata1 == 32 : length of loopback header
888 000042fdata2 == 42 : length of loopback header for full assist
889 ;
890 ;

```
891          : Counter Offsets
892          :
893          000002      c.secs == 2
894          000004      c.prec == 4
895          000010      c.mrec == 10
896          000014      c.rerb == 14
897          000016      c.rerr == 16
898          000020      c.rdat == 20
899          000024      c.rmdb == 24
900          000030      c.rlin == 30
901          000032      c.rlex == 32
902          000034      c.pxmt == 34
903          000040      c.mxmt == 40
904          000044      c.pxm3 == 44
905          000050      c.pxm2 == 50
906          000054      c.pxmd == 54
907          000060      c.xdat == 60
908          000064      c.xmdb == 64
909          000066      c.xabb == 66
910          000070      c.xabt == 70
911          000074      c.coll == 74
912
913          ;---+
914          ; The following equates are for use with the memory management hardware
915          ; and its associated routines
916          ;---+
917          172350      KPAR4 == 172350      : address of KPAR4
918          172352      KPAR5 == 172352      : address of KPAR5
919          172354      KPAR6 == 172354      : address of KPAR6
920
921          001000      NKPAR4 == 001000      : original value for KPAR4
922          001200      NKPAR5 == 001200      : original value for KPAR5
923          002400      TKPAR6 == 002400      : value for KPAR6 to do write rings
924          :           : function only
925
926          177572      MMCSR0 == 177572      : address of MMU CSR0
927          000001      MMUENA == 000001      : mask to enable MMU
928          000000      MMUDIS == 000000      : mask to disable MMU
929
930          ;---+
931          ; The following values will be used as new values for KPAR4 and KPAR5
932          ; registers, which, will then point to the page that contains the
933          ; indicated structures
934          ;---+
935          002000      ORRING == 2000      : offset to receive ring
936          002400      OTRING == 2400      : offset to transmit ring
937          002600      ONTAB == 2600      : offset to node table
938          003000      OSTAB == 3000      : offset to summary table
939          003400      OLLOG == 3400      : offset to listen log
940
941          000000      BA == 0          : base address for call to BUFREQ
942          000001      EA == 1          : extended bits(18:16) for call to BUFREQ
943
944
945          ;---+
946          ; The following equates are virtual addresses of data structures that
947          ; are mapped into extended memory. Since KPAR4 and KPAR5 are the only
948          ; two page address registers that are being used to remap to extended
```

```
948          :      memory, the virtual addresses of the data structures will be in the
949          :      range 100000(0) - 137776(0).
950          :---+
951      100000      NODTBL  ==  100000      ; address of node table
952      110000      NODEND  ==  110000      ; address of end of node table
953      110000      DEFTBL  ==  110000      ; address of default address table
954      120000      DEFEND  ==  120000      ; address of end of default table
955      010000      DEFNOD  ==  010000      ; distance between node and default addr.
956      100000      STATBL  ==  100000      ; address of summary table
957      126000      STAEND  ==  126000      ; address of end of summary table
958      100000      LISLOG  ==  100000      ; address of listen log
959      101034      LISEND  ==  101034      ; address of end of listen log
960      101034      ADRLIS  ==  101034      ; address of listen address list
961      101414      ADREND  ==  101414      ; address of end of listen address list
962      100000      RRING   ==  100000      ; address of receive ring
963      100000      XRING   ==  100000      ; address of transmit ring
964
965          :---+
966          :      The next equates are the actual 18-bit physical addresses of the
967          :      the first transmit and receive buffers, respectively
968          :---+
969      040050      X11501 ==  040050      ; address bits <17:01> ...
970      000001      X11716 ==  000001      ; ... of first transmit buffer
971      000120      R11501 ==  000120      ; address bits <17:01> ...
972      000001      R11716 ==  000001      ; ... of first receive buffer
973
974          :---+
975          :      And now the virtual addresses of the first transmit and receive
976          :      buffers, respectively.
977          :---+
978
979      100050      XBUFV1 ==  100050      ; virtual addr. of first transmit buffer
980      100120      RBUFV1 ==  100120      ; virtual addr. of first receive buffer
981
982          .SBttl GLOBAL DATA SECTION
983
984
985          :++
986          ; THE GLOBAL DATA SECTION CONTAINS DATA THAT ARE USED
987          ; IN MORE THAN ONE TEST.
988
989          ;COMMAND LINE BUFFER, DATA LOCATIONS AND MESSAGES FOR ACTION ROUTINES
990
991 000214 000000      STACKS: .BLKW  100.      ; PARAMETER STACK -- USED TO PASS PROCEDURE ARGS
992 000524 000000      DEVICE: .WORD  0      ;DEFAULT TO DEUNA
993 000526          FILLIN: .BLKB  132.      ;BUFFER FOR SINGLE LINE READ FROM FILE
994 000732          CMDBUF: .BLKB  72.      ;BUFFER FOR OPERATOR COMMANDS
995 001042          CBOBUF: .BLKB  17.      ;BUFFER TO HOLD INPUT ASCII ADDRESS/PROTOCOL TYPE STRING
996
997 001064 000000      .EVEN
998 001066 000000      KEYWD1: .WORD  0      ;
999 001070 000000      KEYWD2: .WORD  0      ;
1000 001072 000000     ADRBUF: .WORD  0      ;BUFFER FOR NODE ADDRESS
1001 001074 000000     .WORD  0
1002 001076 000000     .WORD  0
1003 001076 000000     SOUFIL: .WORD  0      ;BUFFER FOR SOURCE FILTER FOR LISTEN COMMAND
1004 001100 000000     .WORD  0
```

1005 001102	000000		.WORD	0	
1006 001104		DESFLIL::	.WORD	0	:BUFFER FOR DESTINATION FILTER FOR LISTEN COMMAND
1007 001104	000000		.WORD	0	
1008 001106	000000		.WORD	0	
1009 001110	000000		.WORD	0	
1010 001112		PROFLIL::	.WORD	0	:BUFFER FOR PROTOCOL FILTER FOR LISTEN COMMAND
1011 001112	000000		.WORD	0	
1012 001114	000000		.WORD	0	
1013					
1014 001116		STRBUF:	.BLKB	18.	:BUFFER FOR ALPHANUM. ADDRESS STRING
1015 001140		STRBU1:	.BLKB	18.	
1016 001162	000000	LOGVAL:	.WORD	0	:LOGICAL NODE VALUE
1017 001164	000000	TYPADR:	.WORD	0	:ADDR. OF LOC. OF ASCII STRING THAT DESCRIBES NODE TYPE
1018 001166	000000	CBOADR:	.WORD	0	:POINTER FOR BEGINING OF ADDRESS STRING
1019 001170	000000	P\$TYPE:	.WORD	0	:LOC. TO HOLD MESSAGE TYPE
1020 001172	000000	P\$SIZE:	.WORD	0	:LOC. TO HOLD MESSAGE SIZE
1021 001174	000000	P\$CPYS:	.WORD	0	:LOC. TO HOLD NO. OF MESSAGE COPIES
1022 001176	000000	P\$PASS:	.WORD	0	:LOC. TO HOLD NO. OF PASSES
1023 001200	000000	NODTY:	.WORD	0	:LOC. TO HOLD NODE TYPE FOR NODE TABLE SETUP
1024 001202	000000	SLOT::	.WORD	0	:USED BY NODE TABLE SUBROUTINES
1025 001204	000000	SLOT1::	.WORD	0	:FOR DEFAULT NODE ADDRESSES
1026 001206	177777	ILLADR:	.WORD	177777	:ILLEGAL ADDRESS FOR COMPARISON
1027 001210	177777		.WORD	177777	: (MUST NOT BE PHYSICALLY SEPARATED FROM
1028 001212	177777		.WORD	177777	: END OF SAVTBL)
1029					: of an incoming frame
1030 001214		LISBUF:	.BLKW	7	: buffer to hold destination, source, and p.t.
1031 001232	100000	LISNXT:	.WORD	LISLOG	: pointer to next open location in log
1032 001234	000000	LISNUM:	.WORD	0	: number of listen commands since log was started
1033 001236	000000	LPACNM:	.WORD	0	: number of frames that passed filter
1034 001240	000000	LBYTEC:	.WORD	0	: byte count of a received frame
1035 001242	000000	LISMIN:	.WORD	0	: total elapsed time of listen command sequence
1036 001244	000000	LISSEC:	.WORD	0	:
1037 001246	000000	LOGFMN:	.WORD	0	: minutes to fill log (zero if not full)
1038 001250	000000	LOGFSC:	.WORD	0	: seconds to fill log (zero if not full)
1039 001252	000	LISFUL:	.BYTE	0	: flag to indicate if the log was filled
1040 001253	000	SOUFLG:	.BYTE	0	: flag indicating presence of source filter
1041 001254	000	DESFLG:	.BYTE	0	: flag indicating presence of destination filter
1042 001255	000	PROFLG:	.BYTE	0	: flag indicating presence of protocol type filter
1043		.	EVEN		
1044 001256	101034	ADRNXT:	.WORD	ADRLIS	: pointer to next free location in addr. list
1045					
1046					:COMMAND LINE TRAVERSE LOCATIONS (USED BY "P\$TRV")
1047					
1048 001260	000000	P\$BUFA:	.WORD	0	:LOC. TO HOLD ADDR. OF CMD LINE BUFFER
1049 001262	000000	P\$TREE:	.WORD	0	:LOC. TO HOLD ADDR. OF PARSING TREE
1050 001264	000000	P\$ACT:	.WORD	0	:LOC. TO HOLD ADDR. OF ACTION ROUTINE
1051 001266	000000	P\$CNT:	.WORD	0	:LOC. TO BE A COUNTER LOCATION
1052 001270	000000	P\$NUM:	.WORD	0	:LOC. TO HOLD NUMERIC VALUE FROM PARSE
1053 001272	000000	P\$RADX:	.WORD	0	:LOC. TO HOLD RADIX(LO) & +/- (HI BYTE)
1054 001274	000	P\$LIST:	.BYTE	0	:INDICATES THAT THE LISTEN COMMAND WAS ENTERED
1055 001275	000	P\$BLD:	.BYTE	0	:INDICATES THAT THE BUILD COMMAND WAS ENTERED
1056 001276	000	P\$HLP:	.BYTE	0	: -1 if help command was typed
1057 001277	000	P\$HEX:	.BYTE	0	: indicate operator data is hex
1058 001300	000	P\$NNUF:	.BYTE	0	:RETURN =0 IF ENOUGH OF COMMAND FOUND
1059 001301	000	P\$GDBD:	.BYTE	0	:RETURN CODE 0 IF NO ERROR FOUND
1060 001302	000	P\$AERR:	.BYTE	0	:RETURN 0 IF 12 DIGIT ADDRESS ENTERED
1061 001303	000	P\$NCMP:	.BYTE	0	:NO DATA COMPARE FLAG

1062 001304 000 P\$MERR: .BYTE 0 ;RETURN -1 IF ERROR IN OPERATOR SELECTED
1063
1064 001305 000 P\$TEXT: .BYTE 0 ;MESSAGE INPUT OCCURED, 0 FOR GOOD INPUT
1065 001306 000 P\$BONC: .BYTE 0 ; indicates text, not address to TRVADR routine
1066
1067 .EVEN
1068 001310 005732' HLPTAB: .WORD HELP1
1069 001312 006033' .WORD HELP2
1070 001314 006126' .WORD HELP3
1071 001316 006177' .WORD HELP4
1072 001320 006250' .WORD HELP5
1073 001322 006350' .WORD HELP6
1074 001324 006463' .WORD HELP7
1075 001326 006574' .WORD HELP8
1076 001330 006664' .WORD HELP9
1077 001332 006753' .WORD HELP10
1078 001334 007044' .WORD HELP11
1079 001336 007142' .WORD HELP12
1080 001340 007247' .WORD HELP13
1081 001342 007346' .WORD HELP14
1082 001344 007440' .WORD HELP15
1083 001346 007453' .WORD HELP16
1084 001350 007542' .WORD HELP17
1085 001352 007645' .WORD HELP18
1086 001354 007715' .WORD HELP19
1087 001356 010020' .WORD HELP20
1088 001360 010076' .WORD HELP21
1089 001362 010161' .WORD HELP22
1090 001364 010262' .WORD HELP23
1091 001366 010362' .WORD HELP24
1092 001370 010473' .WORD HELP25
1093 001372 010601' .WORD HELP26
1094 001374 010673' .WORD HELP27
1095 001376 011001' .WORD HELP28
1096 001400 011105' .WORD HELP29
1097 001402 011207' .WORD HELP30
1098 001404 011326' .WORD HELP31
1099 001406 011376' .WORD HELP32
1100 001410 011505' .WORD HELP33
1101 001412 000000 HLPEND: .WORD 0
1102
1103 001414 017322' MSGTAB: .WORD MSGTY0 ;MESSAGE TYPE ASCII ADDRESS TABLE
1104 001416 017330' .WORD MSGTY1
1105 001420 017335' .WORD MSGTY2
1106 001422 017343' .WORD MSGTY3
1107 001424 017350' .WORD MSGTY4
1108 001426 017355' .WORD MSGTY5
1109 001430 017363' .WORD MSGTY6
1110
1111 : THIS SECTION DEFINES THE DATA PATTERNS USED BY THE EXERCISER
1112
1113 001432 MSGCNT::
1114 001432 000130 MSG0C: .WORD EMSG0-MSG00 ; THE NUMBER OF BYTES IN EACH MESSAGE
1115 001434 000001 MSG1C: .WORD EMSG1-MSG01
1116 001436 000001 MSG2C: .WORD EMSG2-MSG02
1117 001440 000001 MSG3C: .WORD EMSG3-MSG03
1118 001442 000001 MSG4C: .WORD EMSG4-MSG04

1119 001444	000100		MSG5C: .WORD	EMSG5-MSG05
1120 001446	000000		MSG6C: .WORD	0
1121			MSGAD::	
1122 001450	001466'		.WORD	MSG00
1123 001450	001616'		.WORD	MSG01
1124 001452	001617'		.WORD	MSG02
1125 001454	001620'		.WORD	MSG03
1126 001456	001621'		.WORD	MSG04
1127 001460	001622'		.WORD	MSG05
1128 001462	001722'		.WORD	OPSLBF
1129 001464				
1130				
1131 001466	040	041	042 MSG00:: .ascii	\ !"#\$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNPQRSTUVWXYZ\
001471	043	044		045
001474	046	047		050
001477	051	052		053
001502	054	055		057
001505	060	061		062
001510	063	064		065
001513	066	067		070
001516	071	072		073
001521	074	075		076
001524	077	100		101
001527	102	103		104
001532	105	106		107
001535	110	111		112
001540	113	114		115
001543	116	117		120
001546	121	122		123
001551	124	125		126
001554	127	130		131
001557	132			
1132 001560	133	135	136 .ascii	\[]†-abcdefghijklmnopqrstuvwxyz : alphanumeric
001563	055	141	142	
001566	143	144	145	
001571	146	147	150	
001574	151	152	153	
001577	154	155	156	
001602	157	160	161	
001605	162	163	164	
001610	165	166	167	
001613	170	171	172	
1133 001616			EMSG0::	
1134 001616	377		MSG01:: .byte	377 ; message of all ones
1135 001617			EMSG1::	
1136 001617	000		MSG02:: .byte	0 ; message of all zeros
1137 001620			EMSG2::	
1138 001620	252		MSG03:: .byte	252 ; message of alternating ones
1139 001621			EMSG3::	
1140 001621	125		MSG04:: .byte	125 ; message of alternating zeros
1141 001622			EMSG4::	
1142 001622			MSG05::	; CCITT 511 bit test pattern
1143 001622	177603	157427	031011	.word 177603,157427,031011,047321,163715,105221
001630	047321	163715	105221	
1144 001636	143325	142304	040041	.word 143325,142304,040041,104116,052606,172334
001644	104116	052606	172334	
1145 001652	105025	123754	111337	.word 105025,123754,111337,111523,030030,145064

001660 111523 030030 145064
1146 001666 137642 143531 063617 .word 137642,143531,063617,135075,066730,026575
001674 135075 066730 026575
1147 001702 052012 053627 070071 .word 052012,053627,070071,151172,165044,031605
001710 151172 165044 031605
1148 001716 166632 016147 .word 166632,016147
1149 001722
1150 001722 EMSG5:: OPSLBF: .blk 66. ;BUFFER FOR OPERATOR SELECTED MESSAGE TYPE
1151
1152
1153 002024 000000 CFLAG: .WORD 0 ;ACTION ROUTINE CMD ARGUMENT FLAG
1154
1155 ;;CLOCK TABLES, EVENT LOG AND POINTERS
1156 002026 000000 CLKCSR: .WORD 0 ; Clock CSR address
1157 002030 000000 CLKBR: .WORD 0 ; Clock interrupt level
1158 002032 000000 CLKVEC: .WORD 0 ; Clock interrupt vector
1159 002034 000074 CLKHZ: .WORD 60. ; Clock's frequency in Hertz
1160 002036 000000 CLKEN: .WORD 0 ; Clock's CSR value to intrpt. enable it
1161
1162 002040 000000 TIMMIN: .WORD 0 ; Place to keep time-since-start
1163 002042 000000 TIMSEC: .WORD 0
1164 002044 000000 TIMTCK: .WORD 0 ; Place to keep no. of ticks/sec.
1165
1166 002046 000000 TIMER1: .WORD 0 ; Event timer #1 (ticks)
1167 002050 000000 TIMER2: .WORD 0 ; Event timer #2 (ticks)
1168 002052 000000 TIMERS: .WORD 0 ; Event timer #3 (seconds)
.EVEN
1170 ;
1171 ; STUFF FOR DECNET ADDRESS DECODING
1172 ;
1173 002054 000000 DECNET:: .WORD 0
1174 002056 000000 AREA:: .WORD 0
1175
1176 ;
1177 ; POINTERS FOR BOUNCE COMMAND
1178 ;
1179 002060 000000 BNCPKT: .WORD 0 ;points to frame descriptor
1180 002062 000000 BNCBUF: .WORD 0 ; points to buffer
1181 002064 000000 BNCCNT: .WORD 0 ; count of number of bytes used in bounce buffer
1182
1183
1184
1185 ;---+
1186 ; pointers for transmit and receive rings
1187 ;---+
1188
1189 002066 100000 xrgsrt:.word XRING ; first entry in transmit ring
1190 002070 100000 rrgsrt:.word RRING ; first entry in receive ring
1191 002072 100000 xrgcur:.word XRING ; current entry in transmit ring
1192 002074 100000 rrgcur:.word RRING ; current entry in receive ring
1193 002076 100000 xrgnxt:.word XRING ; next entry in transmit ring
1194 002100 100000 rrgnxt:.word RRING ; next entry in receive ring
1195 002102 100036 xrglst:.word XRING+36 ; last entry in transmit ring
1196 002104 100106 rrglst:.word RRING+106 ; last entry in receive ring
1197
1198
1199 ;*****8

```
1200
1201 ;INFORMATION ABOUT THE CURRENT UNIT AS OBTAINED FROM THE HARDWARE P-TABLE
1202 ;
1203 ;*****=====
1204
1205 ;PCSRs of current slot
1206 002106 000000 PCSR0:: .WORD ; address of PCSR0 (port command field
1207 002110 000000 PCSR1:: .WORD ; 1 (state & self test fields
1208 002112 000000 PCSR2:: .WORD ; 2 (pcb address lo 15 bits
1209 002114 000000 PCSR3:: .WORD ; 3 (pcb address hi 2 bits
1210
1211 002116 000000 PCSROC:: .WORD 0 ;PCSR0 CONTENTS
1212 002120 000000 PCSR1C:: .WORD 0 ;PCSR1 CONTENTS
1213 002122 000000 PCSR2C:: .WORD 0 ;PCSR2 CONTENTS
1214 002124 000000 PCSR3C:: .WORD 0 ;PCSR3 CONTENTS
1215
1216
1217 002126 000000 UNACSR:: .WORD 0 ;CSR
1218 002130 000000 UNAVEC:: .WORD 0 ;VECTOR
1219 002132 000000 UNAPRI:: .WORD 0 ;PRIORITY
1220
1221 002134 000000 FRESIZ:: .WORD 0 ;POINTER TO WORD CONTAINING SIZE OF FREE MEMORY
1222 002136 000000 FREMEM:: .WORD 0 ;POINTER TO FREE MEMORY SPACE
1223
1224 002140 000000 UNIT:: .WORD 0 ;CURRENT UNIT NUMBER BEING TESTED
1225
1226
1227 ; broadcast address - FF-FF-FF-FF-FF-FF
1228
1229 002142 177777 brdaddr: .word -1
1230 002144 177777 .word -1
1231 002146 177777 .word -1
1232
1233 ; Port control block function structures
1234
1235 ;port control block
1236 002150 000000 PCBB0:: .word 0 ; port function
1237 002152 000000 PCBB2:: .word 0 ; port function dependent parameters
1238 002154 000000 PCBB4:: .word 0 ; port function dependent parameters
1239 002156 000000 PCBB6:: .word 0 ; port function dependent parameters
1240
1241 ; function table
1242
1243 002160 002230' FUNTAB:: .word $PNOP ; no op
1244 002162 000000 .word 0 ; fill in the hole
1245 002164 002232' .word $RDDE ; read default physical address
1246 002166 000000 .word 0 ; fill in another hole
1247 002170 002242' .word $RDPH ; read physical address
1248 002172 002252' .word $WDPH ; write physical address
1249 002174 002262' .word $RDMC ; read multicast address list
1250 002176 002322' .word $WDMC ; write multicast address list
1251 002200 002362' .word $RDRN ; read descriptor rings
1252 002202 002406' .word $WDRN ; write descriptor rings
1253 002204 002432' .word $RDCN ; read counters
1254 002206 002546' .word $CLRC ; read and clear counters
1255 002210 002556' .word $RDMO ; read mode
1256 002212 002566' .word $WDMO ; write mode
```

GLOBAL DATA SECTION

```

1257 002214 002576'          .word $RDST      : read status
1258 002216 002606'          .word $CLRS      : read and clear status
1259 002220 002616'          .word $DMEM      : dump internal memory
1260 002222 002640'          .word $LMEM      : load internal memory
1261 002224 002650'          .word $RDSY      : read sys id parameters
1262 002226 002660'          .word $WTSY      : write sys id parameters
1263
1264      ;*
1265      ;:      PNOP == 0                      : port no-operation
1266      ;-
1267      .even   $pnop::      .word 0           : no-op
1268 002230 000000
1269
1270      ;+
1271      ;:      RDDEFA == bit01                : read default physical address
1272      ;-
1273      .even
1274
1275 002232 000002          $rdde::      .word 2           : pcbb+0 function read default
1276 002234 000000          depadr::     .word 0           : pcbb+2    physical address
1277 002236 000000
1278 002240 000000
1279
1280      ;+
1281      ;:      RDPHYA == bit02                : read physical address
1282      ;-
1283      .even
1284
1285 002242 000004          $rdph::      .word 4           : pcbb+0 read current (active)
1286 002244 000000          phyadr::     .word 0           : pcbb+2    physical address
1287 002246 000000
1288 002250 000000
1289
1290      ;+
1291      ;:      WDPHYA == bit02!bit00        : write physical address
1292      ;-
1293      .even
1294 002252 000005          $wdph::      .word 5           : pcbb+0 write physical address
1295 002254 000000
1296 002256 000000
1297 002260 000000
1298
1299
1300      ;+
1301      ;:      RDMULA == bit02!bit01        : read multicast address list
1302      ;-
1303      .even
1304 002262 000006          $RDMC::      .word 6           : function code
1305 002264 002272'          .word ucb6      : ucbb address
1306 002266 000000
1307 002270 000000
1308
1309 002272                 UCB6::       .blkw 12.         : enough room for 4 addresses
1310
1311      ;+
1312      ;:      WDMULA == bit02!bit01!bit00 : write multicast address list
1313

```

```
1314
1315
1316 002322 000007
1317 002324 002332
1318 002326 000400
1319 002330 000000
1320
1321 002332 000253
1322 002334 001000
1323 002336 000000
1324 002340
1325
1326
1327 ;+
1328 ; RDRNGS == bit03 ; read both the rcvr and xmit rings
1329 ;-
1330 .even
1331 002362 000010
1332 002364 002372
1333 002366 000000
1334 002370 000000
1335
1336 .even
1337
1338 002372 140000
1339 002374 002000
1340 002376 000000
1341 002400 100000
1342 002402 002000
1343 002404 000000
1344
1345
1346 ;+
1347 ; WDRNGS == bit03!bit00 ; write both the rcvr and xmit rings
1348 ;-
1349
1350 .even
1351
1352 002406 000011
1353 002410 002416
1354 002412 000000
1355 002414 000000
1356
1357 .even
1358
1359 002416 040000
1360 002416 001
1361 002420 005
1362 002421 005
1363 002422 000004
1364
1365 002424 000000
1366 002426 001
1367 002427 005
1368 002430 000010
1369
1370
```

.even \$WDMC:: .word 7 : function code
.word ucb7 : ucbb address
.word 400 : length of list = 1
.word 0 : pcbb+6

ucb7:: .word 253 : multicast address for loopback
.word 1000
.word 0
.blkw 9. : room for three more addresses

;+ RDRNGS == bit03 ; read both the rcvr and xmit rings
;-

.even \$RDRN:: .WORD 10 : FUNCTION CODE
.word UCB10 : ucbb address
.word 0 : null
.word 0 : null

ucb10:: .word XRING+40000 ; ucbb
.word 2000 ; ucbb+2
.word 0 ; ucbb+4
.word RRING ; ucbb+6
.word 2000 ; ucbb+10
.word 0 ; ucbb+12

;+ WDRNGS == bit03!bit00 ; write both the rcvr and xmit rings
;-

.even \$WDRN:: .WORD 11 : FUNCTION CODE
.word UCB11 : ucbb address
.word 0 : null
.word 0 : null

ucb11:: .word 40000 : transmit ring base address
.byte 1 : hi bits of transmit ring base address
.byte 5 : five words per ring entry (1 for port driver)
.word NO.NTR : four transmit descriptors in the ring

.word 0 : receive ring base address
.byte 1 : hi bits of receive ring base address
.byte 5 : five words per ring entry (1 for port driver)
.word NO.NRR : eight receive descriptors in the ring

```
1371
1372
1373      ;+
1374      ; RDCNTS == bit03!bit01      ; read counters
1375      ;-
1376      .even
1377 002432 000012      $RDCN::      .WORD 12      : FUNCTION
1378 002434 002442'      .word UCB12      : ucbb address
1379
1380
1381
1382 002436 000000      .word 0      : DEFAULT COUNT OF COUNTER LIST
1383
1384 002440 000110      .word 110      : 40 (octal)
1385
1386
1387      .even
1388
1389 002442      ucb13::      .word 0      : null
1390 002442 000000      ucb12::      .word 0      : CTRLEN
1391 002444 000000
1392 002446 000000
1393 002450 000000
1394 002452 000000
1395 002454 000000
1396 002456 000000
1397 002460 000000
1398 002462 000000
1399 002464 000000
1400 002466 000000
1401 002470 000000
1402 002472 000000
1403 002474 000000
1404 002476 000000
1405 002500 000000
1406 002502 000000
1407 002504 000000
1408 002506 000000
1409 002510 000000
1410 002512 000000
1411 002514 000000
1412 002516 000000
1413 002520 000000
1414 002522 000000
1415 002524 000000
1416 002526 000000
1417 002530 000000
1418 002532 000000
1419 002534 000000
1420 002536 000000
1421 002540 000000
1422 002542 000000
1423 002544 000000      .word 0      : ucbb
1424
1425      ;+
1426      ; CLRCNTS == bit03!bit01!bit00 ; read and clear counters
1427      ;-
```

```
1428
1429          .even
1430
1431 002546 000013      $clrc::    .WORD 13      : FUNCTION
1432 002550 002442'      .word UCB13   : ucbb address
1433
1434 002552 000000      .word 0       : DEFAULT COUNT OF COUNTER LIST
1435 002554 000040      .word 40     : null
1436
1437
1438
1439
1440          ;( for ucb13:: see ucb 12 above)
1441
1442
1443          ;+
1444          ;      RDMODE == bit03!bit02      ; read internal link mode register
1445          ;-
1446
1447          .even
1448 002556 000014      $rdmo::    .word 14      : function code
1449 002560 000000      .word 0       : a 16 bit copy of the
1450
1451
1452 002562 000000      .word 0       : bits to read the una internal
1453 002564 000000      .word 0       : mode register
1454
1455          ;+
1456          ;      WDMODE == bit03!bit02!bit00 ; write internal link mode register
1457          ;-
1458
1459          .even
1460 002566 000015      $wdmo::    .word 15      : function code
1461 002570 000000      .word 0       : a 16 bit copy of the
1462
1463
1464 002572 000000      .word 0       : bits to write the una internal
1465 002574 000000      .word 0       : mode register
1466
1467
1468          ;+
1469          ;      RDSTA == bit03!bit02!bit01 ; read port status
1470          ;-
1471
1472          .even
1473 002576 000016      $rdst::    .word 16      : function code
1474 002600 000000      status:::   .word 0       : a list of ERRORS and STATUS
1475 002602 000000      .word 0       : lower byte = # of multicast adrs
1476
1477
1478
1479 002604 000000      .word 0       : maximum supported by UNA
1480
1481
1482
1483
1484          ;+
```

```
1485          :      CLRSTA == bit03!bit02!bit01!bit0
1486          :-      ; read and clear write port status
1487
1488          .even
1489 002606 000017      $clrs::      .word 17      ; function code
1490 002610 000000      .word 0       ; a list of ERRORS and STATUS
1491 002612 000000      .word 0       ; lower byte = # of multicast adrs
1492
1493
1494
1495 002614 000000      .word 0       ; maximum supported by UNA
1496
1497
1498
1499
1500
1501          ::+      DMPMEM == bit04      ; dump internal memory
1502          :-      ;
1503
1504          .even
1505 002616 000020      $dmem::      .word 20      ; function code
1506 002620 002626'      .word ucb20    ; ucbb address
1507 002622 000000      .word 0       ; MBZ
1508 002624 000000      .word 0       ; MBZ
1509
1510 002626      ucb20::      .word 0       ; function length (no of words to xfer)
1511 002626 000000      ucb21::      .word 0       ; hdbb - host memory data block address
1512 002630 000000      .word 0       ; internal DEUNA address ...
1513 002632 000000      .word 21040    ; ... changed if DELUA
1514 002634 021040      .word 0       ; extra word for IDBB<23:0> -- if DELUA
1515 002636 000000
1516
1517
1518          ::+      LDMEM == bit04!bit00      ; load DEUNA/DELUA internal memory
1519          :-      ;
1520
1521          .even
1522 002640 000021      $lmem::      .word 21      ; function code
1523 002642 002626'      .word ucb21    ; ucbb address
1524 002644 000000      .word 0       ;
1525 002646 000000      .word 0       ;
1526
1527
1528          ::+      RDSYS == bit04!bit01      ; read system id
1529          :-      ;
1530
1531          .even
1532 002650 000022      $rdsy::      .word 22      ; function code
1533 002652 002670'      .word ucb22    ; ucbb address
1534 002654 000000      .word 0       ;
1535 002656 000033      .word 27.     ; length of id message
1536
1537          ::+      WTSYS == bit04!bit01!bit00      ; write system id
1538          :-      ;
1539
1540
1541 002660 000023      $wtsy::      .word 23      ; function code
```

GLOBAL DATA SECTION

```

1542 002662 002670'          .WORD ucb23 : ucbb address
1543 002664 000000            .WORD 0
1544 002666 000033            .WORD 27. : length of id message

1545
1546 002670 ucb22:           .WORD 0 :udbb+0
1547 002670 000000            .WORD 0 :udbb+2
1548 002672 000000            .WORD 0 :udbb+4
1549 002674 000000            .WORD 0 :udbb+6
1550 002676 000000            .WORD 0 :udbb+10
1551 002700 000000            .WORD 0 :udbb+12
1552 002702 000000            .WORD 0 :udbb+14
1553 002704 000000            .WORD 0 :udbb+16
1554 002706 000000            .WORD 0 :udbb+20
1555 002710 000000            .WORD 0 :udbb+22
1556 002712 000000            .WORD 0 :udbb+24
1557 002714 000000            .WORD 0 :udbb+26
1558 002716 000000            .WORD 0 :udbb+30
1559 002720 000000            .WORD 0 :udbb+32
1560 002722 000000            .WORD 0 :udbb+34
1561 002724 000000            .WORD 0 :udbb+36
1562 002726 000000            .WORD 0 :udbb+40
1563 002730 000000            .WORD 0 :udbb+42
1564 002732 000000            .WORD 0 :udbb+44
1565 002734 000000            .WORD 0 :udbb+46
1566 002736 000000            .WORD 0 :udbb+50
1567 002740 000000            .WORD 0 :udbb+52
1568 002742 000000            .WORD 0 :udbb+54
1569 002744 000000            .WORD 0 :udbb+56
1570 002746 000000            .WORD 0 :udbb+60
1571 002750 000000            .WORD 0 :udbb+62
1572 002752 000000            .WORD 0 :udbb+64
1573 002754 000000            .WORD 0 :udbb+64
1574

1575 002756 000000            UDBB:: .WORD 0 :UNIBUS DATA BLOCK BASE
1576 002760 000000            .WORD 0 :+2
1577 002762 000000            .WORD 0 :+4
1578 002764 000000            .WORD 0 :+6

1579
1580
1581          : SUMMARY DATA COUNTERS
1582
1583
1584 002766 000000            $rec:: .WORD 0 : messages received
1585 002770 000000            $nrec:: .WORD 0 : messages not received
1586 002772 000000            $len:: .WORD 0 : length errors
1587 002774 000000            $comp:: .WORD 0 : compare errors
1588 002776 000000            $byte:: .WORD 0 : bytes compared
1589 003000 000000            $xfer:: .WORD 0 : bytes transferred

1590
1591
1592          : DEUNA/DELUA DRIVER AND ASSOCIATED SUBROUTINES DATA
1593
1594
1595 003002 000000            fatflg:: .WORD 0 : fatal error flag
1596 003004 000000            pceflg:: .WORD 0 : port command error flag
1597 003006 000000            nirent:: .WORD 0 : DEUNA/DELUA receive message counter
1598 003010 000000            xflag:: .WORD 0 : frame transmitted flag

```

1599 003012 000000 dniflg:::word 0 : done interrupt flag
1600 003014 000000 rbfcnt:::word 0 : receive buffers lost counter
1601 003016 000000 bcount:::word 0 : unexplained interrupts counter
1602 003020 000000 errflg:::word 0 : error flag
1603 003022 000000 timeout:::word 0 : time out counter
1604 003024 000000 retrys:::word 0 : counter for frames failing due to rtry error
1605 003026 000000 rcverr:::word 0 : counts no. of buffers received with errors
1606 003030 000000 rcvbuf:::word 0 : counts no. of good buffers received
1607 003032 000000 count:::word 0 : used in BLDBUF subroutine as counter
1608 003034 000220 prot00:::word 000220 : protocal type for loopback messages
1609 003036 001140 prot02:::word 001140 : protocal type for remote console
1610 003040 tempbl:::blkw 24 : reserve space to hold a system id field
1611 003110 000000 temp:::word 0 : used in XMIT as temporary storage
1612 003112 000000 temp1:::word 0 : used for temporary storage
1613 003114 000000 temp2:::word 0 : used for temporary storage
1614 003116 000000 temp3:::word 0 : used for temporary storage
1615 003120 xfer:::word 0 : stores 'bytes transferred'
1616 003122 000000 cpycnt:::word 0 : 'no. of copies' counter for looping
1617 003124 000000 pccall:::word 0 : stores pc of calling routine for error reports
1618 003126 000000 buflen:::word 0 : stores transmit buffer length
1619 003130 000000 cmpbuf:::word 0 : stores location of data buffer to be compared
1620 003132 patch:::blkw 40. : 40 words for program patch

1621
1622
1623 : Request ID Message Format
1624
1625
1626 003252 reqid:::
1627 003252 000003 .word 3 : byte count (=3 for request id)
1628 003254 000005 .word 5 : function code for request id
1629 003256 051115 .word "MR" : receipt number

1630
1631
1632 : Loop Direct Message
1633
1634
1635 .even
1636
1637 003260 LOPDIR:::
1638 003260 000000 .word 0 : skip count
1639 003262 000002 .word 2 : function = forward data
1640 003264 000000 000000 000000 .word 0,0,0 : local node address
1641 003272 000001 .word 1 : function = reply
1642 003274 000000 000000 000000 .word 0,0,0 : local node address

1643
1644
1645 : Transmit assist message
1646
1647
1648 003302 TASIST:::
1649 003302 000000 .word 0 : skip count
1650 003304 000002 .word 2 : function = forward data
1651 003306 000000 000000 000000 .word 0,0,0 : transmit assist address
1652 003314 000002 .word 2 : function = forward data
1653 003316 000000 000000 000000 .word 0,0,0 : local node address
1654 003324 000001 .word 1 : function = reply
1655 003326 000000 000000 000000 .word 0,0,0 : local node address

```
1656
1657
1658 ; Recieve assist message
1659 ;
1660
1661 003334 RASIST:::
1662 003334 000000 .word 0 ; skip count
1663 003336 000002 .word 2 ; function = forward data
1664 003340 000000 000000 000000 .word 0.0.0 ; transmit assist address
1665 003346 000002 .word 2 ; function = forward data
1666 003350 000000 000000 000000 .word 0.0.0 ; local node address
1667 003356 000001 .word 1 ; function = reply
1668 003360 000000 000000 000000 .word 0.0.0 ; local node address
1669
1670
1671 ; Full assist message
1672 ;
1673
1674 003366 FASIST:::
1675 003366 000000 .word 0 ; skip count
1676 003370 000002 .word 2 ; function = forward data
1677 003372 000000 000000 000000 .word 0.0.0 ; target node address
1678 003400 000002 .word 2 ; function = forward data
1679 003402 000000 000000 000000 .word 0.0.0 ; assist node address
1680 003410 000002 .word 2 ; function = forward data
1681 003412 000000 000000 000000 .word 0.0.0 ; local node address
1682 003420 000001 .word 1 ; function = reply
1683 003422 000000 000000 000000 .word 0.0.0 ; local node address
1684
1685
1686 .SBTTL COMMAND LINE ACTION TREE
1687
1688 ;SAMPLE CLI TREE NODE (ALWAYS AT LEAST 1 WORD)
1689
1690
1691 ; ! ACTION ! CHAR CODE !
1692 ;-----;
1693 ; ! MISS DISPLACEMENT ! ONLY IF "MISS" ARGUMENT DEFINED
1694 ;-----;
1695 ; ! NEXT MODE DISPLMINT ! ONLY IF "ASCII" ARGUMENT DEFINED
1696 ;-----;
1697 ; ! ASCIZ MATCH STRING ! ONLY IF "ASCII" ARGUMENT DEFINED
1698 ; ! (.EVEN) !
1699 ;-----;
1700
1701 003430 .NLIST ME
1702 CLITRE:
1703 ;FIRST KEYWORD
1704 003430 CLI CLISPA,0,N10$ ;SKIP ANY LEADING SPACES
1705 003434 N10$: CLI <'?>,HELP,N12$ ;IS THE FIRST NON-SP CHAR. A "?"
1706 003440 CLI CLISPA,0,N11$ ; skip spaces
1707 003444 N11$: CLI CLISPA,0,N50$ ; error if non-space characters left
1708 003450 N12$: CLI CLISTR,HELP,N14$,<'HELP'> ;ELSE IS FIRST WORD A "HELP"
1709 003464 CLI CLISPA,0,N13$ ; skip spaces after executing
1710 003470 N13$: CLI CLISPA,0,N50$ ; error if nonspace chars left
1711 003474 N14$: CLI CLISTR,NOTNUF,N16$,<'NODE'> ;ELSE IS FIRST WORD A "NODE"
1712 003510 CLI CLIBR,0,N80$ ; IF YES, BR N80$
```

1713 003514	N16\$:	CLI <'B>,NOTNUF,N18\$; is char a b?
1714 003520	CLI	CLISTR,BUILD,N17\$,<'UILD'>	; ELSE IS FIRST WORD A "BUILD"
1715 003534	CLI	CLIBR,O,N70\$; IF YES, SEE BR N70\$
1716 003540	N17\$:	CLI CLISTR,O,N50\$,<'OUNCE'>	; IS IT BOUNCE COMMAND?
1717 003554	CLI	CLIBR,O,N300\$; branch if it is
1718 003560	N18\$:	CLI CLISTR,NOTNUF,N20\$,<'RUN'>	; ELSE IS FIRST WORD A "RUN"
1719 003572	CLI	CLIBR,O,N180\$; IF YES, BR N180\$
1720 003576	N20\$:	CLI <'S>,NOTNUF,N25\$; ELSE IS FIRST CHAR. A "S"
1721 003602	CLI	CLISTR,O,N22\$,<'HOW'>	; IF YES IS REST OF WORD "HOW"
1722 003614	CLI	CLIBR,O,N100\$; IF YES, BR N100\$
1723 003620	N22\$:	CLI CLISTR,SUMMRY,N23\$,<'UMMARY'>	; ELSE IS REST OF WORD "UMMARY"
1724 003636	CLI	CLIEXI,O	; IF YES, DO "SUMM" AND EXIT
1725 003640	N23\$:	CLI CLISTR,O,N24\$,<'AVE'>	; ELSE IS REST OF WORD "AVE"
1726 003652	CLI	CLISPA,CSAVR4,N231\$; SKIP SPACES
1727 003656	N231\$:	CLI CLIEXI,CSAVE	; DO SAVE AND EXIT
1728 003660	N24\$:	CLI CLIERR,O	; ELSE "ILL COMMAND"
1729 003662	CLI	CLIEXI,O	; EXIT
1730 003664	N25\$:	CLI CLISTR,NOTNUF,N26\$,<'CLEAR'>	; ELSE IS FIRST WORD A "CLEAR"
1731 003700	CLI	CLIBR,O,N120\$; IF YES, BR N120\$
1732 003704	N26\$:	CLI CLISTR,NOTNUF,N28\$,<'IDENTIFY'>	; ELSE IS FIRST WORD "IDENTIFY"
1733 003724	CLI	CLIBR,O,N140\$; IF YES, GET ADDRS, BR N140\$
1734 003730	N28\$:	CLI CLISTR,NOTNUF,N29\$,<'MESSAGE'>	; ELSE IS FIRST WORD "MESSAGE"
1735 003746	CLI	CLIBR,O,N160\$; IF YES, BR N160\$
1736 003752	N29\$:	CLI CLISTR,O,N30\$,<'UNSAVE'>	; ELSE IS FIRST WORD "UNSAVE"
1737 003770	CLI	CLIBR,O,N210\$; IF YES, BR TO N210\$
1738 003774	N30\$:	CLI CLISTR,EXIT,N31\$,<'EXIT'>	; ELSE IS FIRST WORD "EXIT"
1739 004010	CLI	CLIEXI,O	; IF YES EXIT
1740 004012	N31\$:	CLI CLISTR,NOTNUF,N32\$,<'FUNCTION'>	; ELSE IS FIRST WORD "FUNCTION"
1741 004032	CLI	CLIBR,O,N200\$; IF YES, BR N200\$
1742 004036	N32\$:	CLI CLISTR,LISTEN,N50\$,<'LISTEN'>	; ELSE IS FIRST WORD "LISTEN"
1743 004054	CLI	CLIBR,O,N145\$; IF YES, BR N145\$
1744 004060	N50\$:	CLI CLIERR,O	; OTHERWISE "ILL CMD".
1745 004062	CLI	CLIEXI,O	; EXIT
1746			
1747		:SECOND KEYWORD FOR BUILD COMMAND	
1748			
1749 004064	N70\$:	CLISPA,O,N72\$; SKIP LEADING SPACES
1750 004070	N72\$:	CLI <'/>,NULL,N50\$; ERR IF ILLEGAL QUALIFIER
1751 004074	CLI	CLISPA,O,N74\$; skip spaces
1752 004100	N74\$:	CLI CLISTR,SETQIK,N50\$,<'QUICK'>	; SET QUCK BUILD FLAG IF QUICK
1753 004114	CLI	CLISPA,O,N76\$; skip spaces
1754 004120	N76\$:	CLI CLISPA,O,N50\$; error if more to command
1755 004124	N78\$:	CLI CLIEXI,O	; EXIT
1756			
1757		:SECOND KEYWORD (ADR/TYPE) FOR NODE COMMAND	
1758			
1759 004126	N80\$:	CLISPA,O,N81\$; SKIP ANY LEADING SPACES
1760 004132	N81\$:	CLI CLIBR,CSAVR4,N82\$; SAVE STRING POINTER LOCATION
1761 004136	N82\$:	CLI CLIBR,NODE,N90\$; PARSE THROUGH ADDRESS,CHECK
1762			;FOR TARGET OR ASSIST, DO NODE
1763 004142	N90\$:	CLI CLIBIF,O,N50\$;TAKE ERROR BRANCH IF ERROR EXISTS
1764 004146	N95\$:	CLI CLIEXI,O	;EXIT
1765			
1766		:SECOND KEYWORD FOR SHOW COMMAND	
1767			
1768 004150	N100\$:	CLISPA,O,N101\$;SKIP LEADING SPACES
1769 004154	N101\$:	CLI CLISTR,CNODE,N102\$,<'NODES'>	;IS NEXT WORD "NODES"

```

1770 004170      CLI    CLIBR,0,N110$ ; IF YES, SET FLAG, BR N110$
1771 004174      N102$: CLI    CLISTR,CSHMSG,N104$,<'MESSAGE'> ;ELSE IS NEXT WORD "MESSAGE"
1772 004212      CLI    CLIBR,0,N110$ ; IF YES ,SET FLAG, BR N110$
1773 004216      N104$: CLI    CLISTR,CCNTR,N106$,<'COUNTERS'> ;ELSE IS NEXT WORD "COUNTERS"
1774 004236      CLI    CLIBR,0,N110$ ; GO TO COUNTERS ROUTINE
1775 004242      N106$: CLI    CLISTR,CSLIST,N108$,<'LISTEN'> ;ELSE IS NEXT WORD "LISTEN"
1776 004260      CLI    CLIBR,0,N110$ ; DO LISTEN ROUTINE AND BRANCH
1777 004264      N108$: CLI    CLIBR,0,N50$ ;ELSE "ILL COMMAND"
1778 004270      N110$: CLI    CLISPA,0,N112$ ; skip spaces
1779 004274      N112$: CLI    CLISPA,0,N50$ ; error if more to command
1780 004300      CLI    CLIEXI,0 ;EXIT

1781
1782      :SECOND KEYWORD FOR CLEAR COMMAND
1783
1784 004302      N120$: CLI    CLISPA,0,N121$ ;SKIP LEADING SPACES
1785 004306      N121$: CLI    CLISTR,0,N130$,<'NODE'> ;IS NEXT WORD "NODE"
1786 004322      CLI    CLISPA,0,N122$ ; IF YES SKIP SPACES
1787 004326      N122$: CLI    <'/>,CSAVR4,N50$ ; LOOK FOR DELIMETER, ELSE "ILL COM"
1788 004332      N1122$: CLI    CLISPA,0,N1124$ ; skip spaces
1789 004336      N1124$: CLI    <'A>,0,N123$ ; IS NEXT CHAR. AN "A"
1790 004342      CLI    CLISTR,CNODAL,N124$,<'LL'> ; IF YES, IS WORD "ALL"
1791 004354      CLI    CLIBR,0,N135$ ; IF YES, SET FLAG, BR N135$
1792 004360      N123$: CLI    <'N>,0,N124$ ; ELSE IS NEXT CHAR. AN "N"
1793 004364      CLI    CLISPA,0,N1123$ ; skip spaces
1794 004370      N1123$: CLI    CLIOCT,0,N50$ ; IF YES, STORE NODE LOGICAL NAME
1795 004374      CLI    CLIBR,CNDLOG,N127$ ; BR TO CLR. NODE LOGICAL ROUTINE
1796 004400      N124$: CLI    CLIBR,CEXADR,N126$ ; ELSE, EXTRACT ADDRESS
1797 004404      N126$: CLI    CLIBR,CNDADR,N127$ ; SET FLAG
1798 004410      N127$: CLI    CLISPA,0,N128$ ; skip spaces
1799 004414      N128$: CLI    54,0,N129$ ; is there more?
1800 004420      CLI    CLIBR,0,N1122$ ; yes
1801 004424      N129$: CLI    CLISPA,0,N50$ ; no, error if more text
1802 004430      N130$: CLI    CLISTR,CCLMSG,N132$,<'MESSAGE'> ;ELSE IS NEXT WORD "MESSAGE"
1803 004446      CLI    CLIBR,0,N135$ ; IF YES, SET FLAG, BR N135$
1804 004452      N132$: CLI    CLISTR,CCLSUM,N134$,<'SUMMARY'> ;ELSE IS NEXT WORD "SUMMARY"
1805 004470      CLI    CLIBR,0,N135$ ; IF YES, CLEAR TABLE AND EXIT
1806 004474      N134$: CLI    CLISTR,CCLIST,N136$,<'LISTEN'> ;ELSE IS NEXT WORD "LISTEN"
1807 004512      CLI    CLIBR,0,N135$ ; IF YES, CLEAR LOG AND EXIT
1808 004516      N136$: CLI    CLIERR,0 ;ELSE, "ILL COMMAND",
1809 004520      N135$: CLI    CLIEXI,0 ;EXIT

1810
1811      :ADDRESS FOR IDENTIFY COMMAND
1812
1813 004522      N140$: CLI    CLISPA,0,N141$ ;SKIP LEADING SPACES
1814 004526      N141$: CLI    <'N>,0,N142$ ; Is this a logical address
1815 004532      CLI    CLIOCT,0,N50$ ; YES, get octal value ...
1816 004536      CLI    CLIBR,BNCLOG,N1412$ ; ... and look up value in nodetable
1817 004542      N1412$: CLI    CLIBIF,0,N50$ ; exit on error
1818 004546      CLI    CLIBR,0,N143$ ; 
1819 004552      N142$: CLI    CLIBR,CSAVR4,N1421$ ;SAVE POINTER TO FIRST CHAR. OF ADDRESS
1820 004556      N1421$: CLI    CLIBR,CEXADR,N1431$ ;GET ADDRESS
1821 004562      N1431$: CLI    CLIBIF,0,N50$ ; exit on error
1822 004566      CLI    CLIBR,0,N143$ ; 
1823 004572      N143$: CLI    CLIEXI,IDENT ;DO "IDENTIFY", EXIT

1824
1825
1826 004574      N145$: CLI    CLISPA,0,N146$ ;SKIP LEADING SPACES

```

1827 004600	N146\$: CLI	<'/>,0,N1461\$: PARSE THROUGH OPTIONAL "/"
1828 004604	N1461\$: CLI	CLISTR,0,N151\$,<'SOURCE'>	: IS NEXT WORD "SOURCE"
1829 004622	CLI	<'>,0,N50\$: NEXT CHAR. MUST BE A "/"
1830 004626	CLI	<'N>,0,N1491\$: IS THIS A LOGICAL ADDRESS?
1831 004632	CLI	CLISPA,0,N147\$: YES SKIP SPACES
1832 004636	N147\$: CLI	CLIOCT,0,N50\$: EXTRACT NUMBER, ERROR IF NONE
1833 004642	CLI	CLIBR,BNCLOG,N148\$: GET ADDR. FROM NODE TABLE
1834 004646	N148\$: CLI	CLIBR,SQUADR,N145\$: SAVE ADDR. IN SOURCE FILTER AND CONT.
1835 004652	N1491\$: CLI	CLIBR,CSAVR4,N149\$: SAVE R4
1836 004656	N149\$: CLI	CLIBR,CEXADR,N150\$: EXTRACT ADDRESS
1837 004662	N150\$: CLI	CLIBIF,0,N50\$: DON'T CONTINUE IF ERROR
1838 004666	CLI	CLIBR,SQUADR,N145\$: SAVE ADDR. IN SOURCE FILTER AND CONT.
1839 004672	N151\$: CLI	CLISTR,0,N156\$,<'DESTINATION'>	: ELSE IS NEXT WORD "DESTINATION"?
1840 004714	CLI	<'>,0,N50\$: NEXT CHAR. MUST BE A "/"
1841 004720	CLI	<'N>,0,N1541\$: IS THIS A LOGICAL ADDRESS?
1842 004724	CLI	CLISPA,0,N152\$: YES, SKIP SPACES
1843 004730	N152\$: CLI	CLIOCT,0,N50\$: EXTRACT NUMBER, ERROR IF NONE
1844 004734	CLI	CLIBR,BNCLOG,N153\$: GET ADDR. FROM NODE TABLE
1845 004740	N153\$: CLI	CLIBR,DESADR,N145\$: SAVE ADDR. IN DEST. FILTER AND CONT.
1846 004744	N1541\$: CLI	CLIBR,CSAVR4,N154\$: SAVE R4
1847 004750	N154\$: CLI	CLIBR,CEXADR,N155\$: EXTRACT ADDRESS
1848 004754	N155\$: CLI	CLIBIF,0,N50\$: DON'T CONTINUE IF ERROR
1849 004760	CLI	CLIBR,DESADR,N145\$: SAVE ADDR. IN DEST. FILTER AND CONT.
1850 004764	N156\$: CLI	CLISTR,0,N50\$,<'PROTOCOL'>	: ELSE NEXT WORD MUST BE "PROTOCOL" OR ERROR
1851 005004	CLI	<'>,0,N50\$: NEXT CHAR. MUST BE A "/"
1852 005010	CLI	CLIBR,CSAVR4,N157\$: SAVE R4
1853 005014	N157\$: CLI	CLIBR,CEXPRO,N145\$: EXTRACT PROTOCOL TYPE AND CONT.
1854	:REMAINING COMMAND LINE FOR MESSAGE COMMAND		
1855			
1856			
1857 005020	N160\$: CLI	CLISPA,0,N161\$: SKIP LEADING SPACES
1858 005024	N161\$: CLI	<'>,0,N178\$: IF CHAR. "/", CONT., ELSE BR N178\$
1859 005030	CLI	CLISTR,0,N170\$,<'TYPE'>	: IS NEXT WORD "TYPE"
1860 005044	CLI	<'=>,0,N50\$: IF YES, FOLLOWED BY "="?
1861 005050	CLI	CLISTR,CALPHA,N162\$,<'ASCII'>	: IF "ASCII", SET FLAG
1862 005064	CLI	CLIBR,0,N168\$: CONTINUE AT N168\$
1863 005070	N162\$: CLI	CLISTR,CONES,N163\$,<'ONES'>	: IF "ONES", SET FLAG
1864 005104	CLI	CLIBR,0,N168\$: CONTINUE AT N168\$
1865 005110	N163\$: CLI	CLISTR,CZERO5,N164\$,<'ZEROS'>	: IF "ZEROS", SET FLAG
1866 005124	CLI	CLIBR,0,N168\$: CONTINUE AT N168\$
1867 005130	N164\$: CLI	CLISTR,C1ALT,N165\$,<'1ALT'>	: IF "1ALT", SET FLAG
1868 005144	CLI	CLIBR,0,N168\$: CONTINUE AT N168\$
1869 005150	N165\$: CLI	CLISTR,COALT,N166\$,<'OALT'>	: IF "OALT", SET FLAG
1870 005164	CLI	CLIBR,0,N168\$: CONTINUE AT N168\$
1871 005170	N166\$: CLI	CLISTR,CCITT,N167\$,<'CCITT'>	: IF "CCITT", SET FLAG
1872 005204	CLI	CLIBR,0,N168\$: CONTINUE AT N168\$
1873 005210	N167\$: CLI	CLISTR,CSAVR4,N50\$,<'TEXT'>	: IF NOT TEXT, ERROR
1874 005224	CLI	<'=>,COPRSL,N50\$: IF "OPERATOR", SET FLAG
1875 005230	CLI	CLIBR,0,N168\$: AND INPUT SPECIFIED STRING
1876 005234	N168\$: CLI	CLIBR,CTYPE,N160\$: DO "TYPE", CHECK FOR MORE INPUT
1877 005240	N170\$: CLI	CLISTR,0,N175\$,<'SIZE'>	: ELSE IS WORD "SIZE"
1878 005254	CLI	CLISPA,0,N1701\$: skip spaces
1879 005260	N1701\$: CLI	<'=>,0,N50\$: IF YES, FOLLOWED BY "="?
1880 005264	CLI	CLISPA,0,N1702\$: skip spaces
1881 005270	N1702\$: CLI	CLIDEC,CSIZE,N50\$: STORE NUMBER IN M\$SIZE
1882 005274	CLI	CLIBR,0,N160\$: CHECK FOR MORE INFO
1883 005300	N175\$: CLI	CLISTR,0,N176\$,<'COPIES'>	: ELSE IS WORD "COPIES"

COMMAND LINE ACTION TREE

```

1884 005316          CLI    CLISPA,0,N1751$      ; skip spaces
1885 005322          N1751$: CLI    <'=>,0,N50$   ; IF YES, FOLLOWED BY "="?
1886 005326          CLI    CLISPA,0,N1752$      ; skip spaces
1887 005332          N1752$: CLI    CLIDEC,CCPYS,N50$ ; STORE NUMBER IN M$CPYS
1888 005336          CLI    CLIBR,0,N160$       ; CHECK FOR MORE INFO
1889 005342          N176$:  CLI    CLISTR,NCMPAR,N177$,<'NOCOMPARE'> ; IF NO DATA CHECKING, SET FLAG
1890 005362          CLI    CLIBR,0,N160$       ; CONTINUE PROCESSING
1891 005366          N177$:  CLI    CLISTR,0,N178$,<'TEXT'> ; branch not "text" command?
1892 005402          CLI    CLISPA,0,N1771$      ; skip spaces
1893 005406          N1771$: CLI    <'=>,CSAVR4,N50$ ; error if wrong delimiter
1894 005412          CLI    CLISPA,0,N1772$      ; skip spaces
1895 005416          N1772$: CLI    CLIBR,COPRSL,N1773$ ; get message
1896 005422          N1773$: CLI    CLIBR,0,N160$       ; process next command
1897 005426          N178$:  CLI    CLIBR,0,N50$       ;ELSE "ILL COMMAND"

1898
1899          ;SECOND KEYWORD FOR RUN COMMAND
1900
1901 005432          N180$:  CLI    CLISPA,0,N181$      ;SKIP LEADING SPACES
1902 005436          N181$:  CLI    CLISTR,CLUPPR,N182$,<'LOOPPAIR'> ;IS NEXT WORD "LOOPPAIR"
1903 005456          CLI    CLIBR,0,N185$       ; IF YES, SET "LOOPPAIR" FLAG
1904 005462          N182$:  CLI    CLISTR,CRNALL,N183$,<'ALL'> ;ELSE IS NEXT WORD "ALL"
1905 005474          CLI    CLIBR,0,N185$       ; IF YES, SET "ALL" FLAG
1906 005500          N183$:  CLI    CLISTR,CDIR,N184$,<'DIRECT'> ;ELSE IS NEXT WORD "DIRECT"
1907 005516          CLI    CLIBR,0,N185$       ; IF YES, SET "DIRECT" FLAG
1908 005522          N184$:  CLI    CLISTR,CPATRN,N50$,<'PATTERN'> ;ELSE IS NEXT WORD "PATTERN"
1909 005540          N185$:  CLI    CLIBR,CDEFLT,N186$ ;SEE IF DEFAULT OF 1 PASS
1910 005544          N186$:  CLI    CLISPA,0,N1861$      ; skip spaces
1911 005550          N1861$: CLI    <'>,0,N190$       ;PARSE THROUGH SWITCH
1912 005554          CLI    CLISPA,0,N1862$      ; skip spaces
1913 005560          N1862$: CLI    CLISTR,0,N50$,<'PASS'> ; error if not "pass"
1914 005564          CLI    CLISPA,0,N1863$      ; skip spaces
1915 005604          N1863$: CLI    <'=>,0,N50$       ;PARSE THROUGH "="
1916 005610          CLI    CLISPA,0,N1864$      ; skip spaces
1917 005610          N1864$: CLI    CLIDEC,0,N50$       ;GET PASS COUNT
1918 005614          N190$:  CLI    CLIEXI,CRUN        ;RUN TEST AND EXIT

1919
1920          ;REMAINING COMMAND LINE FOR FUNCTION COMMAND
1921
1922 005616          N200$:  CLI    CLISPA,0,N201$      ; SKIP SPACES
1923 005622          N201$:  CLI    CLI OCT,CFUNCT,N50$ ; GET OCTAL NUMBER AND DO FUNCT
1924 005626          CLI    CLIEXI,0           ; EXIT

1925
1926          ;REMAINING COMMAND LINE FOR UNSAVE COMMAND
1927
1928 005630          N210$:  CLI    CLISPA,CSAVR4,N50$ ; SAVE POINTER TO FILE NAME
1929 005634          CLI    CLIEXI,CUNSVF        ; DO UNSAVE FROM FILE AND EXIT

1930
1931
1932          ; REST OF BOUNCE COMMAND
1933
1934 005636          N300$:  CLI    CLISPA,0,N310$      ; skip spaces
1935 005642          N310$:  CLI    <'>,0,N50$       ; error if not correct delimiter
1936 005646          N315$:  CLI    CLISPA,0,N320$      ; skip spaces
1937 005652          N320$:  CLI    <'N>,0,N331$      ; error if illegal character
1938 005656          N330$:  CLI    CLI OCT,0,N50$     ; extract number, error if none
1939 005662          CLI    CLIBR,BNCLOG,N335$ ; get address from node table
1940 005666          N331$: CLI    CLIBR,CSAVR4,N332$ ; save r4

```

1941 005672 N332\$:: CLI CLIBR,CEXADR,N335\$: extract address
1942 005676 N335\$:: CLI CLIBIF,0,N50\$: don't continue if error
1943 005702 CLI CLIBR,BOUNCE,N340\$: put address into buffer
1944 005706 N340\$:: CLI CLISPA,0,N350\$: skip spaces
1945 005712 N350\$:: CLI 054,0,N50\$: error if not end and not comma
1946 005716 CLI CLIBR,0,N315\$: process next input
1947
1949 ;*****
1950 : THE ERRTBL MACRO IS REQUIRED IF YOU INTEND TO REPORT ERRORS USING
1951 : THE "ERROR" MACRO. THE ERRTBL MACRO EXPANDS INTO FOUR WORDS THAT
1952 : ARE USED BY THE RUNTIME SERVICES DURING AN ERROR CALL: ERROR TYPE,
1953 : ERROR NUMBER, ADDRESS OF ERROR MESSAGE AND ADDRESS OF MESSAGE
1954 : BLOCK. THERE MUST BE ONLY ONE ERRTBL IN ANY PROGRAM. THIS SECTION
1955 : IS OPTIONAL. REMOVE IF IT IF YOU ARE NOT GOING TO USE THE ERROR
1956 : MACRO. CHANGE THE POINTER MACRO TO REFLECT THIS SECTION'S DEL-
1957 : ETION IF YOU REMOVE IT.
1958 ;*****
1960
1961 005722 ERRTBL
005722 000000 ERRTYP:: .WORD 0
005724 000000 ERRNBR:: .WORD 0
005726 000000 ERRMSG:: .WORD 0
005730 000000 ERRBLK:: .WORD 0

1963 .SBTTL GLOBAL TEXT SECTION
1964
1965 :++
1966 : THE GLOBAL TEXT SECTION CONTAINS FORMAT STATEMENTS,
1967 : MESSAGES, AND ASCII INFORMATION THAT ARE USED IN
1968 : MORE THAN ONE TEST.
1969 :--
1970 .nlist bin :::::
1971 005732 HELP1: .ASCIZ \NIACOMMAND SUMMARY FOR THE NETWORK INTERCONNECT EXERCISER (NIE)\
1972 006033 HELP2: .ASCIZ \NIA(it is only necessary to type the letters in brackets)\
1973 006126 HELP3: .ASCIZ \N2[H]elp or ? - types this help text.\
1974 006177 HELP4: .ASCIZ \N2[E]xit - return to the supervisor.\
1975 006250 HELP5: .ASCIZ \N2[SH]ow [N]odes - prints information in current node table.\
1976 006350 HELP6: .ASCIZ \N2[SH]ow [M]essage - prints the selected message type, size and copies.\
1977 006463 HELP7: .ASCIZ \N2[SH]ow [C]ounters - prints the low level counters of the HOST NODE.\
1978 006574 HELP8: .ASCIZ \N2[R]un [L]ooppair/Pass=nn - runs the looppair test.\
1979 006664 HELP9: .ASCIZ \N2[R]un [A]ll/Pass=nn - runs the node-to-node test.\
1980 006753 HELP10: .ASCIZ \N2[R]un [D]irect/Pass=nn - runs the loop direct test.\
1981 007044 HELP11: .ASCIZ \N2[R]un [P]attern/Pass=nn - runs the message pattern test.\
1982 007142 HELP12: .ASCIZ \N2[M]essage/[T]ype=a/[S]ize=n/[C]opies=m - allows the operator to\
1983 007247 HELP13: .ASCIZ \NIAmodify the default message type, size and copy parameters.\
1984 007346 HELP14: .ASCIZ \N2[N]ode adr - enters a physical address into the node\
1985 007440 HELP15: .ASCIZ \NIAtable.\
1986 007453 HELP16: .ASCIZ \N2[S]ummary - prints a summary of the test results.\
1987 007542 HELP17: .ASCIZ \N2[B]uild - builds a table of remote node physical addresses by\
1988 007645 HELP18: .ASCIZ \NIAlistening to ID messages on the NI.\
1989 007715 HELP19: .ASCIZ \N2[C]lear [N]ode/adr - removes the node specified by either adr\
1990 010020 HELP20: .ASCIZ \NIAor node logical name from the node table.\
1991 010076 HELP21: .ASCIZ \N2[C]lear [N]ode/[A]ll - clears the node table.\
1992 010161 HELP22: .ASCIZ \N2[C]lear [M]essage - sets all message parameters to default.\
1993 010262 HELP23: .ASCIZ \N2[C]lear [S]ummary - clears the table of summary test data.\
1994 010362 HELP24: .ASCIZ \N2[I]dentify adr - uses the request ID function to identify NI nodes.\
1995 010473 HELP25: .ASCIZ \N2[S]ave filename - Saves the contents of the node table to a file\
1996 010601 HELP26: .ASCIZ \N2[U]nsave filename - restores node table from a file.\
1997 010673 HELP27: .ASCIZ \N2[L]isten [S]ource/adr/[D]estination/adr/[P]rotocol/protocol type\
1998 011001 HELP28: .ASCIZ \NIA - listens for frames that pass the specified filters.\
1999 011105 HELP29: .ASCIZ \NIA^{Notes:} 1) adr is the physical address of a node on the NI.\
2000 011207 HELP30: .ASCIZ 2) Pass count is a decimal number between 1 and 65534. A default\
2001 011326 HELP31: .ASCIZ value of 1 is assumed.\
2002 011376 HELP32: .ASCIZ Specifying -1 causes the test to be run indefinitely.\
2003 011505 HELP33: .ASCIZ 3) filename is an xxdp file.\
2004 .EVEN
2005
2006 011560 OPNERR: .ASCIZ /*NIA?Unable to Open "NI"??/
2007 011614 CLI\$PM: .ASCIZ <12><15>/NIE>/ ;NIE PROMPT
2008 011623 CLIERM: .ASCIZ /*NIA?ILL CMD-BAD SYNTAX?/
2009 011654 CLINUF: .ASCIZ /*NIA?INCOMPLETE COMMAND?/
2010 011705 CLINBG: .ASCIZ /*NIA?NUMBER TOO BIG?/
2011 011732 CLIBRX: .ASCIZ /*NIA?BAD RADIX?/
2012 011752 LINHLP: .ASCIZ /*T\$N/
2013 011757 LDRESP: .ASCIZ /*NIA NODE NI HAS RESPONDED./
2014 012014 RECERR: .ASCIZ /*NIA FRAME RECEIVED WITH DEUNA,DELUA ERROR./
2015 012067 RTRYER: .ASCIZ /*NIA TRANSMISSION ABORTED -- EXCESSIVE COLLISIONS./
2016 012151 BLDMSG: .ASCIZ /*NIA D2 Node addresses added, elapsed time: D2 minutes./
2017 012244 BLDDON: .ASCIZ /*NIA Build completed after D2 minutes./
2018 012316 ILADMS: .ASCII /*NIA Cannot use Broadcast address (FF-FF-FF-FF-FF-FF)/
2019 012402 ILADM1: .ASCIZ /*NIA for loop testing. Address was not added to node table.N/

			DEFAULT ADR	NAME	DECnet	DEVICE	NAME	DEVICE/
2020	012477	CADRER: .ASCIZ	/%N%APlease enter twelve hexadecimal digits./					
2021	012553	CPROER: .ASCIZ	/%N%APlease enter four hexadecimal digits./					
2022	012625	NULSTR: .ASCIZ	/%N%AA zero length string was entered./					
2023	012673	NODADR: .ASCIZ	/%N%T/					
2024	012700	DEFADR: .ASCIZ	/%S3%T/					
2025	012706	LOGNAM: .ASCIZ	/%S3%AN%04/					
2026	012720	NODTYP: .ASCIZ	/%S3%T/					
2027	012726	NETADR: .ASCIZ	/%S3%D2%A.%D3%S4/					
2028	012746	UNA: .ASCIZ	/%ADEUNA/					
2029	012756	QNA: .ASCIZ	/%ADEQNA/					
2030	012766	LUA: .ASCIZ	/%ADELUA/					
2031	012776	CNA: .ASCIZ	/%ADECNA/					
2032	013006	SCA: .ASCIZ	/%ADECSA/					
2033	013016	SRV: .ASCIZ	/%ADECserver/					
2034	013032	UNKNWN: .ASCIZ	/%A?????/					
2035	013042	NTBHDR: .ASCIZ	\%N%A CURRENT ADR	DEFAULT ADR	NAME	DECnet	DEVICE	%N\
2036	013152	DTBHDR: .ASCIZ	/ CURRENT ADR	DEFAULT ADR	NAME	DEVICE	/	
2037	013241	EMPSLT: .ASCIZ	/EMPTY SLOT/<015><012>					
2038	013256	SPACES: .ASCIZ	/ /					
2039	013265	LISHD1: .ASCIZ	\%N%A DESTINATION	SOURCE	PROT	TYPE	CHAR COUNT	\%N\
2040	013371	LISHD2: .ASCIZ	/%S3%A# OF RECEIPTS%N/					
2041	013416	NEWLI1: .ASCIZ	/%N/					
2042	013421	NEWLI2: .ASCIZ	<015><012>					
2043	013424	DADDR: .ASCIZ	/%N%T/					
2044	013431	SADDR: .ASCIZ	/%S3%T/					
2045	013437	PTYPE: .ASCIZ	/%S6%T/					
2046	013445	CHARAC: .ASCIZ	/%S6%D4/					
2047	013454	LCOUNT: .ASCIZ	/%S11%D6/					
2048	013464	LFMSG: .ASCIZ	/%N%AListen log was filled after %D2%A minutes %D2%A seconds%N/					
2049	013563	LEMSG: .ASCIZ	/%N%AListen log is empty!/%N%					
2050	013615	ALEMPT: .ASCIZ	/%N%AAddress list is empty, also./					
2051	013657	ALHDR: .ASCIZ	/%N%A SOURCE ADDRESS	COUNT%N/				
2052	013723	AADDR: .ASCIZ	/%N%T%\$4%D6/					
2053	013736	LTMSG: .ASCIZ	/%N%ATotal elapsed listen time: %Z2%A:%Z2%A. Listen commands: %D2/					
2054	014041	TABFUL: .ASCIZ	/%N%AThe %T%A table is filled to capacity!/%N%					
2055	014113	TABEMT: .ASCIZ	/%N%AThe %T%A table is currently empty!/%N%					
2056	014162	NOD: .ASCIZ	/NODE/					
2057	014167	SUMM: .ASCIZ	/SUMMARY/					
2058	014177	CLRMSG: .ASCIZ	/%N%AThe message parameters have been reset to:/					
2059	014256	CPYLMT: .ASCIZ	/%N%AThe number of copies must be between 1 and 255./					
2060	014342	SIZLMT: .ASCIZ	/%N%AThe message size [data] must be between 32 and 1466 bytes./					
2061	014441	NOCMPR: .ASCIZ	/%N%AThe address marked for deletion was not in the table./					
2062	014533	UNBOND: .ASCIZ	/%N%AAn unbounded "operator input" string was entered./					
2063	014621	ADRDEL: .ASCIZ	/%N%AThe address has been deleted from the node table./					
2064	014707	LOGDEL: .ASCIZ	/%N%ANode N%04%A has been deleted from the node table./					
2065	014775	NTBLOV: .ASCIZ	/%N%ANode table too small for all input - table truncated/					
2066	015066	TABCLR: .ASCIZ	/%N%AThe %T%A table has been cleared./					
2067	015133	UNSMMSG: .ASCIZ	/%N%AThe node table has been %T/					
2068	015172	SAVED: .ASCIZ	/SAVED./					
2069	015201	RESTOR: .ASCIZ	/RESTORED./					
2070	015213	MSGPRM: .ASCIZ	/%N%AThe current message parameters are:/					
2071	015263	MSG1: .ASCIZ	/%N%AThe collection of all node addresses could take as long as 40 minutes./					
2072	015376	MSG11: .ASCIZ	/%N%Ahower, if no new nodes are added to the table for a 10 minute period/					
2073	015511	MSG12: .ASCIZ	/%N%At the collection will stop.%N/					
2074	015551	MSG2: .ASCIZ	/%N%AYOU ENTERED NODE: %T/					
2075	015602	MSG3: .ASCIZ	/%N%ATHE SPECIFIED ADDRESS IS: %T/					
2076	015643	MSG4: .ASCIZ	/%N%ATYPE=%T%A,SIZE=%D4%A,COPIES=%D3/					

2077 .EVEN
2078 015710 HDMSG1: .ASCIZ /*N/A ETHERNET DEFAULT ADDRESS (HEX): #T/
2079 015761 HDMSG2: .ASCIZ /*N/A ROM MICROCODE VERSION (DECIMAL): #D3/
2080 016034 HDMSG3: .ASCIZ /*N/A SWITCH PACK SET FOR :/
2081 016070 HDMSG4: .ASCIZ /*N/A REMOTE AND POWER UP BOOT ENABLED/
2082 016145 HDMSG5: .ASCIZ /*N/A REMOTE BOOT ENABLED WITH ROM/
2083 016216 HDMSG6: .ASCIZ /*N/A REMOTE BOOT ENABLED/
2084 016256 HDMSG7: .ASCIZ /*N/A REMOTE BOOT DISABLED/
2085 016317 HDMSG8: .ASCIZ /*N/A SELF TEST LOOP ENABLED/
2086 016362 HDMSG9: .ASCIZ /*N/A SELF TEST LOOP DISABLED/
2087 .EVEN
2088 :
2089 : TEST MESSAGES AND ARGUMENTS
2090 :
2091 :
2092 016426 PASABT: .ASCIZ /*N/A PASS ABORTED!/
2093 016451 TSTMS1: .ASCIZ /*N/T/A TEST -- /
2094 016471 TSTMS2: .ASCIZ /*N/T/A Node: %AN%04%N/
2095 016517 TSTMS3: .ASCIZ /*T/A ERROR/
2096 016532 TSTMS4: .ASCIZ /*N/T/A Node: %AN%04%N %T/A Node: %AN%04/
2097 016602 OK: .ASCIZ /*A - Response ok%N/
2098 016625 OKRE: .ASCIZ /*N/A - Receive assist - response ok%N/
2099 016673 OKTR: .ASCIZ /*N/A - Transmit assist - Response ok%N/
2100 016742 OKFU: .ASCIZ /*N/A - Full assist - Response ok%N/
2101 017005 MESPAT: .ASCIZ /*N/AERROR OCCURED WITH %T/A MESSAGE TYPE/
2102 017056 MESPA1: .ASCIZ /*A Data Pattern: #T/
2103 017102 ALLNOD: .ASCIZ /ALL NODE/
2104 017113 LUPAIR: .ASCIZ /LOOPPAIR/
2105 017124 DIRECT: .ASCIZ /LOOP DIRECT/
2106 017140 FULAST: .ASCIZ /FULL ASSIST/
2107 017154 TRAST: .ASCIZ /TRANSMIT ASSIST/
2108 017174 RECAST: .ASCIZ /RECEIVE ASSIST/
2109 017213 PATTRN: .ASCIZ /MESSAGE PATTERN/
2110 017233 NORESP: .ASCIZ /NO RESPONSE/
2111 017247 RETRY: .ASCIZ /EXCESSIVE COLLISION/
2112 017273 LENGTH: .ASCIZ /LENGTH/
2113 017302 COMPAR: .ASCIZ /DATA COMPARISON/
2114 .EVEN
2115 :
2116 017322 MSGTY0: .ASCIZ /ASCII/ :MESSAGE TYPES
2117 017330 MSGTY1: .ASCIZ /ONES/
2118 017335 MSGTY2: .ASCIZ /ZEROS/
2119 017343 MSGTY3: .ASCIZ /1ALT/
2120 017350 MSGTY4: .ASCIZ /0ALT/
2121 017355 MSGTY5: .ASCIZ /CCITT/
2122 017363 MSGTY6: .ASCIZ /TEXT/
2123 017370 CMDTY1: .ASCIZ /EXIT/
2124 017375 CMDTY2: .ASCIZ /SUMMARY/
2125 017405 CMDTY3: .ASCIZ /BUILD/
2126 017413 CMDTY4: .ASCIZ /SHOW/
2127 017420 CMDTY5: .ASCIZ /RUN/
2128 017424 CMDTY6: .ASCIZ /MESSAGE/
2129 017434 CMDTY7: .ASCIZ /NODE/
2130 017441 CMDTY8: .ASCIZ /CLEAR/
2131 017447 CMDTY9: .ASCIZ /REQUEST ID/
2132 017462 ARGTY1: .ASCIZ /NODES/
2133 017470 ARGTY2: .ASCIZ /MESSAGES/ :ARGUMENT TYPES

```
2134 017501 ARGTY3: .ASCIZ /COUNTERS/
2135 017512 ARGTY4: .ASCIZ /LOOPPAIR/
2136 017523 ARGTYS: .ASCIZ /ALL/
2137 017527 ARGTY6: .ASCIZ /Assist/
2138 017536 ARGTY7: .ASCIZ /Target/
2139 .EVEN
2140
2141 :
2142 : UNA COUNTER INFORMATION MESSAGES
2143 :
2144
2145 017546 cntr00: .asciz /*NS5A CONTENTS OF NODE #T#A INTERNAL COUNTERS:/
2146 017626 cntr01: .asciz /*NS2A SECONDS SINCE LAST ZEROED:#S15#Z5/
2147 017675 cntr02: .asciz /*NSA FRAMES RECEIVED:#S20#T/
2148 017730 cntr03: .asciz /*NSA MULTICAST FRAMES RECEIVED:#S10#T/
2149 017775 cntr04: .asciz /*NSA FRAMES REC'D WITH ERROR - BITMAP:#S10#B3/
2150 020052 cntr05: .asciz /*NSA FRAMES RECEIVED WITH ERROR:#S14#Z5/
2151 020121 cntr06: .asciz /*NSA DATA BYTES RECEIVED:#S16#T/
2152 020160 cntr07: .asciz /*NSA MULTICAST DATA BYTES RECEIVED:#S6#T/
2153 020230 cntr08: .asciz /*NSA RECEIVED FRAMES LOST-INTERNAL:#S11#Z5/
2154 020302 cntr09: .asciz /*NSA RECEIVED FRAMES LOST -LOCAL:#S13#Z5/
2155 020352 cntr10: .asciz /*NSA FRAMES TRANSMITTED:#S17#T/
2156 020410 cntr11: .asciz /*NSA MULTICAST FRAMES TRANSMITTED:#S7#T/
2157 020457 cntr12: .asciz /*NSA FRAMES TRANSMITTED 3+ TRYs:#S9#T/
2158 020524 cntr13: .asciz /*NSA FRAMES TRANSMITTED 2 TRYs:#S10#T/
2159 020571 cntr14: .asciz /*NSA FRAMES DEFERRED:#S20#T/
2160 020624 cntr15: .asciz /*NSA DATA BYTES TRANSMITTED:#S13#T/
2161 020666 cntr16: .asciz /*NSA MULTICAST BYTES TRANSMITTED:#S8#T/
2162 020734 cntr17: .asciz /*NSA TRANSMIT FRAMES ABORTED-BITMAP:#S9#B6/
2163 021006 cntr18: .asciz /*NSA TRANSMIT FRAMES ABORTED:#S17#Z5/
2164 021052 cntr19: .asciz /*NSA XMIT COLLISION CHECK FAILURE:#S12#Z5/
2165 021123 cntr20: .asciz /*NSA PORT DRIVER ERRORS:#S22#Z5/
2166 021162 cntr21: .asciz /*NSA BABBLE COUNTER:#S26#Z5/
2167
2168 :
2169 : ERROR MESSAGES FOR DEUNA/DELUA DRIVER
2170 :
2171
2172 021215 emsg01: .asciz /DELUA,DEUNA PORT COMMAND ERROR/
2173 021254 emsg02: .asciz /DELUA,DEUNA FATAL ERROR/
2174 021304 emsg03: .asciz /UNEXPLAINED DELUA,DEUNA INTERRUPT/
2175 021346 emsg04: .asciz /UNKNOWN DELUA,DEUNA ERROR/
2176 021400 emsg05: .asciz /DELUA,DEUNA WON'T READ PCB ADDRESS/
2177 021443 emsg06: .asciz /UNABLE TO READ PHYSICAL ADDRESS/
2178 021503 emsg07: .asciz /DELUA,DEUNA WILL NOT GO INTO RUNNING STATE/
2179 021556 emsg08: .asciz /TIMEOUT!--TRANSMIT FLAG NOT SET/
2180 021616 emsg09: .asciz /PDMD PORT COMMAND ERROR/
2181 021646 emsg10: .asciz /TRANSMIT RING BOOKKEEPING ERROR/
2182 021706 emsg14: .asciz /MESSAGE SIZE TOO BIG FOR MAX. FRAME LENGTH/
2183 021761 emsg15: .asciz /DNI DID NOT SET FROM RESET/
2184 022014 emsg16: .asciz /DELUA,DEUNA WILL NOT READ DESCRIPTOR RINGS/
2185 022067 emsg18: .asciz /CAN'T GET INITIAL STATUS INFO FROM DELUA,DEUNA/
2186 022146 emsg19: .asciz /MESSAGE DATA COMPARISON ERROR/
2187 022204 emsg20: .asciz /TOTAL DATA COMPARE ERRORS/
2188 022236 emsg22: .asciz /NO RESPONSE FROM NODE./
2189 022265 emsg23: .asciz /ERROR WHILE ATTEMPTING TO WRITE MODE/
2190 022332 emsg24: .asciz /TRANSMIT ERROR, ALL FRAMES NOT TRANSMITTED/
```

2191 022405 emsg25: .asciz /ERROR WHILE ATTEMPTING TO WRITE MULTICAST ADDRESS LIST/
2192 022474 emsg26: .asciz /TRANSMIT LOOP DIRECT FAILED/
2193 022530 emsg30: .asciz /ERROR WHILE ATTEMPTING PORT FUNCTION/
2194 022575 emsg31: .asciz /UNABLE TO READ INTERNAL COUNTERS/
2195 022636 emsg33: .asciz /TIMEOUT ERROR/
2196 022654 emsg34: .asciz <15><12>/TIMEOUT OCCURED BEFORE LOOPBACK REPLY/
2197 022724 emsg35: .asciz /*AFAILING NODE ADDRESS: #T#N/
2198 022761 emsg36: .asciz /*ADATA PATTERN: #T#N/
2199 023006 EMSG37: .ASCIZ /*AFAILING TARGET NODE ADDRESS: #T#N/
2200 023052 EMSG38: .ASCIZ /*AFAILING ASSIST NODE ADDRESS: #T#N/
2201 023116 EMSG41: .ASCIZ <15><12>/TIMEOUT OCCURED - TRANSMIT FAILED/
2202 023152 EMSG42: .ASCIZ <15><12>/TIMEOUT OCCURED - RECEIVE FAILED/
2203 023225 EMSG43: .ASCIZ /DELUA,DEUNA RAN OUT OF RECEIVE BUFFERS/
2204 023274 EMSG44: .ASCIZ /ERROR CONVERTING HEX TEXT TO BINARY/
2205 023340 EMSG45: .ASCIZ /*NNXATOO MUCH DATA FOR BOUNCE/
2206 023375 EMSG46: .ASCIZ /*NNXANO ADDRESS FOR LOGICAL NODE NAME/
2207 023442 EMSG47: .ASCIZ /DELUA,DEUNA WOULD NOT ENTER READY STATE/
2208 023512 EMSG48: .ASCIZ <15><12>/LOOP DIRECT FAILED/
2209 023537 EMSG49: .ASCIZ /TRANSMIT FAILED AFTER THREE ATTEMPTS -- ETHERNET EXTREMELY LOADED/
2210 023641 EMSG50: .ASCIZ /FATAL DEVICE ERROR WHILE ATTEMPTING TRANSMIT/
2211 023716 EMSG51: .ASCIZ /BAD CLOCK - PROGRAM WILL HANG ON "TIMEOUT"!!!
2212 023773 EMSG52: .ASCIZ /CAN'T READ DEVICE'S PHYSICAL ADDRESS/
2213 024040 EMSG53: .ASCIZ /CAN'T READ ROM VERSION NUMBER/
2214 024076 EMSG54: .ASCIZ /STACK OVERFLOW ERROR - CRASH!/
2215 .even
2216
2217 ;---+
2218 ; Descriptions of generic fields of system ID messages
2219 ;---+
2220 024134 simsg1: .asciz /*NNXACURRENT HARDWARE ADDRESS: #T/
2221 024206 simsg2: .asciz /*NNXARecipient number: #06/
2222 024261 simsg3: .asciz /*NNXAMaintenance version: #22/
2223 024334 simsg4: .asciz /*NNXAEC0: #22/
2224 024407 simsg5: .asciz /*NNXAUser ECO: #22/
2225 024462 simsg6: .asciz /*NNXAFunction: #02/
2226 024535 simsg7: .asciz /*NNXADevice: /
2227 024605 simsg8: .asciz /*NNXAConsole User Address: #T/
2228 024657 simsg9: .asciz /*NNXAReservation Timer: #06/
2229 024732 smsg10: .asciz /*NNXAConsole Command Size: #06/
2230 025005 smsg11: .asciz /*NNXAConsole Response Size: #06/
2231 025060 smsg12: .asciz /*NNXAFAULT HARDWARE ADDRESS: #T/
2232 025132 smsg13: .asciz /*NNXASystem Time: #06#06#06#06#06/
2233
2234
2235 ;---+
2236 ; Poseidon Specific fields of a system ID message
2237 ;---+
2238 025221 posds: .asciz /*NN2ADiagnostic Status/
2239 025250 posds0: .asciz /*NNAA WORD 0: #06#A(0)/
2240 025330 posds1: .asciz /*NNAA WORD 1: #06#A(0)/
2241 025410 possn: .asciz /*NNAServer Number: #06#A(0)/
2242 025470 posrvn: .asciz /*NNARom Version Number: /
2243 025540 possvn: .asciz /*NNASoftware Version Number: /
2244 025610 posnam: .asciz /*NNAServer Name: /
2245 025660 posloc: .asciz /*NNAServer Location: /
2246 025730 posstr: .asciz /*T/
2247

2248 .even
2249 025734 PCMSG:: .asciz /*N%APC OF CALLING ROUTINE = #06/
2250 .even
2251 025774 cmperh: .asciz /*N%COMPARE ERRORS IN LOOP MESSAGE%N2/
2252 026042 cmper1: .asciz /*N%AWord number:#D4%A(D) Expected=#06%A(0)/
2253 026115 cmper2: .asciz /*N%A Received=#06%A(0)/
2254 026142 cmper3: .asciz /*N%ATotal mismatches in message = #D4/
2255 026210 lgerms: .asciz /*N%ALength Error -- Bytes Expected: #06%A Bytes Received: #06/
2256 026306 summe1: .asciz /*N%A NODE RECEIVES RECEIVES NOT LENGTH COMPARE BYTES
2257 026426 summe2: .asciz /*N%A ADDRESS COMPLETE COMPLETE ERRORS ERRORS COMPARED
2258 026553 summe3: .asciz /*N%T#S2#Z5#S7#Z5#S5#Z5/
2259 026602 summe5: .asciz /*S2#Z5#S2#T/
2260 026616 summe6: .asciz /*S2#T/
2261 .even
2262 .list bin :::::
2263

BYTES/
TRANSFERRED%N/

```
2265          .SBTTL GLOBAL ERROR REPORT SECTION
2266
2267          ;+
2268          : THE GLOBAL ERROR REPORT SECTION CONTAINS MESSAGE PRINTING AREAS
2269          : USED BY MORE THAN TEST TO OUTPUT ADDITIONAL ERROR INFORMATION. PRINTB
2270          : (BASIC) AND PRINTX (EXTENDED) CALLS ARE USED TO CALL PRINT SERVICES.
2271          ;-
2272
2273
2274 026624      BGNMSG  ERR1
2275 026624      PRINTX  #PCMSG,PCCALL
2276 026650      DOCLN
2277 026652      ENDMSG
2278
2279 026654      BGNMSG  ERR2
2280 026654 010146    MOV     R1,-(SP)
2281 026656 013701 001170'   MOV     P$TYPE,R1
2282 026662 006301    ASL     R1
2283 026664 062701 001414'   ADD     #MSGTAB,R1
2284 026670          PRINTX  #EMSG35,#STRBUF
2285 026714          PRINTX  #EMSG36,(R1)
2286 026736 012601      MOV     (SP)+,R1
2287 026740      ENDMSG
2288
2289 026742      BGNMSG  ERR3
2290 026742      PRINTX  #EMSG37,#STRBUF
2291 026766      PRINTX  #EMSG38,#STRBU1
2292 027012      ENDMSG
2293
2294          ;*****
2295          : THESE MESSAGE AREAS ARE USED TO OUTPUT SUPPLEMENTARY INFORMATION
2296          : AFTER AN ERROR CALL. THEY ARE INVOKED BY APPENDING THE NAME
2297          : OF THE AREA TO AN ERROR CALL: ERRXXX 1,ERRORMESSAGE,AREANAME.
2298          : THE CORRESPONDING MESSAGE AREA IS SET UP IN THIS SECTION:
2299          : BGNMSG AREANAME
2300          : [CODE]
2301          : ENDMSG
2302
2303
2304          : THE AREAS IN THIS SECTION ARE FOR MESSAGES USED IN MORE THAN ONE
2305          : TEST. USE THE PRINTB (PRINT BASIC) AND PRINTX (PRINT EXTENDED)
2306          : MACROS.
2307
2308
2309
2310
```

```
2312 .SBTTL GLOBAL SUBROUTINES SECTION
2313
2314
2315 : THE GLOBAL SUBROUTINES SECTION CONTAINS THE SUBROUTINES
2316 : THAT ARE USED IN MORE THAN ONE TEST.
2317 :--
2318
2319
2320 : FUNCTIONAL DESCRIPTION:
2321 :   SUBROUTINE TO....
2322
2323 :-----+
2324 :      COMPLETE THE "SUBROUTINE TO...." STATEMENT WITH A FUNCTIONAL
2325 :      DESCRIPTION OF THIS SUBROUTINE.
2326 :-----+
2327
2328
2329 : INPUTS:
2330
2331 :-----+
2332 :      LIST THE INPUT DATA THAT ARE EXPLICITLY PASSED TO THIS SUBROUTINE.
2333 :-----+
2334
2335 : IMPLICIT INPUTS:
2336
2337 :-----+
2338 :      LIST THE INPUT DATA THAT ARE IMPLICITLY USED BY THIS SUBROUTINE;
2339 :      FOR EXAMPLE, DATA READ FROM COMMON AREAS.
2340 :-----+
2341
2342 : OUTPUTS:
2343
2344 :-----+
2345 :      LIST THE OUTPUT DATA THAT ARE EXPLICITLY GIVEN BY THIS SUBROUTINE
2346 :-----+
2347
2348 : IMPLICIT OUTPUTS:
2349
2350 :-----+
2351 :      LIST THE OUTPUT DATA THAT ARE IMPLICITLY GIVEN BY THIS SUBROUTINE;
2352 :      FOR EXAMPLE, DATA STORED IN COMMON AREAS.
2353 :-----+
2354
2355 : SUBORDINATE ROUTINES USED:
2356
2357 :-----+
2358 :      LIST THE SUBROUTINES CALLED BY THIS SUBROUTINE.
2359 :-----+
2360
2361 : FUNCTIONAL SIDE EFFECTS:
2362
2363 :-----+
2364 :      DESCRIBE ANY EFFECTS THIS SUBROUTINE MAY HAVE UPON OTHER
2365 :      MODULES OF THE DIAGNOSTIC PROGRAM. AN EXAMPLE OF THIS IS
2366 :      THE SUBROUTINE INHIBITS ALL INTERRUPTS WITH PRIORITY 7.
2367 :-----+
2368
2369 : CALLING SEQUENCE:
```

```
2383
2385
2386      ; GIVE THE EXACT CALLING SEQUENCE USED TO ACCESS THIS SUBROUTINE.
2387      ; FOR EXAMPLE:    MOV COUNT,R1    ;MOVE INPUT TO R1
2388          JSR     PC,ROUTINE   ;GO TO ROUTINE
2389          BCS     ERROR       ;CARRY SET IF ROUTINE HAD ERROR
2390
2392      ;-
2393
2395      ; INSERT THE CODE FOR THIS SUBROUTINE.  THE NAME OF THE SUBROUTINE SHOULD
2396      ; BE DEFINED WITH A DOUBLE-COLON (::); THIS WILL MAKE THE SUBROUTINE GLOBAL.
2398
2400
2402      ; BEGIN EACH SUBROUTINE AT THE TOP OF A NEW PAGE.
2403
2404
2406
2407      .SBTTL CLKSET Clock Setup Subroutine
2408
2409      ;--+
2410      ; Functional Description:
2411          This subroutine sets up the clock information table following
2412          a "CLOCK" call executed in the initialization code. But since
2413          the "CLOCK" call says nothing about an LSI-11's clock, the
2414          routine is only used if a line or P-Clock is found.
2415
2416      ; Inputs - Implicit -
2417          R1 - Points to supervisor space where clock info was returned
2418          R2 - Points to "CLK" table where clock info will be kept
2419
2420      ; Outputs - Implicit -
2421          "CLKCSR" gets loaded with the clock's CSR address
2422          "CLKBR" gets loaded with the clock's interrupt level
2423          "CLKVEC" gets loaded with the clock's interrupt vector
2424          "CLKHZ" gets loaded with the line freq. (in Hertz)
2425
2426      ; Calling Procedure: JSR      PC,CLKSET
2427
2428      ; Side effects - none
2429
2430      ; Subordinate Routines - none
2431
2432      ; Register Usage
2433          R1 - Points to supervisor space where clock info was returned
2434          R2 - Points to "CLK" table where clock info will be kept
2435
2436      ;--+
2437 027014      CLKSET:::
2438 027014 012122      mov    (R1)+,(R2)+      ; Load clock's CSR addr. into "CLKCSR"
2439 027016 012112      mov    (R1)+,(R2)      ; Load clock's intr. level into "CLKBR"
2440 027020 006312      asl    (R2)           ; Adjust the intr. level for loading
2441 027022 006312      asl    (R2)           ;   into the PSW with a "SETVEC" call
2442 027024 006312      asl    (R2)
2443 027026 006312      asl    (R2)
2444 027030 006322      asl    (R2)+          ; Load clock's intr. vector into "CLKVEC"
2445 027032 012122      mov    (R1)+,(R2)+
```

2446 027034 012122
2447 027036 000207
2448

mov (R1)+,(R2)+ ; Load clock's freq. into "CLKHZ"
rts PC

```

2450
2451 .sbttl CLKINT Clock Interrupt Service Routine
2452
2453 :--+
2454 : Functional Description:
2455 : This is the clock interrupt service routine which takes care
2456 : of keeping the "time-since-start" and counting down any of the
2457 : "event" timers. The timers are used to time completion of
2458 : device requests. The "time-since-start" is used to be logged
2459 : with each entry into the event log.
2460
2461 : Inputs - Implicit -
2462 :     TIMTCK - The current no. of ticks left to be counted until
2463 :             a second has been counted off
2464 :     CLKHZ - The no. of ticks in a second, determined by the
2465 :             sys. line freq.
2466 :     TIMMIN & TIMSEC - Current value of "time-since-start" in
2467 :             minutes and seconds
2468 :     TIMER 1,2 and S - Current values of "event timers"
2469
2470 : Outputs - Implicit -
2471 :     New value of event timer "1" & "2" decremented by 1 tick
2472 :     if it was non-zero
2473 :     New value of event timer "S" decremented by 1 second if it
2474 :     was non-zero
2475
2476 : Calling procedure : This routine is entered upon clock interrupt
2477
2478 : Side effects -
2479 :     The clock is disabled upon entry and reenabled when leaving
2480
2481 : Subordinate Routines - none
2482
2483 : Register Usage - none
2484
2485
2486 :--+
2487 027040          BGNSRV CLKINT
2488
2489 027040 005077 152762      clr    $CLKCSR           ; disable the clock from interrupting
2490 027044 005337 002044'     dec    TIMTCK           ; decrement the no. of ticks/sec
2491 027050 001015            bne    1$                ; go check timers
2492 027052 013737 002034' 002044'   mov    CLKHZ,TIMTCK ; reset the no. of ticks/sec.
2493 027060 005237 002042'     inc    TIMSEC           ; inc. no of secs-since-start
2494 027064 022737 000074 002042'   cmp    #60.,TIMSEC ; see if we've counted 60 sec.s yet
2495 027072 001004            bne    1$                ; if not, go check timers
2496 027074 005237 002040'     inc    TIMMIN           ; else, inc. minutes-since-start
2497 027100 005037 002042'     clr    TIMSEC           ; and restart second counter
2498
2499 027104 005737 002046'     1$:   tst    TIMER1           ; see if TIMER1 timing anything
2500 027110 001402            beq    2$                ; if=0, no, check next timer
2501 027112 005337 002046'     dec    TIMER1           ; else decrement the timer value (by 1 tick)
2502 027116 005737 002050'     2$:   tst    TIMER2           ; see if TIMER2 timing anything
2503 027122 001402            beq    3$                ; if=0, no, check next timer
2504 027124 005337 002050'     dec    TIMER2           ; else decrement timer value (by 1 tick)
2505 027130 005737 002052'     3$:   tst    TIMERS           ; see if TIMERS timing anything
2506 027134 001406            beq    4$                ; if=0, nothing be timed, leave

```

```
2507 027136 023737 002034' 002044'      cmp    CLKHZ,TIMTCK      ; see if a second has been counted off
2508 027144 001002                      bne    4$                  ; br if no
2509 027146 005337 002052'                dec    TIMERS            ; else, decrement timer value (by 1 sec.)
2510 027152 013777 002036' 152646 4$:     mov    CLKEN,BCLKCSR      ; reenable the clock to interrupt
2511 027160                      ENDSRV

2512
2513
2514 .SBTTL PREG14 Preserve Registers 1 through 4 across subroutine calls
2515 :--+
2516 : Functional Description:
2517 : This routine is a relocatable module designed to preserve
2518 : registers 1 through 4 across subroutine calls. It saves
2519 : these registers and then does a JSR to the routine specified
2520 : in the "CALL".
2521 :
2522 : Inputs - Implicit
2523 : The address of the routine to "CALL" relative to the "ANCHOR"
2524 : label is located in the word following the JSR to this routine.
2525 :
2526 : Outputs - None
2527 :
2528 : Calling Procedure: This routine is used implicitly by the "CALL" macro.
2529 : The macro expands to the following:
2530 :
2531 :          JSR    R4,PREG14
2532 :          .WORD [subroutine name]-ANCHOR
2533 :
2534 : Side effects - None
2535 :
2536 : Subordinate Routines -
2537 : The routine specified in the "CALL" macro is called.
2538 :
2539 : Register Usage -
2540 : R1 - used to form the absolute address of the call
2541 : R4 - link register in call to this routine
2542 : SP - registers 1 through 4 are saved on the stack
2543 :
2544 :--+
2545
2546 027162 010346
2547 027162 010346      MOV    R3,-(SP)      ;Push R3, R2, R1
2548 027164 010246      MOV    R2,-(SP)      ;
2549 027166 010146      MOV    R1,-(SP)      ;
2550
2551 027170 010437 003124'      MOV    R4, PCCALL
2552 027174 012401      MOV    (R4)+,R1      ;Get the relative address of the called
2553 :routine.
2554 027176 060701      ADD    PC,R1       ;Make it an absolute address.
2555
2556 027200 010446      ANCHOR: MOV    R4,-(SP)      ;Save the return to the calling routine.
2557
2558 027202 022706 001000      CMP    #1000,SP      ; Don't allow the stack to crush ...
2559 : ... floating vector space
2560 027206 103404      BLO    ERRSF      1$,EMSG54,ERR1      ; print stack overflow error ... and depart!
2561 027210
2562
2563 027220 004711      1$:    JSR    PC,(R1)      ;Call the specified routine.
```

2564
2565 027222 012604 MOV (SP)+,R4 ;Restore the return to the calling routine.
2566
2567 027224 012601 MOV (SP)+,R1 ;Restore the registers.
2568 027226 012602 MOV (SP)+,R2 ;
2569 027230 012603 MOV (SP)+,R3 ;
2570 027232 000204 RTS R4 ;Back to the calling routine.
2571
2572

```

2574
2575          .sbttl WAIT      Wait For DEUNA/DELUA Interrupt with Timeout
2576
2577
2578          ;+
2579          ; Functional Description:
2580          ; This routine is called to wait for the Done Interrupt bit (DNI)
2581          ; of PCSR0 to be set signifying the completion of a port command.
2582          ; If the DEUNA/DELUA reports some sort of error, ERRFLG will
2583          ; have been raised in the interrupt service routine. In this
2584          ; case the error reporting routine will be called.
2585          ; Inputs - none
2586
2587          ; Outputs -
2588          ;           P1: success/failure      0=success/-1=failure
2589
2590          ; Calling Procedure:
2591          ;           call    wait
2592          ;           p$pop   p1
2593
2594          ; Side effects - none
2595
2596          ; Subordinate routines -
2597          ;           ERROR - error reporting routine
2598
2599          ; Register Usage -
2600          ;           R2 - used to hold return status
2601          ;           R4 - address of word that contains timer value
2602
2603
2604 027234 012703 000012          WAIT::  mov    #10.,R3      ; move no. of counts to R3
2605 027240 012704 002046'        mov    #timer1,R4    ; and timer to be used to R4
2606 027244 005002          clr    r2      ;local STATUS parameter
2607 027246 010314          mov    r3,(r4)    ;set number of ticks. (global)
2608 027250 005737 003020'        1$:   tst    errflg     ;check if error occurred
2609 027254 001011          bne    3$      ; br if yes
2610 027256 005737 003012'        tst    dniflg     ;check for dni interrupt
2611 027262 001403          beq    2$      ; br if interrupt received
2612 027264 005037 003012'        clr    dniflg
2613 027270 000410          br    6$      ;has timer expired?
2614 027272 005714          2$:   tst    (r4)      ; br if no to wait for interrupt
2615 027274 001365          bne    1$      ;br to 5$
2616 027276 000403          br    5$      ;call error routine
2617 027300          3$:   call   ERROR     ;indicate failure
2618 027306 012702 177777        5$:   mov    #-1,r2    ;return with success/failure indication
2619 027312          6$:   return r2
2620
2621          .sbttl ERROR      Handle UNA interrupt errors
2622
2623
2624          ;+
2625          ; Functional Description:
2626          ; This subroutine checks the error flags set by
2627          ; UNAISR the interrupt service routine and prints
2628          ; out the appropriate error messages.
2629
2630          ; Inputs - implicit -
2631          ;           error flags should be set by UNAISR routines.

```

```
2631          : Outputs - implicit -
2632          :           error messages are printed out to the operator console.
2633          :
2634          : calling sequence:
2635          :           call ERROR
2636          :
2637          : Side effects -
2638          :           1.) error flags that were set in UNAISR are cleared here.
2639          :           2.) errors will be reported at the user's terminal
2640          :           3.) the diagnostic will be exited
2641          :
2642          : Subordinate routines -
2643          :           ERR1 - extended error report
2644          :
2645          :---+
2646
2647 027316 005337 003020'      ERROR:: dec    errflg      ;decrement error counter to show
2648                      ;that it has been handled
2649 027322 005737 003004'      tst     pceflg      ;see if port command error
2650 027326 001016 003002'      bne     5$        ; if yes, branch
2651 027330 005737 003002'      tst     fatflg      ;see if UNA fatal error
2652 027334 001022 003016'      bne     10$       ; if yes, branch
2653 027336 005737 003016'      tst     bcount      ;see if unexplained interrupt
2654 027342 001026 003014'      bne     15$       ; if yes, branch
2655 027344 005737 003014'      tst     rbfcnt      ; receive buffers unavailable?
2656 027350 001032            bne     18$       ; branch if yes
2657 027352            errdf   2,emsg04,err1  ;else unknown error
2658 027362 000433            br      20$       ; exit
2659
2660 027364 005337 003004'      5$:    dec     pceflg      ; indicate that it was handled
2661 027370            errdf   3,emsg01,err1  ;port command error
2662 027400 000424            br      20$       ; exit
2663
2664 027402 005337 003002'      10$:   dec     fatflg      ; keep up on book keeping
2665 027406            errdf   4,emsg02,err1  ;UNA fatal error
2666 027416 000415            br      20$       ; exit
2667
2668 027420 005337 003016'      15$:   dec     bcount      ; book keeping
2669 027424            errdf   5,emsg03,err1  ;unexplained interrupt
2670 027434 000406            br      20$       ; exit
2671
2672 027436 005337 003014'      18$:   dec     rbfcnt      ; report it
2673 027442            errdf   6,emsg43,err1  ;return
2674
2675 027452            20$:   return      ;return
2676
2677          :---+
2678          : Name - DEVSTART          Start the DELUA/DEUNA
2679          :
2680          : Functional Description:
2681          :           This routine is called to start up the DELUA/DEUNA.
2682          :           The transmit and receive rings will be reset with their
2683          :           associated pointers reset to the beginnings of their
2684          :           respective rings. This is done because when given a
2685          :           start port command, the DELUA or DEUNA will reset its
2686          :           pointers to the host rings.
2687          :           After resetting the rings, a START port command
```

```
2688          ; will be issued, causing the DELUA/DEUNA to transition to
2689          ; the running state.
2690          ;
2691          ; Inputs - none
2692          ;
2693          ; Outputs - none
2694          ;
2695          ; Calling Procedure: CALL DEVSTART
2696          ;
2697          ; Side Effects -
2698          ;      1.) transmit and receive rings are reset, and
2699          ;      2.) the DELUA/DEUNA is in the running state
2700          ;
2701          ; Subordinate Routines - none
2702          ;
2703          ; Register Usage -
2704          ;      R1 - pointer to transmit and receive rings
2705          ;      R2 - scratch
2706          ;
2707          ;---+
2708 027454    DEVSTART:
2709          ;---+
2710          ;      Reset transmit and receive ring pointers
2711          ;---+
2712 027454 013737 002066' 002072'    mov    XRGSR, XRGCUR           ; point them ...
2713 027462 013737 002066' 002076'    mov    XRGSR, XRGNXT           ; ... all to the ...
2714 027470 013737 002070' 002074'    mov    RRGSRT, RRGCUR           ; ... beginning of their ...
2715 027476 013737 002070' 002100'    mov    RRGSRT, RRGNXT           ; ... associated rings.
2716          ;
2717          ;---+
2718          ;      Clear the ownership bit of all entries in the transmit ring. This
2719          ;      will make us the owner of all entries.
2720          ;---+
2721 027504    CALL    REMAP #OTRING          ; enable access to transmit ring
2722 027516 012702 000004    mov    #NO.NTR,R2           ; R2 is loop control
2723 027522 013701 002072' 10$:    mov    XRGCUR, R1           ; point R1 to transmit ring
2724 027526 042761 100000 000004    bic    #own,4(R1)          ; we own all entries
2725 027534    CALL    GETXNX #XRGCUR          ; point to next entry
2726 027546 005302    dec    R2               ; do for all ring entries
2727 027550 001364    bne    10$              ;
2728          ;
2729          ;---+
2730          ;      Give ownership of all receive ring entries to the DELUA/DEUNA by
2731          ;      setting each entry's OWN bit.
2732          ;---+
2733 027552    CALL    REMAP #ORRING          ; enable access to receive ring
2734 027564 012702 000010    mov    #NO.NRR,R2           ; R2 is loop control
2735 027570 013701 002074' 20$:    mov    RRGCUR, R1           ; point R1 to receive ring
2736 027574 052761 100000 000004    bis    #own,4(R1)          ; DELUA/DEUNA owns all entries
2737 027602    CALL    GETRNX #RRGCUR          ; point to next entry
2738 027614 005302    dec    R2               ; do for all ring entries
2739 027616 001364    bne    20$              ;
2740          ;
2741          ;---+
2742          ;      Now put the device in the running state by issuing a START port
2743          ;      command.
2744          ;---+
```

ERROR Handle UNA interrupt errors

```

2745 027620          call    comand #strt           : put una in running state
2746 027632          P$POP  r2                   : check for error
2747 027634 001404   beq    30$                 : if OK, continue
2748 027636          errdf  7,emsg07,err1       : else report error
2749
2750 027646          30$:   CALL    RETMEM          : restore memory mapping
2751 027654          RETURN                         : leave ...
2752
2753
2754
2755          ;--+
2756          ; Name - STOP                         Stop the DELUA/DEUNA
2757
2758          ; Functional Description:
2759          ; This routine is called to stop the DELUA/DEUNA and
2760          ; leave it in the ready state.
2761
2762          ; Inputs - none
2763
2764          ; Outputs - none
2765          ; Calling Procedure: CALL DEVSTOP
2766
2767          ; Side Effects -
2768          ; 1.) The DELUA/DEUNA will be left in the ready state
2769
2770          ; Subordinate Routines - none
2771
2772          ; Register Usage -
2773          ;      R1 - return status of STOP port command
2774
2775
2776 027656          ;--+
2777 027656          DEVSTOP:                      : Issue the STOP port command
2778 027670          CALL    COMAND #STOP          : get return status
2779 027672 001404   P$POP  R1                   : leave if okay
2780 027674          BEQ    10$                 : indicate error ... and exit
2781
2782 027704          10$:   RETURN                         : return to caller
2783
2784
2785
2786          .sbttl UNAINI Initialize the UNA
2787
2788
2789
2790          ;--+
2791          ; Functional Description:
2792          ; The purpose of this routine is to initialize and startup
2793          ; the DELUA/DEUNA. The initialization of the DELUA/DEUNA is
2794          ; as follows:
2795
2796          ; 1.) Issue a GET PCBB port command to tell the device where
2797          ;     the port control block is located in host memory.
2798
2799          ; 2.) Issue a write ring descriptor port command to tell the
2800          ;     device where the receive and transmit rings are located
2801          ;     in host memory.

```

```

2802          : The device is then started by issuing a START port command.
2803          : Then the devices physical address is read and stored.
2804          :
2805          : Inputs - none
2806          :
2807          : Outputs - none
2808          :
2809          : Calling Procedure: CALL UNAINI
2810          :
2811          : Side effects -
2812          :           PHYADR - contains the device's default physical address
2813          :
2814          : Subordinate Routines -
2815          :           COMAND - subroutine to issue a port command
2816          :           FUNCT - subroutine to issue an ancillary port command
2817          :           REMAP - used to modify KPAR4 and KPAR5 so that receive/transmit
2818          :           rings can be accessed
2819          :
2820          : Register Usage -
2821          :           R1, R2 - scratch
2822          :           R3 - contains address of PCSR0
2823          :           R4 - pointer to memory location to hold devices's physical
2824          :           address
2825          :
2826          :---+
2827 027706    UNAINI:::
2828          :---+
2829          :     Reset the DELUA/DEUNA then enable device interrupts
2830          :---+
2831 027706 013703 002106'      mov   PCSR0, R3          : move address of PCSR0 to R3
2832 027712 042713 000100        bic   #inte,(R3)       : disable interrupts
2833 027716 012713 000040        mov   #reset, (R3)     : hardware reset una
2834          :
2835 027722 005002              clr   r2                : loop counter init
2836 027724 011301              mov   (R3), r1          : read PCSR0
2837 027726 032701 004000        bit   #DNI, r1          : wait for command to finish
2838 027732 001006              bne   9$                : back til DNI =1
2839 027734 005302              dec   r2                : count down delay
2840 027736 001372              bne   7$                : back until timeout
2841 027740              errdf  9,EMSG15,ERR1      : print " DNI Did not set from"
2842                                :           " a RESET"
2843 027750 012713 004000        mov   #dni, (R3)       : write one to clear DNI
2844 027754 052713 000100        bis   #inte, (R3)       : enable interrupts
2845          :
2846          :---+
2847          :     Tell the device where the port command block is located in
2848          :     host memory
2849          :---+
2850 027760 012763 002150' 000004      mov   #PCBB0,4(r3)  : lower 16 bits of adrs
2851 027766 005063 000006        clr   6(r3)            : upper 2
2852          :
2853 027772              call   comand #getpcb      : load address
2854 030004              P#POP  r2                : get success/failure report
2855 030006 001404              beq   10$              : continue if OK
2856 030010              errdf  10,emsg05,err1      : else report error
2857          :
2858 030020              10$:

```

```
2859
2860
2861
2862
2863 030020      ;---+
2864 030032      ;     Write the rings ...
2865 030034 001407
2866 030036
2867
2868 030046      call   funct @wdrngs      ; write descriptor rings
2869
2870
2871      P$POP R2
2872      beq   20$      ; check for error
2873      errdf 11.emsg16,err1      ; if OK, continue
2874
2875 030054      ; else report error
2876 030066
2877 030070 001404      call   funct @rdphysa      ; read una physical address
2878 030072
2879
2880 030102 012701 002152'      20$:    P$POP r2      ; check for error
2881 030106 012704 002244'      beq   25$      ; if OK, continue
2882 030112 012124      errdf 12.emsg06,err1      ; else report error
2883 030114 012124      mov    #PCBB2, R1      ; store physical address
2884 030116 011114      mov    #PHYADR, R4
2885
2886 030120      mov    (R1)+, (R4)      ; move first two bytes
2887 030126      mov    (R1)+, (R4)      ; and second two
2888      mov    (R1),(R4)      ; and done
2889
2890      CALL  RETMEM      ; restore memory mapping
2891      RETURN
2892
2893      .sbttl uneisr una interrupt service routine
2894      ;---+
2895      ; Functional Description:
2896      ; This is the interrupt service routine for the DELUA/DEUNA.
2897      ; Each time this routine is entered, the following takes place:
2898      ;
2899      ; 1.) All CSRs are saved for debug
2900      ;
2901      ; 2.) All write-one-to-clear bits are cleared
2902      ;
2903      ; 3.) flags corresponding to all bits, except port command
2904      ; field, of PCSR0 are set if the corresponding bits in PCSR0
2905      ; are set.
2906      ;
2907      ; 4.) and, If an error has occurred, then ERRFLG is set
2908
2909      ; Inputs - none
2910
2911      ; Outputs - Implicit -
2912      ; flags are set corresponding to the set bits in PCSR0
2913
2914      ; Calling Procedure: the routine is an interrupt routine, so it is vectored
2915      ; to on device interrupt
2916
2917      ; Side effects - none
2918
```

```

2916          ; Subordinate Routines - none
2917          ;
2918          ; Register Usage -
2919          ;           R1 - address of PCSR0
2920          ;           R3 - contents of PCSR0
2921          ;
2922          ;---+
2923
2924 030130      BGNSRV UNAISR
2925
2926 030130 010146      mov    r1,-(sp)      ;save r1
2927 030132 010246      mov    r2,-(sp)      ;...
2928 030134 010346      mov    r3,-(sp)      ;...
2929
2930 030136 005003      clr    r3            ;setup write 1 to clr mask
2931 030140 013701 002106'    mov    pcsr0,r1      ;get pcsr0 address
2932
2933 030144 011103      mov    (r1),r3      ;and its contents
2934
2935 030146 012137 002116'    mov    (R1)+,PCSR0C      ;save pcsr's for debug
2936 030152 012137 002120'    mov    (R1)+,PCSR1C
2937 030156 012137 002122'    mov    (R1)+,PCSR2C
2938 030162 011137 002124'    mov    (R1),PCSR3C
2939 030166 013701 002106'    mov    PCSR0,R1
2940
2941 030172 000303      swab   r3            ;reorient contents of pcsr0
2942 030174 110361 000001      movb   r3,(r1)      ;write one to clear
2943
2944 030200 000303      swab   r3            ; ONLY CLEAR UPPER BYTE
2945
2946
2947 030202 032703 100400      bit    #seri!fati,r3      ;any fatal status ??
2948 030206 001403      beq    10$          ;
2949
2950 030210 005237 003002'    inc    fatflg        ;set flag
2951 030214 000441      br     90$          ;exit
2952
2953 030216 032703 040000      10$:  bit    #pcei,r3      ;port command error interrupt?
2954 030222 001402      beq    30$          ;no
2955 030224 005237 003004'    inc    pceflg        ;yes, increment flag
2956
2957 030230 032703 010000      30$:  bit    #txi,r3      ;transmit interrupt ??
2958 030234 001402      beq    40$          ;no
2959 030236 005037 003010'    clr    xflag         ;yes, set flag
2960
2961 030242 032703 004000      40$:  bit    #dni,r3      ;command done ??
2962 030246 001402      beq    45$          ;no
2963 030250 005237 003012'    inc    dniflg        ;yes, count each dni
2964
2965 030254 032703 002000      45$:  bit    #rcbi,r3      ;recieve buffer unavailable?
2966 030260 001405      beq    50$          ;no
2967
2968 030262 105737 001274'    tstb   p$list        ; are we listening?
2969 030266 001014      bne    90$          ; YES, we'll have to ignore this
2970 030270 005237 003014'    inc    rbfcnt        ; NO, count them
2971
2972 030274 032703 034000      50$:  bit    #rxii!txii!dni,r3      ;check for non-error interrupt

```

Una interrupt service routine

```

2973 030300 001007      bne    90$          ;exit if one occurred
2974 030302 032703 142000   bit    #seri!pxei!rcbi,r3 ;check for error interrupt
2975 030306 001002      bne    80$          ;if one occurred, incr. errflg
2976 030310 005237 003016'   inc    bcount       ;else, nonsense interrupt
2977 030314 005237 003020'  80$:   inc    errflg_      ;restore registers
2978 030320 012603      90$:   mov    (sp)+,r      ;restore registers
2979 030322 012602      mov    (sp)+,r      ;restore registers
2980 030324 012601      mov    (sp)+,r1     ;restore registers
2981
2982 030326      ENDSRV
2983
2984 .sbttl COMAND Subr to issue a DELUA/DEUNA port command
2985
2986 ;---
2987 ; Functional Description
2988 ; This subroutine issues a DELUA/DEUNA Port Command. Errors
2989 ; are handled by the subroutine ERROR and reported in
2990 ; P2 if one occurred.
2991 ;
2992 ; Inputs -
2993 ; P1 - The DELUA/DEUNA Port Command mnemonic of the
2994 ; desired command.
2995 ;
2996 ; Outputs -
2997 ; P2 - Success report. Contains 0 for success
2998 ; -1 if a DELUA/DEUNA error occurred. This parameter
2999 ; is passed directly from the WAIT
3000 ; routine and is untouched by COMAND.
3001
3002 ; Calling procedure - Call COMAND #<command type>
3003
3004 ; Side effects - If an error has occurred, the routine ERROR will
3005 ; be called.
3006
3007 ; Subordinate Routines -
3008 ; WAIT - wait for the port command to be completed
3009
3010 ; Register usage - R1 contains the command type.
3011
3012 ;---
3013
3014 030330      COMAND:::          P$POP  R1           ;MOVE COMMAND TYPE TO R1
3015 030330          BIS    #INTE,R1        ;ADD INTERRUPT TO COMMAND
3016 030332 052701 000100          MOV    R1,SPCSR0      ;MOV COMMAND TO PCSR0
3017 030336 010177 151544          CALL   WAIT         ;WAIT FOR DCNE INTERRUPT
3018 030342          10$:  RETURN        ;RETURN - ERROR INFO STILL ON
3019 030350          ;                           ;PARAMETER STACK FROM WAIT SUB.
3020
3021
3022 .sbttl FUNCT subr to perform a DELUA/DEUNA Port Function
3023
3024
3025 ;---
3026 ; Functional Description:
3027 ; This subroutine performs a DELUA/DEUNA Ancillary Port command.
3028 ; The function specific PCB is moved into the DELUA/DEUNA PCB.
3029

```

FUNCT subr to perform a DELUA/DEUNA Port Function

```

3030          : Inputs -
3031          :           P1 - The DELUA/DEUNA Port Function mnemonic of the
3032          :           desired function.
3033          : Outputs -
3034          :           P2 - Success report. Contains 0 for success
3035          :           -1 if a DELUA/DEUNA error occurred,
3036          :           This parameter is passed directly from the
3037          :           COMMAND sub and is not affected by FUNCT.
3038
3039          : Calling procedure - Call FUNCT #<function type>
3040
3041          : Side effects - none
3042
3043          : Subordinate routines -
3044          :           COMMAND - used to issue a GET COMMAND port command
3045
3046          : Register usage -
3047          :           R1 - contains the function type, which is transformed
3048          :           to the address of the function specific PCB.
3049          :           R2 - contains the address of the DELUA/DEUNA PCB.
3050
3051          :---+
3052
3053 030352          FUNCT:: P$POP    R1          : get function type into R1
3054 030354 006301      asl        R1          : multiply by two
3055 030356 062701 002160'    add        #funtab,R1   : add function table offset
3056                                     : R1 now contains address of address
3057                                     : of function specific PCB
3058 030362 012702 002150'    mov        #PCBBO, R2   : put DELUA/DEUNA PCB into R2
3059 030366 011101      mov        (R1),R1   : put address of PCB into R1
3060 030370 012122      mov        (R1)+,(R2)+  : mov pcb's
3061 030372 012122      mov        (R1)+,(R2)+  : mov pcb's
3062 030374 012122      mov        (R1)+,(R2)+  : mov pcb's
3063 030376 012122      mov        (R1)+,(R2)+  : mov pcb's
3064 030400              call     COMMAND #getfnt : issue get port function command
3065 030412              return   : success info from COMMAND subroutine
3066
3067                                     : is still on parameter stack
3068 .sbttl XMIT      Transmit DELUA/DEUNA frames
3069
3070          :---+
3071          : Functional Description:
3072          :           This subroutine is used to transmit frames over the DELUA/
3073          :           DEUNA. It sets up the transmit ring for the buffer to be
3074          :           transmitted, namely the status bits (STP,ENP,OWN) and message
3075          :           length. Then a POLL DEMAND port command is issued to alert
3076          :           the device that we have something to transmit.
3077
3078          : Inputs - Implicit
3079          :           The buffer that is pointed to by the ring entry that is
3080          :           pointed to by XRGCUR has been loaded with the data that will
3081          :           be transferred. Also, the variable BUflen has been set to
3082          :           the number of bytes to transmit.
3083
3084          : Outputs - P1 - Success report => 0 = success, -1 = failure
3085
3086          : Implicit - 'RETRYs' : nonzero if transmit failed due to

```

```

3087 ; traffic.
3088
3089 : Calling procedure: Call XMIT
3090 P$POP P1
3091
3092 : Side effects - The ring pointer XRGNXT will be updated to point the next
3093 available entry after the transmit operation.
3094
3095 : Subordinate Routines -
3096     COMMAND - issues poll demand
3097     GETXNX - updates transmit ring pointer
3098     REMAP - used to remap memory so that the transmit ring may
3099         be accessed
3100     RETMEM - used to return the mapping of memory to its original
3101         state
3102
3103 : Register Usage - R1 points to timeout timer location
3104     R2 is used as a pointer if retries is set
3105     R3 is used to pass the success/failure message back
3106     R4 is used as a pointer to ring entries or status info.
3107 :--+
3108
3109 030414 XMIT::: CALL REMAP #OTRING ; enable access to transmit memory
3110 030414
3111
3112 030426 005037 003024' 1$: clr retries
3113 030432 013704 002072' mov xrgcur,R4 ; move ring entry location into R4
3114 030436 032764 100000 000004 bit #own,4(R4) ; make sure we own this
3115 030444 001127 bne 40$ ; else, bookkeeping error
3116 030446 013714 003126' mov buflen,(R4) ; move buffer length into first word of
3117 ; next available ring entry
3118 030452 052764 101400 000004 bis #own!stp!enp,4(R4) ; set ownership, start and end of frame bits
3119 030460 012737 000001 003010' 20$: mov #1,xflag ; set transmit flag
3120 030466 call command #pdmd ; issue pdmd command
3121 030500 P$POP R3 ; check for errors
3122 030502 001130 bne 50$ ; if yes, exit
3123 030504 012701 002050' 22$: mov #TIMER2,R1 ; set up to wait for transmit to complete
3124 030510 012711 000100 mov #100,(R1)
3125 030514 005737 003010' 23$: tst xflag ; see if transmit done bit set
3126 030520 001403 beq 24$ ; if set, skip wait loop
3127 030522 005711 tst (R1) ; else, see if timeout yet
3128 030524 001373 bne 23$ ; no, wait
3129 030526 000510 br 45$ ; yes, exit
3130 030530 032764 100000 000004 24$: bit #own,4(R4) ; see who owns this entry
3131 030536 001072 bne 40$ ; if DELUA/DEUNA still owns this, somethings wrong
3132 030540 032764 040000 000004 bit #errs,4(R4) ; see if any errors
3133 030546 001015 bne 30$ ; if yes, branch and take care of them
3134 030550 26$: CALL GETXNX #xrgcur ; update "transmit ring current" pointer
3135 030562 005003 clr R3 ; indicate success
3136 030564 023737 002072' 002076' cmp xrgcur,xrgnxt ; see if current pointer = next pointer
3137 030572 001054 bne 40$ ; if no, error
3138 030574 005037 003024' clr retries ; let 'retries' reflect success
3139 030600 000473 br 55$ ; return
3140 030602 032764 016000 000004 30$: bit #def!one!more,4(R4) ; was message still sent?
3141 030610 001357 bne 26$ ; if yes, go to next one
3142 030612 032764 002000 000006 bit #retry,6(R4) ; else, did DELUA/DEUNA give up after 16 tries
3143 030620 001434 beq 32$ ; if not, fatal device error, exit

```

```
3144 030622 005237 003024'          inc    retries           ; if yes, keep count of them
3145 030626 022737 000003 003024'    cmp    #3,retries      ; how many tries?
3146 030634 100440                   bmi    43$              ; give up after 3 attempts
3147 030636                   call   getnxn  #xrgcur     ; update pointers
3148 030650                   call   getnxn  #xrgnxt
3149 030662 016402 000010          mov    10(R4),R2       ; set up to copy data buffer
3150 030666 013704 002072'          mov    xrgcur,R4       ; R2 points to old buffer
3151 030672 016403 000010          mov    10(R4),R3       ; R3 points to new buffer
3152 030676 013704 003126'          mov    buflen,R4       ; R4 counts number of bytes to copy
3153 030702 112223                 31$:  movb  (R2)+,(R3)+  ; copy data
3154 030704 005304                 dec    R4
3155 030706 001375                 bne    31$              ; have we copied all of it
3156 030710 000650                 br    1$               ; if yes, try again
3157
3158 030712                   32$:  errdf 13,emsg50,err1 ; else, fatal device error
3159 030722 000420                 br    50$              ; exit
3160
3161 030724                   40$:  errsf 14,emsg10,err1 ; transmit ring bookkeeping error
3162 030734 000413                 br    50$              ; indicate failed due to excessive ...
3163
3164 030736                   43$:  errhfd 15,emsg49    ; ... retries and split!!
3165 030746 000406                 br    50$              ; report error
3166
3167 030750 005237 003022'          45$:  inc    TIMOUT
3168 030754                   errdf 16,emsg08,err1 ; error indicator
3169
3170 030764 012703 177777          50$:  mov    #-1,R3        ; remap memory to its original value
3171
3172 030770                   55$:  CALL   RETMEM      ; return
3173 030776                   return R3
3174
3175 .sbttl RECEIVE Receive DELUA/DEUNA ring buffers
3176
3177 :---+
3178 : Functional Description
3179 : This subroutine handles the reception of incoming frames
3180 : from the DELUA/DEUNA. When called, it looks at the status of
3181 : RRGCUR (current entry in receive ring). If this entry is owned
3182 : by the host and there are no errors in the status information,
3183 : the frame is delivered to the caller of the routine. Upon
3184 : seeing a successful routine, the caller will take the contents
3185 : of the buffer pointed to by the ring entry pointed to by RRGCUR
3186 : as the received frame. If there is an error or the entry
3187 : pointed to by RRGCUR belongs to the device, then an unsuccessful
3188 : status is returned.
3189 : After a valid frame is found, a POLL DEMAND is issued
3190 : to let the device know that we've got an empty buffer.
3191 :
3192 : Inputs - none
3193 :
3194 : Outputs - P1 - The number of frames handled by this call to RECEIVE.
3195 : either 1 or 0.
3196 :
3197 : Implicit - If P1 = 1 then the received frame is located in the
3198 : buffer pointed to by the entry pointed to by RRGCUR.
3199 :
3200 : Calling procedure - Call RECEIVE
```

```

3201          : P$POP P1
3202
3203          : Side effects -
3204          :   1.) The pointers RRGCUR and RRGNXT are updated.
3205          :   2.) KPAR4 and KPAR5 are left mapping to the receive ring. This
3206          :       is done because this structure is consistently accessed
3207          :       immediately after a call to RECEIVE
3208
3209          : Subordinate Routines -
3210          :   GETRNX - updates RRGCUR and RRGNXT
3211          :   COMAND - used to issue poll demand
3212          :   REMAP - used to remap memory so that the receive ring may
3213          :       be accessed.
3214          :   RELBUF - used to release unwanted receive buffers
3215
3216          : Register usage - R1 is used to hold current frame status information
3217          : R2 counts the number of frames handled
3218          : R4 points to the ring descriptor entry
3219
3220          :---+
3221
3222 031002  RECEIVE:::          005002
3223 031002          clr      R2          : clear frames handled counter
3224
3225 031004          1$:      CALL     REMAP  #0RRING
3226 031016 013704 002074'          mov      rrcur,R4          : allow access to receive ring
3227 031022 016401 000004          mov      4(R4),R1          : move current receive ring pointer to R4
3228 031026 032701 100000          bit      #own,R1          : move status of frame to R1
3229 031032 001070          bne      60$          : see who owns this buffer
3230
3231          :---+
3232          ; If the listen command has been issued, then don't do any protocol filtering
3233          ; here
3234          :---+
3235
3236 031034 105737 001274'          tstdb    p$list          : Are we listening?
3237 031040 001031          bne      10$          : yes, don't protocol filter
3238
3239 031042 016403 000010          mov      10(R4),R3          : move buffer address into R3
3240 031046 016303 000014          mov      protot(R3),R3          : move prototype into R3
3241 031052 020337 003034'          cmp      R3,prot00          : see if it is an acceptable protocall type
3242 031056 001422          beq      10$          : if yes, cont.
3243 031060 020337 003036'          cmp      R3,prot02          : else check other good type
3244 031064 001417          beq      10$          : if OK, cont.
3245
3246 031066          5$:      CALL     GETRNX #RRGCUR          : update current receive pointer
3247 031100          CALL     GETRNX #RRGNXT          : update next receive pointer
3248 031112          CALL     RELBUF R4          : release buffer to DELUA/DEUNA
3249 031122 000434          BR      60$          : and exit
3250
3251 031124 032701 040000          10$:     bit      #errs,R1          : see if any errors
3252 031130 001421          beq      20$          : for no errors br to 20$
3253
3254          :---+
3255          ; If a CRC error has occurred and we are in promiscuous mode (LISTEN
3256          ; command is executing) then ignore this error. Most likely the device's
3257          ; own system ID will be the cause of the error. When the device tries

```

```

3258          ; to send (sys. ID) and receive (prom. mode) it gets a CRC error.
3259          ;---+
3260
3261 031132 105737 001274'      tstb   p$list           ; Are we executing listen command
3262 031136 001403               beq    15$              ; No, go log error
3263 031140 032701 004000       bit    #crc,R1          ; Is this a CRC error?
3264 031144 001350               bne    5$              ; yes, just leave without logging error
3265
3266 031146 005237 003026'      15$:   inc    rcverr          ; else,
3267 031152               printf  #recerr          ; increment receive error counter
3268 031172 000735               br     5$              ; print error message
3269 031174 005237 003030'      20$:   inc    rcvbuf          ; update pointers and return
3270 031200 005202               inc    R2              ; increment good buffers received counter
3271
3272 031202               CALL   GETRNX #RRGCUR        ; keep count of how many buffers received
3273 031214               return  R2              ; update "receive ring current" pointer
3274
3275          ;---+
3276          ; Name - RELBUF                      Release a receive buffer
3277
3278          ; Functional Description
3279          ; This routine is called to release a receive buffer to the
3280          ; DELUA/DEUNA. It will set the ownership of a receive ring
3281          ; entry and then issue a poll demand port command to alert
3282          ; the device of an available buffer.
3283
3284          ; Inputs - Explicit -
3285          ; P1 - pointer to receive ring entry
3286
3287          ; Outputs - none
3288
3289          ; Calling Procedure: CALL RELBUF P1
3290
3291          ; Side Effects -
3292          ; 1.) The ownership of the ring entry pointed to by P1 goes
3293          ; to the device.
3294          ; 2.) If the poll demand fails then an error is printed and
3295          ; the diagnostic is exited
3296
3297          ; Subordinate Routines - noe
3298
3299          ; Register usage -
3300          ; R1 - pointer to receive ring entry
3301
3302          ;---+
3303 031220               RELBUF:::                    ; get pointer to receive ring entry
3304 031220               P$POP   R1              ; allow access to receive ring
3305 031222               CALL    REMAP  #ORRING
3306 031234 052761 100000 000004      BIS    #OWN,4(R1)          ; release the buffer to the device
3307 031242               CALL    COMAND #PDMD
3308 031254               P$POP   R1              ; issue poll demand port command
3309 031256 001404               BEQ    10$              ; get success indicator
3310 031260               ERRDF  17,EMSG09,ERR1        ; SUCCESS, continue
3311
3312 031270               10$:   CALL   RETMEM          ; print error message
3313 031276               RETURN
3314

```

3315
3320
3321 :---
3322 : Functional Description:
3323 :
3324 : This routine will convert a string of HEX characters into a
3325 : right justified binary stream (with leading zeros),
3326 : compatible with Ethernet conventions. The source string must
3327 : be formatted using either a word by word hex description
3328 : or a byte by byte hex description. The returned string
3329 : will be BYTE oriented as required by the Ethernet:
3330 :
3331 : lo-byte-word0 hi-byte-word0 lo-byte-word1 hi-byte-word1, etc.
3332 :
3333 : Inputs -
3334 : p1 - address of the source (HEX) string to be converted to
3335 : a binary stream.
3336 : p2 - address of the desired destination buffer which will
3337 : accept binary data
3338 : p3 - length (in bytes) of the destination buffer
3339 :
3340 : Outputs - p4 - zero if successful, -1 if buffer too long or odd number of
3341 : hex characters
3342 :
3343 : Implicit - The buffer at p2 will contain a right justified binary
3344 : stream w/ leading zeros and corresponding to hex string
3345 : at R5.
3346 :
3347 : Calling Procedure: CALL EDPACK p1,p2,p3
3348 : P\$POP P4
3349 :
3350 : Side Effects - none
3351 :
3352 : Subordinate Routines -
3353 : HXFORM - Strip non-HEX characters from input string
3354 : HEXBIN - HEX to binary conversion
3355 :
3356 :---
3357 031300 locdst: .blk 74. :max number of characters that may be entered
3358 031412 000000 source: .word :source address
3359
3360 031414 EDPACK::
3361 031414 p\$pop source,r4,r3 :r4=destination, r3-number of chars reqd
3362 :source-src address, orient-word/byte?
3363 031424 005002 clr r2 :assume no errors, value returned
3364 031426 006303 asl r3 :number of characters required w/ "0"s
3365 031430 call HXFORM source,#locdst,r3
3366 031450 p\$pop r1,r2 :r1=address of last char
3367 :r2=success/fail code (0/-1)
3368 031454 005702 tst r2 :R1 will point to rightmost character
3369 031456 001010 bne 9\$:right justify buffer
3370 :convert hex at locdst to binary
3371 031460 006203 asr r3 :r3 bytes in output bit stream
3372 031462 call HEXBIN #locdst,r4,r3
3373
3374 031500 9\$: return r2 :return with success/failure indication
3375

HXFORM HEX FORMAT ROUTINE

```

3380      ;---+
3381      ; Functional Description
3382      ; This routine is used to form a string of packed HEX characters.
3383      ; It accepts an input string and the number of characters
3384      ; to be used in the output sting. Any spaces and dashes are
3385      ; stripped out of the string. Invalid characters will cause
3386      ; an error to be returned.
3387
3388      ; Inputs -    P1 - the address of the source string to be formatted.
3389      ;                 P2 - the address of a buffer to get the formatted string.
3390      ;                 P3 - the number of HEX characters to look for.
3391
3392      ; Outputs -   P4 - pointer to the last valid character of the output string.
3393      ;                 P5 - success indicator - 0=success, -1=error.
3394
3395      ; Calling Procedure - CALL HXFORM P1,P2,P3
3396      ;                 P$POP      P4,P5
3397
3398      ; Side effects - None
3399
3400      ; Subordinate Routines - None
3401
3402      ; Register Usage
3403      ; R1 - address of source string
3404      ; R2 - address of destin string
3405      ; R3 - number of HEX characters desired
3406      ; R4 - byte of source string/success indicator
3407
3408      ;---+
3409 031504 HXFORM:::          P$POP  R1,R2,R3      ; Get inputs
3410 031504
3411
3412 031512 112104           5$:  MOVB  (R1)+,R4      ; get a byte of the source string
3413 031514 120427 000040     CMPB  R4,#40      ; Are we looking at a space?
3414 031520 001774           BEQ   5$          ; Yes, valid char., get next
3415 031522 120427 000055     CMPB  R4,#55      ; Are we looking at a dash?
3416 031526 001771           BEQ   5$          ; Yes, valid char., get next
3417
3418
3419      ; Check to see if we've got a HEX digit. ASCII range for HEX is 60 <= CHAR < 72
3420      ; and 101 <= CHAR < 107
3421
3422
3423 031530 120427 000060           CMPB  R4,#60      ; Is CHAR < 60?
3424 031534 100417           BMI   HXERR      ; CHAR out of range - error
3425 031536 120427 000072           CMPB  R4,#72      ; Is 60 <= CHAR < 72?
3426 031542 100407           BMI   10$        ; CHAR is good
3427 031544 120427 000101           CMPB  R4,#101     ; Is CHAR < 101?
3428 031550 100411           BMI   HXERR      ; CHAR out of range - error
3429 031552 120427 000107           CMPB  R4,#107     ; Is 101 <= CHAR < 107?
3430 031556 100401           BMI   10$        ; CHAR is good
3431 031560 000405           BR    HXERR      ; Else - error
3432
3433 031562 110422           10$: MOVB  R4,(R2)+    ; put HEX digit in dest. string
3434 031564 005303           DEC   R3          ; decrement # of chars. to find
3435 031566 001351           BNE   5$          ; non-zero means more to do
3436 031570 005004           CLR   R4          ; indicate success in R4

```

HXFORM HEX FORMAT ROUTINE

```

3437 031572 000402          BR      HXEXIT           ; and depart!!
3438
3439 031574 012704 177777    HXERR: MOV   #1,R4        ; indicate error in R4
3440 031600                   HXEXIT: RETURN R2,R4     ; return results
3441
3442
3443
3444
3445
3446
3447 : Functional Description:
3448 : This procedure will convert a string of hex (ASCII) characters
3449 : directly to a binary stream. The destination binary stream will
3450 : require only half as many bytes as the hex string because only
3451 : one byte is required to represent two hex digits
3452
3453 : Inputs -
3454 : p1 - source string address (delimited by a null)
3455 : p2 - destination address for the binary data.
3456 : p3 - the number of bytes required (half the number of
3457 : characters at p1).
3458
3459 : Outputs - Implicit -
3460 : The buffer at p2 will contain the binary stream, converted
3461 : directly from the buffer at p1.
3462
3463 : Calling Procedure: CALL      HEXBIN p1,p2,p3
3464
3465 : Side Effects - none
3466
3467 : Subordinate Routines - none
3468
3469 : Register Usage -
3470 : R1 - source string address
3471 : R2 - destination string address
3472 : R3 - holds one byte of binary representation of two characters
3473 : R4 - pointer to compare string
3474
3475 :---
3476 031606 000000          hn: .word
3477 031610 060   061   062  cmpstr: .ASCIZ /0123456789ABCDEF/
            031613 063   064   065
            031616 066   067   070
            031621 071   101   102
            031624 103   104   105
            031627 106   000
3478
3479
3480 031632
3481 031632          .even
3482
3483
3484
3485 031642 060237 031606'  HEXBIN:::          p:pop  r1,r2,hn
3486
3487 031646 012704 031610'  p:pop  r1,r2,hn
3488 031652 121124          1$:  mov   #cmpstr,r4
3489 031654 001376          2$:  cmpb  (r1),(r4)
3490 031656 005201          bne   2$              ;compare current char with a char in cmpstr
3491 031660 162704 031611'  inc   r1              ;repeat until character found in list
3492                      sub   #cmpstr+1,r4
                                r1              ;point to the next ASCII byte
                                sub   #cmpstr+1,r4
                                r4              ;r4 now contains the actual binary value for
                                r4              ;the nibble described by the current byte.

```

```

3493 031664 006304
3495 031666 006304
3496 031670 006304
3497 031672 006304
3498 031674 010403
3499
3500 031676 012704 031610'
3501 031702 121124
3502 031704 001376
3503 031706 005201
3504 031710 162704 031611'
3505
3506
3507 031714 050403
3508
3509 031716 110322
3510 031720 020237 031606'
3511 031724 100750
3512 031726
3513
3514
3515
3520
3521
3522 ; Functional Description:
3523 ; This procedure will convert a binary data stream into a hex string.
3524
3525 ; Inputs -
3526 ; p1 - binary data buffer address
3527 ; p2- number of bytes in the buffer
3528 ; p3- address of output buffer for hex string. Contains hex
3529 ; character pairs seperated by "-"'s (note: this buffer must
3530 ; be at least 3*p2 bytes long)
3531 ; Outputs - Implicit
3532 ; the buffer at p3 will contain the hex string followed by a
3533 ; NULL character.
3534
3535 ; Calling Procedure: CALL BINHEX P1,P2,P3
3536
3537 ; Subordinate Routines - none
3538
3539 ; Register Usage -
3540 ; R1 - input buffer address
3541 ; R2 - output buffer address
3542 ; R3 - contains one nibble of input string
3543 ; R4 - contains one byte of input string
3544
3545
3546 031730 060 061 062 hexc: .ASCII /0123456789ABCDEF/
031733 063 064 065
031736 066 067 070
031741 071 101 102
031744 103 104 105
031747 106
3547 031750 000000
3548

```

```

3549 031752          BINHEX:::                                ;R1 has the input buffer address
3550 031752          p$pop    r1,1st,r2      ;1st: has the number of bytes in input buffer
3551                                         ;R2 has the output buffer address
3552                                         ;1st is now address of last source byte + 1
3553 031762 060137 031750'        1$:      add     r1,1st      ;get the current byte and point to next byte
3554 031766 112103           movb    (r1)+,r3      ;separate nibbles and get characters separately
3555 031770 110304           movb    r3,r4      ;only right binary nibble remains in r3
3556 031772 042703 177760       bic     #177760,r3      ;shift over for left binary nibble in r4
3557 031776 006204           asr     r4
3558 032000 006204           asr     r4
3559 032002 006204           asr     r4
3560 032004 006204           asr     r4
3561 032006 042704 177760       bic     #177760,r4      ;only left binary nibble remains in r4
3562                                         ;r4 is the most significant nibble (first)
3563                                         ;r3 is the least significant nibble (second)
3564 032012 116422 031730'        movb    hexc(r4),(r2)+   ;put the ascii byte into the buffer hi position
3565 032016 116322 031730'        movb    hexc(r3),(r2)+   ;put the ascii byte into the buffer lo position
3566 032022 112722 000055       movb    #'-,(R2)+      ;put - between hex pairs
3567 032026 020137 031750'        cmp     r1,1st      ;result is negative until r1=1st
3568 032032 103755           blo    1$          ;until r1=1st. (transfer all source bytes)
3569 032034 105042           clrb    -(r2)      ;terminate output buffer with a null
3570 032036           RETURN

3571
3572 .sbttl BLDLD Build loop direct data buffers for transmit.
3573
3574
3575 :--+
3576 : Functional Description:
3577 : This subroutine builds loop direct frames for transmission
3578 : from the DELUA/DEUNA. Source address, Destination address,
3579 : Prot. type, and loop direct header info are added
3580 : to the message buffer. The message buffer is built
3581 : by a call to BLDBUF.
3582 : Inputs -
3583 : P1 - The address of the destination address (from node table)
3584 : implicit - P$SIZE contains the size of the message buffer data
3585 : XRGNXT points to the next available ring entry
3586 : PHYADR holds the current local DELUA/DEUNA physical address
3587 : Outputs - Implicit -
3588 : The buffer pointed to by the transmit ring entry pointed to
3589 : by XRGNXT contains a loop direct message to the address pointed
3590 : to by P1.
3591 : Calling procedure - CALL BLDLD P1
3592
3593 : Side effects - none
3594
3595 : Subordinate Routines -
3596 : BLDBUF - build a data buffer for transmit
3597 : GETXNX - update XRGNXT
3598 : REMAP - used to remap memory so that the transmit ring may be
3599 : accessed
3600 : RETMEM - used to return the mapping of memory to its original
3601 : state
3602
3603 : Register usage - R1 holds address of destination address
3604 : R2 is a pointer for the loop direct header info
3605

```

```

3606 : R3 holds the frame length
3607 : R4 holds address of next ring entry data buffer
3608 :
3609 :
3610 :
3611 032040 BLDLD:::
3612 032040 P$POP R1 : put address of dest. address in R1
3613 032042 CALL REMAP #OTRING : allow access to transmit ring
3614 032054 013704 002076' mov xrgnxt,R4 : move next frame address to R4
3615 032060 032764 100000 000004 bit $own,4(R4) : check ownership bit
3616 032066 001075 bne 40$ : if don't own, bookkeeping error.
3617 032070 016404 000010 mov 10(R4),R4 : point R4 to data block
3618 032074 005064 000006 clr sourcc(R4) : leave blank space for source address
3619 032100 005064 000010 clr sourcc+2(R4) : six bytes worth
3620 032104 005064 000012 clr sourcc+4(R4)
3621 032110 013764 003034' 000014 mov prot00,protot(R4) : move protocall type into header
3622 032116 012702 003260' mov #LOPDIR,R2 : move loopdirect format header loc. to R2
3623 032122 012264 000016 mov (R2)+,ldskip(R4) : skip count
3624 032126 011264 000020 mov (R2),ldfct1(R4) : function code (forward)
3625 032132 013764 002244' 000022 mov PHYADR,1ddadr1(R4) : local node address
3626 032140 013764 002246' 000024 mov PHYADR+2,1ddadr1+2(R4) : six bytes
3627 032146 013764 002250' 000026 mov PHYADR+4,1ddadr1+4(R4)
3628 032154 016264 000010 000030 mov 10(R2),ldfct2(R4) : function code (reply)
3629 032162 013764 002244' 000032 mov PHYADR,1ddadr2(r4) : local node address
3630 032170 013764 002246' 000034 mov PHYADR+2,1ddadr2+2(R4) : six bytes
3631 032176 013764 002250' 000036 mov PHYADR+4,1ddadr2+4(R4)
3632 032204 CALL MOVEXT #ONTAB,R1,#OTRING,R4,$3 : move dest. addr. into frame
3633 032232 CALL BLDBUF R4,#1data : build data buffer
3634 032246 CALL GETXNX #XRGNX : update pointer to next ring entry
3635 032260 000405 br 60$ : exit
3636 :
3637 032262 40$: errsf 18,emsg10,err1 : transmit ring bookkeeping error
3638 032272 000400 br 60$ : exit
3639 :
3640 032274 60$: CALL RETMEM : remap memory to original
3641 032302 RETURN
3642

```

```
3644 .sbttl BLDFAS Build frame for full assist transmission.  
3645  
3646 :---+  
3647 : Functional Description:  
3648 : This subroutine builds full assist frames for transmission  
3649 : from the DELUA/DEUNA. A full assist is a loop through two  
3650 : nodes: the target and assist nodes. The target node is the  
3651 : node that is being tested and the assist node is the node  
3652 : that is helping with the transmission to and the reception  
3653 : from the target node. The full assist frame is sent from the  
3654 : NIE node to the assist node, which sends it to the target node,  
3655 : which sends it back to the assist node, which, finally  
3656 : returns it to the NIE node.  
3657 :  
3658 : Inputs -  
3659 : P1 - pointer to the ethernet address of the target node  
3660 : P2 - pointer to the ethernet address of the assist node  
3661 :  
3662 : Implicit -  
3663 : P$SIZE - contains the size of the message buffer data  
3664 : XRGNXT - points to the next available ring entry  
3665 : PHYADR - holds the current local node address  
3666 :  
3667 : Outputs - Implicit -  
3668 : A full assist loopback frame has been built in the buffer  
3669 : pointed to by the transmit ring entry pointed to by XRGNXT  
3670 :  
3671 : Calling Procedure - CALL BLDFAS P1  
3672 :  
3673 : Side Effects - XRGNXT is updated to point to the next transmit ring entry  
3674 :  
3675 : Subordinate Routines -  
3676 : BLDDBUF - fills frame to be transmitted with data  
3677 : GETXNX - update current transmit ring pointer  
3678 : REMAP - used to remap memory so that the transmit ring may be  
3679 : accessed  
3680 : RETMEM - used to return the mapping of memory to its original  
3681 : state  
3682 :  
3683 : Register usage - R1 holds address of target node address  
3684 : R2 holds address of assist node address  
3685 : R3 holds the frame length  
3686 : R4 holds address of next ring entry data buffer  
3687 :  
3688 :---+  
3689 :  
3690 032304 :  
3691 032304 :  
3692 :  
3693 :  
3694 032310 :  
3695 :  
3696 032322 013703 002076' :  
3697 032326 032763 100000 000004 :  
3698 032334 001144 :  
3699 032336 016304 000010 :  
3700 :  
BLDFAS::  
P$POP R1,R2 ; put address of target address into R1  
; and address of assist address into R2  
CALL REMAP #OTRING ; enable access to transmit memory  
mov xrgnxt,R3 ; move next frame address to R3  
bit #own,4(R3) ; check ownership bit  
bne 40$ ; if don't own, bookkeeping error.  
mov 10(R3),R4 ; point R4 to buffer
```

```

3701
3702
3703
3704
3705 032342 005064 000006      :---+ DELUA/DEUNA will add in source address.
3706 032346 005064 000010      :---+
3707 032352 005064 000012      :---+ clr    source(R4)          ; leave blank space for source address
3708
3709
3710
3711
3712
3713 032356 013764 003034' 000014      :---+ clr    source+2(R4)        ; six bytes worth
3714 032364 012764 000000 000016      :---+ clr    source+4(R4)
3715 032372 012764 000002 000020      :---+
3716 032400 012764 000002 000030      :---+ Add protocol type, skip count, and function code fields to frame
3717 032406 012764 000002 000040      :---+
3718 032414 012764 000001 000050      :---+ mov    prot00.protot(R4)   ; move protocall type into header
3719
3720
3721
3722
3723
3724
3725 032422 013764 002244' 000042      :---+ mov    #0.fskip(R4)       ; skip count
3726 032430 013764 002246' 000044      :---+ mov    #2.fafct1(R4)     ; function code (forward)
3727 032436 013764 002250' 000046      :---+ mov    #2.fafct2(R4)     ; function code (forward)
3728
3729
3730
3731
3732
3733
3734 032444 013764 002244' 000052      :---+ mov    #2.fafct3(R4)     ; function code (forward)
3735 032452 013764 002246' 000054      :---+ mov    #1.fafct4(R4)     ; function code (reply)
3736 032460 013764 002250' 000056      :---+
3737
3738
3739
3740
3741
3742
3743 032466 062704 000022      :---+ Our physical address is the third forward address. This completes
3744 032514 062704 000010      :---+ the loop.
3745 032520
3746 032546
3747 032552
3748
3749 032600
3750 032612 016304 000010      :---+ Now add all portions of the frame that come from the node table.
3751 032616
3752 032632
3753 032644 000405      :---+ Namely, destination, target node, and assist node
3754
3755 032646 40$: errsf 19.emsg10,err1      :---+ CALL   MOVEXT #ONTAB,R2,#0TRING,R4,#3 ; move in dest. addr.
3756 032656 000400      :---+ ADD    #FAADDR1,R4           ; point R4 to first forward addr.
3757

```

3758 032660
3759 032666
3760

50\$: CALL RETMEM
RETURN ; remap memory to original

```

3762          .sbttl BLDREQ Build Request ID Frames for transmit.
3763
3764
3765      :--+
3766      : Functional Description:
3767      : This subroutine builds Request ID frames for transmission
3768      : from the DELUA/DEUNA. Source address, destination address,
3769      : protocall type, sequence number and Request ID
3770      : header info are built into the buffer.
3771
3772      : Inputs - Implicit -
3773      : The destination address is contained in ADRBUF.
3774
3775      : Outputs - Implicit -
3776      : The buffer pointed to by the transmit ring entry pointed
3777      : to by XRGNXT contains a request ID message.
3778
3779      : Calling Procedure - CALL BLDREQ
3780
3781      : Side Effects -
3782      : XRGNXT - updated to point to next transmit ring entry
3783
3784      : Subordinate Routines -
3785      : GETXNX - updates XRGNXT
3786      : REMAP - used to remap memory so that the transmit ring may be
3787      : accessed
3788      : RETMEM - used to return the mapping of memory to its original
3789      : state
3790
3791      : Register Usage -
3792      : R2 - is a pointer for Request ID header info.
3793      : R4 - holds address of next ring entry data buffer.
3794
3795      :--+
3796 032670          BLDREQ:::
3797 032670          013704 002076'          CALL    REMAP  #0TRING      ; allow access to transmit ring
3798 032702 032764 100000 000004          mov     XRGNXT,R4      ; move next frame address to R4
3799 032706          bit    #own,4(R4)      ; check ownership bit
3800 032714 001050          bne    40$          ; if don't own, bookkeeping error
3801 032716 016404 000010          mov    10(R4),R4      ; point R4 to data block
3802 032722 012737 000100 003126'          mov    #100,buflen   ; move buffer size to buflen
3803 032730 005064 000006          clr    sourcec(R4)    ; leave blank space for source addr.
3804 032734 005064 000010          clr    sourcec+2(R4)
3805 032740 005064 000012          clr    sourcec+4(R4)
3806 032744 013764 003036' 000014          mov    prot02,protot(R4) ; move protocall type into header
3807 032752 012702 003252'          mov    #REQID,R2      ; move Request ID header loc. to R2
3808 032756 012264 000016          mov    (R2)+,header(R4) ; byte count
3809 032762 012264 000020          mov    (R2)+,header+2(R4) ; function code (request ID)
3810 032766 011264 000022          mov    (R2),header+4(R4) ; receipt no.
3811 032772          CALL   MOVEEXT #0NTAB,#ADRBUF,#0TRING,R4,#3 ; set up destination addr. of frame
3812 033022          CALL   GETXNX #XRGNXT      ; update pointer to next ring entry
3813 033034 000404          br    50$          ; exit
3814 033036          40$: errsf 20,errmsg0,err1 ; transmit ring bookkeeping error
3815 033046          50$: CALL   RETMEM       ; return memory mapping to its origin
3816 033054          RETURN
3817
3818

```

```
3819          .sbttl GET?NX Get next transmit or receive ring entry
3820
3821          ;---+
3822          ; Functional Description
3823          ; This subroutine gets the next transmit or receive ring
3824          ; entry. It is entered at separate points depending on
3825          ; which ring is being used.
3826          ;
3827          ; Inputs - P1 - The address of the ring pointer to be updated.
3828          ;
3829          ; Outputs - The ring pointer is updated to point to the next available
3830          ; entry.
3831          ;
3832          ; Calling procedure - CALL GETXNX #P1           ; for transmit updates
3833                  CALL GETRNX #P1           ; for receive updates
3834          ;
3835          ; Side effects - None
3836          ;
3837          ; Subordinate Routines - none
3838          ;
3839          ; Register Usage - R1 points to the first entry in the ring
3840          ; R2 points to the last entry in the ring
3841          ; R3 is the address of the ring pointer to be updated
3842          ;
3843          ;---+
3844
3845 033056
3846 033056 013701 002070'
3847 033062 013702 002104'
3848 033066 000404
3849 033070
3850 033070 013701 002066'
3851 033074 013702 002102'
3852 033100
3853 033102 021302
3854 033104 001403
3855 033106 062713 000012
3856 033112 000401
3857 033114 010113
3858 033116
3859
3860
3861
3862          .sbttl BLDBUF Build Message Buffers
3863
3864          ;---+
3865          ; Functional Description
3866          ; This routine fills a transmit buffer with data. It will load
3867          ; bytes into the buffer to pad the data field out to P$SIZE bytes.
3868          ;
3869          ; Inputs -
3870          ; P1 - address of the beginning of a transmit buffer
3871          ; P2 - number of bytes already loaded into data field of
3872          ; the transmit buffer to be worked on
3873          ;
3874          ; Implicit -
3875          ; P$SIZE contains the size the buffer is to be
```

```

3876 ; P$TYPE contains the message type
3877 ;
3878 ; Outputs - Implicit -
3879 ; Buffer starting at location P1 contains a message P$SIZE bytes
3880 ; long using the message type specified by P$TYPE.
3881 ;
3882 ; Calling procedure: Call BLDBUF P1,P2
3883 ;
3884 ; Side effects -
3885 ; XFER - gets loaded with the number of bytes that will be
3886 ; transferred -- used by summary routine
3887 ; BUflen - loaded with the length of the transmit buffer
3888 ; CMPBUF - address of the data field of the transmit buffer to
3889 ; be used in data compare routine
3890 ;
3891 ; Subordinate Routines - none
3892 ;
3893 ; Register usage - R1 - scratch
3894 ; R2 - (message type X 2), used as offset for pointers
3895 ; R3 - points to the next byte of the buffer under construction
3896 ; R4 - points to the last byte of the buffer under construction
3897 ;
3898 ;---+
3899 ;
3900 033120 BLDBUF:::
3901 033120 P$POP R3,R1 ; put buffer address into R3
3902 ; and number of bytes in buffer in R1
3903 033124 CALL REMAP #0TRING ; allow access to transmit ring
3904 ;
3905 ;---+
3906 ; set up the boundaries of the data transfer
3907 ;---+
3908 ;
3909 033136 062703 000016 add #16,R3 ; point R3 past header info
3910 033142 013704 001172' mov P$SIZE,R4 ; put size into R4
3911 033146 060304 add R3,R4 ; make R4 = last byte of data buffer
3912 033150 010337 003130' MOV R3,CMPBUF ; store pointer to data field for data
3913 ; compare
3914 ;
3915 033154 060103 add R1,R3 ; point R3 past data already in buffer
3916 ;
3917 ;---+
3918 ; Set up transfer size and buffer length
3919 ;---+
3920 033156 012737 000016 003126' MOV #16,BUflen ; buffer length = header ...
3921 033164 063737 001172' 003126' ADD P$SIZE,BUflen ; ... + data field
3922 033172 013737 003126' 003120' MOV BUflen,XFER ; transfer size for summary
3923 ;
3924 ;---+
3925 ; Set up pointer to message to fill with
3926 ;---+
3927 033200 013702 001170' mov P$TYPE,R2 ; put message type into R2
3928 033204 006302 asl R2 ; multiply by 2
3929 033206 016201 001450' mov MSGAD(R2),R1 ; point R1 to first byte of stored message
3930 ;
3931 033212 005037 003032' 10$: clr COUNT ; clear byte counter
3932 033216 005237 003032' inc COUNT ; count no. of bytes copied

```

3933 033222 112123	112123	movb (R1) .,(R3)+	: put byte in buffer
3934 033224 026237 001432'	003032'	cmp MSGCNT(R2),COUNT	: are we at end of stored message
3935 033232 001004	bne 20\$: if no, check if done	
3936 033234 016201 001450'	mov MSGAD(R2),R1	: else, point R1 to begining	
3937 033240 005037 003032'	clr COUNT	: and clear counter	
3938 033244 020304	20\$: cmp R3,R4	: is buffer filled?	
3939 033246 001363	bne 10\$: if no, loop	
3940	CALL RETMEM	: restore memory mapping	
3941 033250	RETURN	: else, return	
3942 033256			
3943			

```

3945
3946 .sbttl DATCMP Compare data buffers
3947
3948 :---+
3949 : Functional Description
3950 : This subroutine compares two data buffers byte by byte.
3951 : If comparison errors occurred, location, expected data
3952 : and received data are printed out for the first five
3953 : errors. The total number of errors is also printed.
3954
3955 : Inputs - P1 - The size (in bytes) of the buffer to be compared.
3956 : P2 - The address of buffer to compare other buffer against.
3957 : P3 - The address of the second buffer.
3958
3959 : Outputs - P4 - The number of comparison errors.
3960
3961 : Calling Procedure - CALL DATCMP P1,P2,P3
3962 : P$POP P4
3963
3964 : Subordinate Routines - none
3965
3966 : Side effects - none
3967
3968 : Register Usage - R1 - number of words to compare
3969 : R2 - pointer to data in transmit buffer
3970 : R3 - pointer to data in receive buffer
3971 : R4 - contains the word offset (words from beginning of data)
3972
3973 :---+
3974
3975 033260 DATCMP::: P$POP R1,R2,R3 : put compare size in R1
3976 033260 : R2 gets transmit data address
3977 : R3 gets receive data address
3978 : init. return value
3979 033266 005037 003110' CLR TEMP : has no compare been selected?
3980 033272 105737 001303' TSTB P$NCMP : branch if yes
3981 033276 001402 BEQ 1$ : leave
3982 033300 000137 033562' JMP 30$ : initialize byte offset
3983 033304 1$: MOV #-1,R4 : make even number of word compares
3984 033304 012704 177777 BIC #BIT0,R1 : increment offset counter
3985 033310 042701 000001 10$: inc CMPEXT #OTRING,R2,#ORRING,R3,#1 : compare a word
3986 : P$POP R0 : get compare indicator
3987 033314 005204 beq 20$ : if same, branch
3988 033316 inc temp : increment error counter
3989 033344 cmp temp,#1 : is this the first error?
3990 : bgt 15$ : NO, skip header
3991 033346 001462 PRINTX #CMPERH : YES, print a header
3992 033350 005237 003110' 15$: cmp #5,temp : if more than 5 errors,
3993 033354 023727 003110' 000001 blt 20$ : don't print message
3994 033362 003010 CALL REMAP #OTRING : allow access to transmit buffer
3995 033364 : printx #cmper1,R4,(R2) : print expected word
3996 : CALL REMAP #ORRING : allow access to receive buffer
3997 033404 022737 000005 003110' 15$:
3998 033412 002440
3999 033414
4000 033426
4001 033452

```

DATCMP Compare data buffers

```

4002 033464          PRINTX  #CMPPER2,(R3)      ; print received word
4003 033506          CALL    RETMEM            ; restore memory mapping
4004
4005 033514 005722   20$:   TST    (R2)+           ; point R2 to next transmitted word
4006 033516 005723   TST    (R3)+           ; point R3 to next received word
4007 033520 162701 000002   SUB    #2,R1           ; decrement number of words to compare
4008 033524 003273   bgt    10$              ; if not finished, go back for more
4009 033526 022737 000000 003110'   cmp    #0,temp        ; were there any errors?
4010 033534 001412   beq    30$              ; if no, exit
4011 033536          printx #cmper3,temp       ; return with error count on stack
4012 033562          RETURN  temp             ; return with error count on stack
4013
4014
4015 .sbttl WRITES Write data onto summary table
4016
4017 :--+
4018 : Functional Description:
4019 : This subroutine updates the summary table data for
4020 : the nodes specified in the call statement. Either one
4021 : or two nodes can be updated per call. After the call,
4022 : the summary data counters are cleared. The summary table
4023 : is checked for a matching node address and updates the
4024 : counters for that node, or adds the node to the table if it
4025 : doesn't exist. An error is reported if the end of the table
4026 : is reached.
4027
4028 : Inputs -
4029 : P1 - The number of nodes to update (1 or 2).
4030 : P2 - The address of the first node address.
4031 : P3 - The address of the second node address if P1 = 2 or
4032 : blank if P1 = 1.
4033 : P4 - page register value for accessing the structure that
4034 : contains the node addresses.
4035
4036 : Implicites -
4037 : The summary counters: S.NREC, S.REC, S.LEN, S.COMP, S.BYTE,
4038 : and S.XFER
4039
4040 : Outputs - The summary table is updated.
4041
4042 : Calling procedure - CALL WRITES P1,P2(,P3)
4043
4044 : Side effects - The summary counters are cleared.
4045
4046 : Subordinate Routines -
4047 : CMPTWO - routine to compare two strings
4048
4049 : Register Usage -
4050 : R1 points to the current location in the summary table.
4051 : R2 points to the node to be updated's address.
4052 : R3 is scratch
4053 : R4 holds the second node to be updated address.
4054
4055 :--+
4056 033570          WRITES:::                ; see how many nodes to write
4057 033570          P$POP  temp             ; if only one, get address
4058 033574 023727 003110' 000001          cmp    temp,#1

```

```

4059 033602 001002          bne   5$           ; if two, get both addresses
4060 033604
4061 033606 000402          P$POP R2
4062 033610
4063
4064 033614          5$:  P$POP R2,R4      ; get page register value
4065
4066 033620 012701 100000    6$:  P$POP TEMP2
4067
4068 033624          10$: mov #statbl,R1      ; move statistical table address into R1
4069 033636 020127 126000    12$: CALL REMAP #OSTAB
4070 033642 001475          CMP   R1,#STAEND
4071 033644 005711          BEQ   25$          ; YES, that's all that can be done
4072 033646 001420          TST   (R1)         ; Is this spot empty then?
4073
4074          BEQ   15$          ; YES, go fill it then
4075 033650          ; Else is it equal to the current summary table entry
4076 033676          CALL  CMPEXT #OSTAB,R1,TEMP2,R2,#3
4077 033700 001416          P$POP R3
4078 033702 062701 000026    beq   20$          ; if yes, br
4079 033706 000746          add   #26,R1        ; else, point R1 to next entry
4080
4081 033710          15$: CALL MOVEXT TEMP2,R2,#OSTAB,R1,#3 ; copy node address into summary table
4082
4083 033736          20$: CALL REMAP #OSTAB
4084 033750 062701 000006    add   #6,R1          ; MOVEXT has changed memory mapping
4085 033754 063721 002770'   add   s.nrec,(R1)+ ; point R1 to data
4086 033760 063721 002766'   add   s.rec,(R1)+ ; update summary data, receives not complete
4087 033764 063721 002772'   add   s.len,(R1)+ ; receives complete
4088 033770 063721 002774'   add   s.comp,(R1)+ ; length errors
4089 033774 063721 002776'   add   s.byte,(R1)+ ; compare errors
4090 034000 103001          bcc   22$          ; bytes compared
4091 034002 005511          adc   (R1)          ; if overflow, increment next word
4092 034004 062701 000002    22$: add #2,R1
4093 034010 063721 003000'   add   s.xfer,(R1)+ ; point R1 to next data
4094 034014 103001          bcc   23$          ; bytes transferred
4095 034016 005511          adc   (R1)          ; if overflow, increment next word
4096 034020 062701 000002    23$: add #2,R1
4097 034024 005337 003110'   dec   temp          ; point R1 to next data
4098 034030 001414          beq   30$          ; decr no of nodes counter
4099 034032 010402          mov   R4,R2        ; if no more, exit
4100 034034 000671          br   10$          ; point R2 to next node
4101 034036          25$: printf #tabful,#summ ; and update summary data
4102 034062 005037 002770'   30$: clr  s.nrec        ; print table full message
4103 034066 005037 002766'   .    clr  s.rec
4104 034072 005037 002772'   .    clr  s.len
4105 034076 005037 002774'   .    clr  s.comp
4106 034102 005037 002776'   .    clr  s.byte
4107 034106 005037 003000'   .    clr  s.xfer
4108 034112          CALL  RETMEM        ; clear summary data counters
4109 034120          return          ; return memory to original mapping
4110

```

```

4112
4113 .sbttl BINDEC Convert a 32 bit binary number to decimal
4114
4115
4116 ;--+
4117 ; Functional Description:
4118 ; This subroutine converts a 32 bit binary number to
4119 ; a decimal number represented as an asciz string.
4120
4121 ; Inputs - P1 - The address of the first word of binary data
4122 ; bits 0-15. The second word, bits 16-31, is
4123 ; expected to immediately follow the first word.
4124
4125 ; Outputs - The ascii string will be located starting at DECSTR
4126
4127 ; Calling Procedure: CALL BINDEC P1
4128
4129 ; Side effects - none
4130
4131 ; Subordinate Routines - none
4132
4133 ; Register Usage - R1 points to bits 0-15 of binary data
4134 ; R2 points to bits 16-31 of binary data
4135 ; R3 points to the output string
4136 ; R4 points to the powers of 10 table
4137
4138 ;--+
4139 034122
4140 034122
4141 034124 010546 P$POP R1 : put address of binary word into R1
4142 034126 012137 003112' mov R5,-(SP)
4143 034132 011137 003114' mov (R1)+,temp1 : put low word in TEMP1
4144 034136 012703 034324' mov (R1),temp2 : put high word in TEMP2
4145 034142 012704 034254' mov #DECSTR,R3 : put address of ouput string into R3
4146 034146 012705 034256' mov #TENPWR,R4 : address of ten power table
4147 034152 012737 000012 034242' mov #10.,4$ : clear partial counter
4148 034160 005037 034340' 1$: clr part
4149 034164 161437 003112' 2$: sub (R4),temp1 : subtract 10 power
4150 034170 005637 003114' sbc temp2
4151 034174 161537 003114' sub (R5),temp2
4152 034200 002403 blt 3$ : branch if 10 power too large
4153 034202 005237 034340' inc part : else add 1 to partial
4154 034206 000766 br 2$ : loop
4155 034210 062437 003112' 3$: add (R4)+,temp1 : restore binary words
4156 034214 005537 003114' adc temp2 : and point R4 to next table entries
4157 034220 062437 003114' add (R4)+,temp2
4158 034224 022525 cmp (R5)+,(R5)+
4159 034226 052737 000060 034340' bis #'0,part : change partial to ascii
4160 034234 113723 034340' movb part,(R3)+ : and put into output string
4161 034240 005327 dec (PC)+ : have we done all 10 digits
4162 034242 000000 4$: .word 0 : if no, branch
4163 034244 001345 bne 1$ : if yes, terminate with zero
4164 034246 105023 clrb (R3)+
4165 034250 012605 mov (SP)+,RS
4166 034252 return
4167
4168 034254 145000 TENPWR: 145000 : 1.0 E09

```

4169 034256	035632	35632	
4170 034260	160400	160400	: 1.0 E08
4171 034262	002765	2765	
4172 034264	113200	113200	: 1.0 E07
4173 034266	000230	230	
4174 034270	041100	041100	: 1.0 E06
4175 034272	000017	17	
4176 034274	103240	103240	: 1.0 E05
4177 034276	000001	1	
4178 034300	023420	23420	: 1.0 E04
4179 034302	000000	0	
4180 034304	001750	1750	: 1.0 E03
4181 034306	000000	0	
4182 034310	000144	144	: 1.0 E02
4183 034312	000000	0	
4184 034314	000012	12	: 1.0 E01
4185 034316	000000	0	
4186 034320	000001	1	: 1.0 E00
4187 034322	000000	0	
4188			
4189 034324		DECSTR::BLKB 12.	: 12 bytes for asciz output string
4190 034340	000000	PART:: .WORD 0	: partial counter
4191			

```

4193
4194 .SBTTL COMMAND LINE TRAVERSE ROUTINES
4195
4196
4197
4198
4199
4200
4201
4202
4203
4204
4205
4206
4207
4208
4209
4210
4211
4212 034342
4213 034342 013704 001260'
4214 034346 013703 001262'
4215 034352 121327 000003
4216 034356 003405
4217 034360 105714
4218 034362 001441
4219 034364 121327 000013
4220 034370 003023
4221 034372 111301
4222 034374 006301
4223 034376 016101 034412'
4224 034402 062701 034412'
4225 034406 004711
4226 034410 000760
4227
4228
4229 034412 000114
4230 034414 000134
4231 034416 000152
4232 034420 000162
4233 034422 000204
4234 034424 000270
4235 034426 000612
4236
4237 034430 000000
4238 034432 000270
4239 034434 000256
4240 034436 000656
4241
4242
4243
4244 034440 121314
4245 034442 001403
4246 034444 004737 034510'
4247 034450 000740
4248 034452 005204
4249 034454 004737 034470'

;++
;      P$TRV SUBROUTINE
;
;PARSE THE COMMAND LINE SUBROUTINE
;TAKE ACTIONS (VIA ACTION TREE) AS PARSING LINE
;PARSING DIRECTIONS FROM "CLI PARSING NODES"
;REGS USED:
;
;      R1,R5=SCRATCH
;      R2=ACTION CODE PARAMETER FROM TREE
;      R3=PARSE TREE POINTER
;      R4=INPUT STRING POINTER
;
;CALLING SEQUENCE:
;      JSR      PC,P$TRV
;
;--
;P$TRV:::
;      MOV      P$BUFA,R4
;      MOV      P$TREE,R3
;P$TR5:   CMPB    (R3),#3
;          BLE     5$
;          TSTB    (R4)
;          BEQ     P$EXIT
;          CMPB    (R3),#11.
;          BGT     20$
;          MOVB    (R3),R1
;          ASL     R1
;          MOV     10$(R1),R1
;          ADD     #10$,R1
;          JSR     PC,(R1)
;          BR      P$TR5
;
;      ;SEE IF ONE OF FIRST THREE SPECIAL CODES
;      ;IF YES, DON'T CHECK INPUT STRING
;      ;SEE IF ANY CHARS LEFT IN INPUT STRING
;      ;BR IF NO
;      ;SEE IF SPECIAL CLI CHAR CODE OR ASCII
;      ;BR IF REGULAR ASCII CHAR.
;      ;GET SPECIAL CHAR CODE INTO R5
;      ;BUILD TRAVERSE ROUTINE ADDRESS
;      ;JSR TO SPECIAL CLI TRAVERSE ROUTINE
;      ;GO SEE IF MORE OF STRING LEFT
;
;10$:    .WORD    TRVERR-10$
;          .WORD    TRVEXI-10$
;          .WORD    TRVBR-10$
;          .WORD    TRVBIF-10$
;          .WORD    TRVSPA-10$
;          .WORD    TRVNUM-10$
;          .WORD    TRVALP-10$
;          .WORD    0
;          .WORD    TRVOCT-10$
;          .WORD    TRVDEC-10$
;          .WORD    TRVSTR-10$
;
;      ;TRAVERSE TABLE FOR "CLI FUNCTIONS"
;      ;1
;      ;2
;      ;3
;      ;4
;      ;5
;      ;6
;      ; *** NEW .WORD TRVALN-10$ ***
;      ;8
;      ;9
;      ;10
;
;NOT A SPECIAL CODE
;
;20$:    CMPB    (R3),(R4)
;          BEQ     22$
;          JSR     PC,TRVBRC
;          BR      P$TR5
;22$:    INC     R4
;          JSR     PC,TRVACT
;
;      ;SEE IF FIRST CHAR OF STRING IS A MATCH
;      ;BR IF A MATCH
;      ;IF NOT A MATCH, GO TAKE MISS BRANCH
;      ;THEN GO BACK PT'G TO MISS NODE
;      ;IF A MATCH, INCR. CHAR POINTER
;      ;GO DO ACTION DEFINED BY

```

COMMAND LINE TRAVERSE ROUTINES

4250 034460 062703 000004	ADD #4,R3	: ACTION CODE IN CLI NODE, THEN
4251	BR P\$TR5	: ADJUST PTR TO NEXT CLI NODE
4252 034464 000732		
4253		
4254 034466 000207	P\$EXIT: RTS . PC	:RETURN FROM PARSER
4255		
4256		
4257		
4258		
4259 034470 116302 000001	:GOTO USER ACTION ROUTINE	
4260 034474 042702 177400	TRVACT: MOVB 1(R3),R2	:GET ACTION CODE FROM CLI NODE
4261 034500 013701 001264	BIC #177400;R2	:CLEAR ANY SIGN EXTENSION
4262 034504 004711	MOV P\$ACT,R1	:GET ADDRESS OF CLI ACTION ROUTINE
4263 034506 000207	JSR PC,(R1)	:GO DO ACTION DEFINED BY CODE
	RTS PC	:RETURN TO CALLING CODE
4264		
4265		
4266 034510 016301 000002	:TAKE BRANCH IN TREE	
4267 034514 060103	TRVBRC: MOV 2(R3),R1	:GET BRANCH DISPLACEMENT FROM TREE
4268 034516 000207	ADD R1,R3	: AND POINT R3 TO THE "MISS" NODE
	RTS PC	: RETURN TO P\$TRV
4269		
4270		
4271 034520 062703 000004	:NO BRANCH TAKEN	
4272 034524 000207	TRVN0B: ADD #4,R3	:THINGS OK, UPDATE R3 TO POINT TO NEXT
	RTS PC	: NODE AND RETURN TO P\$TRV
4273		
4274		
4275		
4276 034526 004737 034470'	:ERROR HANDLING	
4277 034532 112737 177777 001301'	TRVERR: JSR PC,TRVACT	:TAKE ERROR ACTION
	MOVB #-1,P\$GDBD	:SET ERROR RETURN FLAG
	TST (SP)+	:GET RID OF "JSR PUSH TO TRVERR"
4279 034542 000137 034466'	JMP P\$EXIT	:RETURN DIRECT TO EXIT OF P\$TRV ROUTINE
4280		
4281		
4282 034546 004737 034470'	:EXIT ACTION CODE	
4283 034552 105037 001301'	TRVEXI: JSR PC,TRVACT	:TAKE EXIT ACTION
	CLRB P\$GDBD	:SET GOOD/BAD FLAG TO "SUCCESS (0)"
	TST (SP)+	:GET RID OF "JSR PUSH TO TRVEXI"
4285 034560 000137 034466'	JMP P\$EXIT	:RETURN DIRECT TO EXIT OF P\$TRV ROUTINE
4286		
4287		
4288 034564 004737 034470'	:BRANCH ACTION CODE	
4289 034570 000137 034510'	TRVBR: JSR PC,TRVACT	:GO TAKE BRANCH ACTION
	JMP TRVBRC	
4290		
4291		
4292 034574 004737 034470'	:BRANCH-IF ACTION CODE	
4293 034600 105737 001301'	TRVBIF: JSR PC,TRVACT	
	TSTB P\$GDBD	:SEE IF P\$GDBD SET OR CLEARED BY ACTION
	BEQ 1\$:IF CLEAR FALL THRU TO NEXT NODE
4294 034604 001402	JMP TRVBRC	:ELSE TAKE THE "MISS" BRANCH
4295 034606 000137 034510'	1\$: JMP TRVN0B	:JUST UPDATE TO NEXT NODE IF THINGS OK
4296 034612 000137 034520'		
4297		
4298		
4299 034616 005001	:SPACE ACTION CODE	
4300 034620 121427 000011	TRVSPA: CLR R1	:CLEAR "SPACE OR TAB FOUND" FLAG
	1\$: CMPB (R4),#11	:SEE IF CHAR. IN CMD LINE= TAB
4301 034624 001003	BNE 2\$:BR IF NO, NOT A TAB
4302 034626 005204	INC R4	:INC INPUT STRING POINTER
4303 034630 005201	INC R1	:INDICATE A TAB FOUND
4304 034632 000772	BR 1\$:GO CHECK NEXT CHAR
4305		
4306 034634 121427 000040	2\$: CMPB (R4),#40	:SEE IF CHAR. IN CMD LINE= SPACE

4307 034640 001003		BNE	10\$:BR IF NO, NON-SPACE OR NON-TAB CHAR.
4308 034642 005204		INC	R4	:INC INPUT STRING POINTER
4309 034644 005201		INC	R1	:INDICATE A SPACE FOUND
4310 034646 000764		BR	1\$:GO CHECK NEXT CHAR
4311 034650 005701	10\$:	TST	R1	:SEE IF ANY SPACES OR TABS FOUND
4312 034652 001404		BEQ	15\$:BR IF NO, TAKE NO ACTION
4313 034654 004737 034470'		JSR	PC.TRVACT	:GO TAKE ACTION IF ANY FOUND
4314 034660 000137 034520'		JMP	TRVN0B	:JUST GO UPDATE R3 TO NEXT NODE IF OK
4315 034664 000137 034510'	15\$:	JMP	TRVBRC	:TAKE BRANCH (MISS) IF NONE FOUND
4316				
4317				
4318 034670 012737 000012 001272' TRVDEC:		MOV	#10.,P\$RADX	:USE DECIMAL AS RADIX AND ASSUME +
4319 034676 000137 034710'		JMP	TRVNMA	
4320 034702	TRVOCT: ;(SAME AS TRVNUM SINCE DEFAULT RADIX IS OCTAL)			
4321 034702 012737 000010 001272' TRVNUM:	MOV	#8.,P\$RADX	:USE OCTAL AS RADIX AND ASSUME +	
4322 034710 TRVNMA: PUSH R5				
4323 034712 005001 CLR R1				:CLEAR DIGIT COUNTER
4324 034714 121427 000053 CMPB (R4),#'+				:SEE IF THERE'S A + SIGN THERE
4325 034720 001001 BNE 10\$: BR IF NO
4326 034722 000406 BR 11\$: ELSE P\$RADX ALREADY SAYS +, JUST BR
4327 034724 121427 000055 10\$:	CMPB	(R4),#'-		:SEE IF THERE'S A - SIGN THERE
4328 034730 001004 BNE 1\$: BR IF NO
4329 034732 112737 177777 001273' MOVB #-1,P\$RADX+1				:SET "MINUS FLAG" (HI BYTE OF P\$RADX)
4330 034740 005204 11\$:	INC	R4		:BUMP R4 TO POINT TO FIRST CHAR
4331				
4332 034742 121427 000060 1\$:	CMPB	(R4),#60		:SEE IF CHAR. LESS THAN A "0"
4333 034746 002434 BLT 2\$:BR IF YES (NOT NUMERIC)
4334 034750 121427 000067 CMPB (R4),#67				:SEE IF CHAR. GREATER THAN A "7"
4335 034754 003426 BLE 13\$: BR IF YES
4336 034756 123727 001272' 000012 CMPB P\$RADX,#10.				:SEE IF IN DECIMAL MODE
4337 034764 001417 BEQ 12\$: BR IF YES (CAN USE HIGHER LIMIT)
4338 034766 121427 000071 CMPB (R4),#71				:SEE IF DIGIT WAS A 8 OR 9
4339 034772 003022 BGT 2\$:BR IF NON-NUMERIC
4340 034774 PRINTF #CLIBRX				:ELSE WAS A 8 OR 9 WHEN IN OCTAL RADIX
4341 035014 112737 177777 001301' MOVB #-1,P\$GDBD				:SET ERROR RETURN FLAG
4342 035022 000475 BR 5\$: PRINT ERROR AND TAKE MISS
4343				
4344 035024 121427 000071 12\$:	CMPB	(R4),#71		:SEE IF CHAR. GREATER THAN A "9"
4345 035030 003003 BGT 2\$:BR IF YES (NOT NUMERIC)
4346 035032 005204 13\$:	INC	R4		:UPDATE CMD LINE PTR TO NEXT CHAR.
4347 035034 005201 INC R1				:INDICATE A NUMERIC FOUND
4348 035036 000741 BR 1\$:GO LOOK AT NEXT CHAR.
4349				
4350 035040 005701 2\$:	TST	R1		:SEE IF FOUND ANY NUMERICS
4351 035042 001465 BEQ 5\$:BR IF NO, TAKE "MISS" BRANCH
4352 035044 010405 MOV R4,R5				:GET POINTER TO START OF NUMERIC STRING
4353 035046 160105 SUB R1,R5				
4354 035050 005037 001270' CLR P\$NUM				:CLEAR LOC. WHERE VALUE WILL BE STORED
4355 035054 112502 3\$:	MOVB	(R5)+,R2		:GET ASCII CHAR AND CONVERT IT TO A #
4356 035056 162702 000060 SUB #60,R2				
4357 035062 006337 001270' ASL P\$NUM				
4358 035066 103440 BCS 7\$				
4359 035070 013737 001270' 001266' MOV P\$NUM,P\$CNT				
4360 035076 006337 001270' ASL P\$NUM				
4361 035102 103432 BCS 7\$				
4362 035104 006337 001270' ASL P\$NUM				
4363 035110 103427 BCS 7\$				

COMMAND LINE TRAVERSE ROUTINES

4364 035112	123727	001272'	000012	CMPB	P\$RADX,#10.	:SEE IF DECIMAL RADIX	
4365 035120	001004			BNE	4\$:BR IF NOT EQUAL	
4366 035122	063737	001266'	001270'	ADD	P\$CNT,P\$NUM	:ERROR IF NUMBER TOO BIG	
4367 035130	103417			BCS	7\$		
4368 035132	060237	001270'		4\$:	ADD	R2,P\$NUM	
4369 035136	103414			BCS	7\$:ERROR IF NUMBER TOO BIG	
4370 035140	005301			DEC	R1		
4371 035142	001344			BNE	3\$		
4372 035144	105737	001273'		TSTB	P\$RADX+1	:SEE IF NUM WAS PRECEDED BY A - SIGN	
4373 035150	001402			BEQ	15\$: BR IF NO	
4374 035152	005437	001270'		NEG	P\$NUM	: ELSE NEGATE THE NUMBER BEFORE LEAVING	
4375 035156				15\$:	POP	R5	
4376 035160	004737	034470'		JSR	PC,TRVACT	:RESTORE R5	
4377 035164	000137	034520'		JMP	TRVN0B	:SINCE NUMERIC FOUND, GO TAKE ACTION	
4378						:GO POINT R3 TO NEXT NODE	
4379 035170				7\$:	PRINTF	#CLINBG	:PRINT NUMBER TOO BIG ERROR
4380 035210	112737	177777	001301'	MOV	#-1,P\$GDBD	:SET ERROR RETURN FLAG	
4381 035216				5\$:	POP	R5	:RESTORE R5
4382 035220	000137	034510'		JMP	TRVBRC	:TAKE "MISS" BRANCH	
4383							
4384							
4385 035224	005001			TRVALP:	CLR	R1	:CLEAR ALPHA FOUND FLAG
4386 035226	121427	000101		1\$:	CMPB	(R4),#101	:SEE IF CHAR. LESS THAN A "A"
4387 035232	002406			BLT	2\$:BR IF YES (NOT ALPHA)	
4388 035234	121427	000132		CMPB	(R4),#132	:SEE IF CHAR. GREATER THAN A "Z"	
4389 035240	003003			BGT	2\$:BR IF YES (NOT ALPHA)	
4390 035242	005204			INC	R4	:UPDATE CMD LINE PTR TO NEXT CHAR	
4391 035244	005201			INC	R1	:INDICATE AN ALPHA WAS FOUND	
4392 035246	000767			BR	1\$:GO LOOK AT NEXT CHAR.	
4393 035250	005701			2\$:	TST	R1	:SEE IF ANY ALPHA'S WERE FOUND
4394 035252	001404				BEQ	3\$:BR IF NO
4395 035254	004737	034470'			JSR	PC,TRVACT	:IF ANY FOUND TAKE ACTION
4396 035260	000137	034520'			JMP	TRVN0B	:THEN UPDATE R3 TO NEXT NODE -NO BRANCH
4397 035264	000137	034510'		3\$:	JMP	TRVBRC	:NONE FOUND, TAKE MISS BRANCH
4398							
4399							
4400							
4401 035270				TRVSTR:	PUSH	R5	:SAVE R5
4402 035272	010401				MOV	R4,R1	:POINT R1 TO CMD STRING
4403 035274	010305				MOV	R3,R5	
4404 035276	062705	000006			ADD	#6,R5	:POINT R5 TO MATCH STRING FROM CLI NODE
4405 035302	005037	001266'			CLR	P\$CNT	:CLEAR CHAR MATCH COUNT
4406 035306	105715			2\$:	TSTB	(R5)	:SEE IF END OF MATCH STRING YET
4407 035310	001411				BEQ	10\$:BR IF YES
4408 035312	105711				TSTB	(R1)	:SEE IF END OF CMD LINE YET
4409 035314	001407				BEQ	10\$:BR IF YES
4410 035316	121115				CMPB	(R1),(R5)	:SEE IF CHARACTERS MATCH
4411 035320	001005				BNE	10\$:BR IF NO
4412 035322	005237	001266'			INC	P\$CNT	:MATCH -INCREMENT MATCH COUNT
4413 035326	005201				INC	R1	:UPDATE STRING POINTERS
4414 035330	005205				INC	R5	
4415 035332	000765				BR	2\$:BR TO CONTINUE CHECKING CHARS.
4416							
4417 035334	005737	001266'		10\$:	TST	P\$CNT	:WHEN DONE SEE IF ANY MATCHES FOUND
4418 035340	001407				BEQ	15\$:BR IF NO, GO TAKE THE MISS BRANCH
4419 035342	010104				MOV	R1,R4	:POINT CMD POINTER TO END OF STRING &
4420 035344					POP	R5	:RESTORE R5

```

4421 035346 004737 034470'          JSR    PC,TRVACT      ;IF A MATCH FOUND, GO DO MATCH ACTION
4422 035352 066303 000004          ADD    4(R3),R3       ;UPDATE R3 TO NEXT NODE (NO BRANCH)
4423 035356 000207          RTS    PC           ;(NO RETURN THRU TRVN08 SINCE DIFFERENT
4424                               ;DISPLACEMENT DUE TO MATCH STRING)
4425 035360          15$: POP   R5           ;RESTORE R5
4426 035362 000137 034510'          JMP    TRVBRC      ;GO TAKE BRANCH
4427                               ;(PARSED OK). -1 IF ILL CMD.....
4428
4429
4430          :---+ TRVADR          TRAVERSE COMMAND LINE INPUT ADDRESS
4431
4432          : THIS ROUTINE IS CALLED BY TWO DIFFERENT ACTION ROUTINES. THE
4433          : NODE ACTION ROUTINE CALLS IT TO PARSE THROUGH THE NODE
4434          : ADDRESS INPUT BY THE OPERATOR. THE OPRSEL ACTION ROUTINE
4435          : CALLS TRVADR TO PARSE THROUGH THE "OPERATOR SELECTED" MESSAGE
4436          : WHICH HAS BEEN INPUT IN THE COMMAND LINE. FOR A NODE ADDRESS,
4437          : THE ROUTINE LOOKS FOR A '/' AS A DELIMETER FOR THE ADDRESS,
4438          : AND REPLACES THE / WITH A NULL BYTE FOR USE BY THE ADDRESS
4439          : PACKING ROUTINE. WHEN CALLED BY THE OPRSEL ROUTINE, A '"'
4440          : IS EXPECTED AS THE DELIMETER FOR THE OPERATOR SELECTED MESSAGE.
4441          : IF A NULL STRING IS ENTERED, AN ERROR MESSAGE IS PRINTED.
4442
4443          : INPUTS -          R4 - POINTS TO THE BEGINING OF THE ADDRESS
4444          :                      OR MESSAGE IN THE COMMAND LINE
4445          : OUTPUTS -          SUMMARIZED IN TABLE BELOW
4446
4447          : COMMAND LINE          OUTPUTS
4448          : INPUT CONDITION ! P$GDBD ! R4 POINTS TO ! CFLAG CONTAINS ! P$MERR
4449          :-----+-----+-----+-----+-----+-----+
4450          : ILLEGAL CHAR.    ! -1    ! ILL. CHAR.    !          ! N/A
4451          : ADR./ASSIST     ! 0     ! END OF LINE ! CASIST   ! N/A
4452          : ADR./TARGET      !          ! END OF LINE ! CTARGT   ! N/A
4453          : ADR./          ! 0     !          !          !
4454          : ADR.          !          !          !
4455          : ADR./CHAR. &R    !          !
4456          : "OPR SEL/CHAR." !
4457          : OTHER THAN "A" ! -1    ! /          ! CTARGT   ! N/A
4458          : "T" OR BLANK    !          !
4459          : "          ! 0     ! CHAR. AFTER " !          ! -1
4460          : "OPR SEL"        ! 0     ! CHAR. AFTER " ! OPRSEL   ! 0
4461
4462          : CALLING PROCEDURE - JSR PC,TRVADR
4463          : REGISTER USAGE - R1 IS USED AS A COUNTER TO REPORT ERROR MESSAGES
4464          :                      IF NULL STRINGS ARE ENTERED.
4465          : R4 POINTS TO THE NEXT CHAR. IN THE COMMAND LINE
4466
4467
4468          :---+ TRVADR: CLR   R1          :CLEAR HEX DIGIT FOUND FLAG
4469 035366 005001          1$: CMPB  (R4),#0  :SEE IF NUL CHAR.
4470 035370 121427 000000          BEQ   20$      :IF YES, RETURN
4471 035374 001435          CMPB  (R4),#40     :SEE IF ILLEGAL CHARACTER
4472 035376 121427 000040          BLT   10$      :IF YES; BRANCH TO ERROR ROUTINE
4473 035402 002426          BNE   4$       :branch if not a space
4474 035404 001002          INC    R4       :skip space
4475 035406 005204          BR    1$       :check next character
4476 035410 000767          4$: CMPB  (R4),#42     :SEE IF CHAR. IS A '"'
4477 035412 121427 000042

```

```

4478 035416 001007          BNE   6$           ; branch if not
4479 035420 112714 000000      MOVB  #0.(R4)    ; ELSE, REPLACE "" WITH NULL
4480 035424 005204          INC    R4           ; point R4 past "" in input string
4481 035426 012737 000006 002024'      MOV    #OPRSEL,CFLAG ; set operator selected flag ...
4482 035434 000501          BR    50$          ; ... and take off
4483 035436 121427 000057          CMPB  (R4),#57  ; SEE IF CHAR. IS A "/"
4484 035442 001420          BEQ   30$          ; BRANCH IF YES
4485 035444 121427 000132          CMPB  (R4),#132 ; SEE IF CHAR. GREATER THAN "F"
4486 035450 003003          BGT   10$          ; IF YES, ILLEGAL CHAR.
4487 035452 005204          INC    R4           ; UPDATE CMD LINE POINTER TO NEXT CHAR.
4488 035454 005201          INC    R1           ; INDICATE A VALID CHAR. FOUND
4489 035456 000744          BR    1$           ; LOOK AT NEXT CHAR.
4490 035460 112737 177777 001301' 10$:      MOVB  #-1,P$GDBD ; SET ERROR FLAG
4491 035466 000464          BR    50$          ; RETURN
4492 035470 005701          20$: TST   R1           ; SEE IF VALID CHARACTERS FOUND
4493 035472 001772          BEQ   10$          ; IF NO, ILLEGAL CHAR.
4494 035474 012737 000000 002024' 25$:      MOV    #CTARGT,CFLAG ; SET TARGET FLAG
4495 035502 000456          BR    50$          ; RETURN
4496 035504 005701          30$: TST   R1           ; SEE IF VALID CHARACTERS FOUND
4497 035506 001764          BEQ   10$          ; IF NO, ILLEGAL CHAR.
4498 035510 105737 001305'          TSTB  P$TEXT  ; is it text?
4499 035514 001027          BNE   40$          ; branch if it is
4500 035516 112714 000000          MOVB  #0.(R4)  ; IF YES, REPLACE "/" WITH NULL CHAR.
4501 035522 005204          INC    R4           ; UPDATE CMD. LINE POINTER TO NEXT CHAR.
4502 035524 121427 000000          CMPB  (R4),#0  ; IS NEXT CHAR. NULL
4503 035530 001761          BEQ   25$          ; IF YES, TAKE DEFAULT OF TARGET
4504 035532 121427 000101          CMPB  (R4),#A  ; IS NEXT CHAR. "A"
4505 035536 001412          BEQ   35$          ; IF YES, BR 35$
4506 035540 121427 000124          CMPB  (R4),#T  ; IS NEXT CHAR. "T"
4507 035544 001753          BEQ   25$          ; IF YES, SET TARGET FLAG
4508 035546 112737 177777 001301'          MOVB  #-1,P$GDBD ; ELSE, SET ERROR FLAG,
4509 035554 005304          DEC    R4           ; READJUST COMMAND LINE POINTER
4510 035556 112714 000057          MOVB  #'/,.(R4) ; AND REPLACE / IN CMD LINE TO FIX ERROR
4511 035562 000744          BR    25$          ; SET TARGET FLAG AND RETURN
4512 035564 012737 000001 002024' 35$:      MOV    #CASIST,CFLAG ; SET ASSIST FLAG
4513 035572 000422          BR    50$          ; SEE IF ANY CHARACTERS TYPED
4514 035574 005701          40$: TST   R1           ; IF NO, BRANCH TO 45$
4515 035576 001404          BEQ   45$          ; SET OPERATOR SELECTED FLAG
4516 035600 012737 000006 002024'          MOV    #OPRSEL,CFLAG ; RETURN
4517 035606 000414          BR    50$          ; PRINT NULL STRING ERROR MESSAGE
4518 035610          45$: PRINTF #NULSTR ; SET OPER. SELECTED MSG. ERROR FLAG
4519 035630 112737 177777 001304'          MOVB  #-1,P$MERR ; MOVE CMD. LINE POINTER TO NEXT CHAR.
4520 035636 005204          INC    R4           ; RETURN
4521 035640 000207          50$: RTS   PC           ; -----
4522
4523
4524
4525 .SBTTL REPORT CODING SECTION
4526
4527
4528 ;+
4529 ; THE REPORT CODING SECTION CONTAINS THE
4530 ; "PRINTS" CALLS THAT GENERATE STATISTICAL REPORTS.
4531 ;-
4532
4533 035642          BGNRPT
4534

```

```
4536 ;*****  
4537 : THIS SECTION, WHICH IS OPTIONAL, CONTAINS THE CODE FOR PRINTING  
4538 : STATISTICAL INFORMATION GATHERED BY THE DIAGNOSTIC. IT IS  
4539 : EXECUTED BY THE OPERATOR COMMAND "PRINT" OR BY THE MACRO CALL  
4540 : "DORPT". USE THE PRINTS MACRO TO PRINT THE INFORMATION.  
4541 : USE FORMAT STATEMENTS AS IN THE PRINTB/PRINTX MACROS. IT IS  
4542 : THE PROGRAMMER'S RESPONSIBILITY TO DEVISE AND IMPLEMENT THE  
4543 : FORM AND CONTENT OF THE STATISTICS.  
4544 ;*****  
4545 035642 004737 042674' JSR PC,ACTSUM  
4546 035646 EXIT RPT  
4547 ;*****  
4548 : INSERT LOCAL STORAGE THAT IS USED ONLY  
4549 : DURING THE REPORT SECTION.  
4550 ;*****  
4551 ;*****  
4552 : INSERT MESSAGES THAT ARE USED ONLY  
4553 : DURING THE REPORT SECTION.  
4554 ;*****  
4555 ;*****  
4556 ;*****  
4557 ;*****  
4558 ;*****  
4559 ;*****  
4560 ;*****  
4561 .EVEN  
4562  
4563  
4564 035652 ENDRPT
```

```
4566          .SBTTL PROTECTION TABLE
4567
4568
4569      ;++: THIS TABLE IS USED BY THE RUNTIME SERVICES
4570      ; TO PROTECT THE LOAD MEDIA.
4571      ;--
4572
4573 035654          BGNPROT
4574
4575 035654 177777      -1          :OFFSET INTO P-TABLE FOR CSR ADDRESS
4576 035656 177777      -1          :OFFSET INTO P-TABLE FOR MASSBUS ADDRESS
4577 035660 177777      -1          :OFFSET INTO P-TABLE FOR DRIVE NUMBER
4578
4579 035662          ENDPROT
4580
4582      ;-----: INSERT BYTE OFFSET FOR DATA NOTED IN COMMENTS ABOVE. (OFFSET
4583      ;      REFERS TO THE NUMBER OF BYTES FROM THE BEGINNING OF A PTABLE
4584      ;      ENTRY TO THE ITEM IN QUESTION.) IF THE PARTICULAR
4585      ;      ITEM DOES NOT APPLY, LEAVE ENTRY AS -1. WHEN THE RUNTIME
4586      ;      SERVICES EXECUTES A GPHARD, IT USES THESE OFFSETS (IF NOT
4587      ;      SET TO -1) TO GET THE ITEMS AND COMPARE WITH THOSE SAVED
4588      ;      IN THE XXDP+ MONITOR. IF THE UNIT BEING REQUESTED MATCHES THE
4589      ;      LOAD DEVICE, THE RUNTIME SERVICES RETURN AN INCOMPLETE FLAG ON
4590      ;      THE GPHARD.
4591
4592      ;-----
```

```
4595      .SBTTL INITIALIZE SECTION
4596
4597      ;+
4598      ; THE INITIALIZE SECTION CONTAINS THE CODING THAT IS PERFORMED
4599      ; AT THE BEGINNING OF EACH PASS.
4600      ;-
4601
4602 035662      BGNINIT
4603
4604
4605      ; THE INITIALIZE CODE IS EXECUTED UNDER FIVE CONDITIONS. THERE
4606      ; ARE SUPERVISOR EVENT FLAGS THAT ARE USED TO LET THE
4607      ; DIAGNOSTIC KNOW UNDER WHICH CONDITION THE EXECUTION IS TAKING
4608      ; PLACE. THE EVENT FLAGS ARE READ USING THE "READEF" MACRO.
4609      ; THE CONDITIONS UNDER WHICH THE INIT CODE IS EXECUTED AND THE
4610      ; CORRRESPONDING EVENT FLAGS ARE:
4611      ;     START COMMAND          EF.START
4612      ;     RESTART COMMAND        EF.RESTART
4613      ;     CONTINUE COMMAND       EF.CONTINUE
4614      ;     POWERDOWN/POWERUP      EF.PWR
4615      ;     NEW PASS               EF.NEW
4616
4617      ; EXAMPLE OF EVENT FLAG USE:
4618      ;     READEF #EF.START
4619      ;     BCOMPLETE      STARTCODE
4620      ; DURING THE INIT CODE, USE THE "GPHARD" MACRO TO OBTAIN P-TABLE
4621      ; INFORMATION FOR DEVICE TESTING. GET ONE UNIT'S INFORMATION IF
4622      ; THIS IS A SEQUENTIAL DIAGNOSTIC. GET INFORMATION ON ALL
4623      ; UNITS AVAILABLE FOR TESTING IF THIS IS AN EXERCISER. THE NUMBER
4624      ; OF UNITS AVAILABLE IS IN A HEADER LOCATION: "L$UNIT".
4625
4626
4627      ;--+
4628      ; Functional Description:
4629      ; This routine performs all initialization functions necessary
4630      ; to run the diagnostic. In sequential order, the functions
4631      ; executed are:
4632
4633      ; 1.) determine how we got into the INIT code -- START, RESTART,
4634      ;     CONTINUE, or NEW PASS. The rest of these steps are all
4635      ;     done for a START. For RESTART and CONTINUE
4636
4637      ; 2.) set up the two stacks that the program uses -- PARAMETER
4638      ;     and MACHINE stacks
4639
4640      ; 3.) interrogate DRS for the amount of free memory available
4641      ;     and save the information
4642
4643      ; 4.) set up the system clock information
4644
4645      ; 5.) set DELUA/DEUNA interrupt service routine address and
4646      ;     vector
4647
4648      ; 6.) set up addresses of CSRs
4649
4650      ; 7.) Find out what kind of device we are running on. This
4651      ;     information is contained in PCSR1 <6:4>
4652      ;     --> 000 = DEUNA
4653      ;           001 = DELUA
```

```

4654          ;                                8.) Call MEMMAP to format extended memory
4655          ;                                9.) set processor priority to ZERO
4656          ;                                10.) CALL UNAINI to initialize the device we are running on
4657          ;                                11.) print out header information
4658          ;                                12.) setup system clock interrupt service routine address and
4659          ;                                vector and enable clock
4660          ;
4661          ;
4662          ;
4663          ;
4664          ;
4665          ;
4666          ;
4667          : Inputs - none
4668          ;
4669          : Outputs - A header message will be printed
4670          ;
4671          : Calling Procedure: Invoked by the DRS at either a START, RESTART, or CONTINUE
4672          ;
4673          : Side Effects - listed above
4674          ;
4675          : Subordinate Routines -
4676          ;      UNAINI - initialize the DELUA/DEUNA
4677          ;      FUNCT - perform an ancillary port command
4678          ;      DEVSTOP - stop the DELUA/DEUNA
4679          ;
4680          : Register Usage -
4681          ;      R2,R3 - scratch
4682          ;
4683          ;
4684          ;---+
4685 035662    INIT:
4686 035662 022737 000020 002024'    CMP    #CEXIT.CFLAG      ;SEE IF EXIT COMMAND TYPED
4687 035670 001004      BNE    INIT1      ;IF NO, DO INIT CODE
4688 035672 005037 002024'    CLR    CFLAG      ;ELSE, CLEAR EXIT FLAG
4689 035676 000137 037276'    JMP    INICLN     ;EXIT INIT CODE
4690 035702      READEF #EF.START      ;IF HERE BECAUSE OF "START", DO INIT
4691 035710      BCOMPLETE START
4692 035712      READEF #EF.RESTART
4693 035720      BNCOMPLETE $0
4694 035722 000137 037214'    JMP    RESTRT      ;IF HERE BECAUSE OF "RESTART", DO SOME INIT
4695 035726      READEF #EF.CONTINUE
4696 035734      BNCOMPLETE 10$      ;IF HERE BECAUSE OF "CONTINUE", EXIT
4697 035736 000137 037214'    JMP    RESTRT
4698 035742      READEF #EF.NEW      ;IF HERE ON NEW PASS, SKIP SOME INIT
4699 035750      BNCOMPLETE 15$      ;
4700 035752 000137 037250'    JMP    NEW
4701 035756 000137 037276'    15$:   JMP    INICLN      ;IF DON'T KNOW WHY WE'RE HERE, EXIT
4702 035762      START: I$STACK #STACK5.SP      ;SET PARAMETER STACK POINTER
4703 035770      MEMORY FRESIZ      ;GET FREE MEMORY INFO
4704 035776 013737 002134' 002136'    MOV    FRESIZ,FREMEM  ;SIZE OF FREE MEMORY IN FRESIZ
4705 036004 062737 000002 002136'    ADD    #2,FREMEM    ;START OF FREE MEMORY IN FREMEM
4706 036012 012702 002026'    MOV    #CLKCSR,R2      ;SETUP R2 AS A PRT. TO CLOCK INFO. BLOCK
4707 036016      CLOCK L,R1      ;GET LINE CLOCK INFO
4708 036026      BNCOMPLETE 20$      ;IF NONE, SEE IF P CLOCK PRESENT
4709 036030 004737 027014'    JSR    PC,CLKSET    ;SET UP CLOCK INFO TABLE AND VECTOR
4710 036034 012737 000100 002036'    MOV    #LCLKEN,CLKEN  ;SET UP THE ENABLE LINE CLOCK DATA

```

```

4711 036042 000430
4712 036044
4713 036054
4714 036056 004737 027014'
4715 036062 062737 000002 002026'
4716 036070 012777 001600 143730
4717 036076 162737 000002 002026'
4718 036104 012737 000111 002036'
4719 036112 000404
4720
4721 036114
4722
4723 036124
4724 036134
4725 036136 000137 037276'
4726
4727 036142 012137 002126'
4728 036146 012137 002130'
4729 036152 012137 002132'
4730 036156
4731 036204 013737 002126' 002106'
4732 036212 013737 002106' 002110'
4733 036220 062737 000002 002110'
4734 036226 013737 002110' 002112'
4735 036234 062737 000002 002112'
4736 036242 013737 002112' 002114'
4737 036250 062737 000002 002114'
4738
4739 036256 013703 002110'
4740 036262 011302
4741 036264 042702 177617
4742
4743 036270 010237 000524'
4744
4745
4746 036274
4747
4748 036302 005037 002770'
4749 036306 005037 002766'
4750 036312 005037 002772'
4751 036316 005037 002774'
4752 036322 005037 002776'
4753 036326 005037 003000'
4754
4755 036332 013737 002034' 002044'
4756 036340
4757 036366 013777 002036' 143432
4758 036374
4759 036402
4760
4761
4762
4763
4764
4765 036410
4766 036422
4767 036424 001405

      BR    30$          ;GET P CLOCK INFO
      CLOCK P,R1        ;IF NO CLOCK, ERROR
      BNCOMPLETE 25$     ;ELSE SET UP CLOCK INFO AND VECTOR
      JSR   PC,CLKSET    ;POINT CLKCSR TO P-CLK COUNT SET REG.
      ADD   #2,CLKCSR    ;LOAD CLK SET REG. WITH COUNT VALUE
      MOV   #PCLKCT,CLKCSR;POINT CLKCSR BACK TO P-CLK CSR
      SUB   #2,CLKCSR    ;SETUP TO ENABLE P-CLK DATA
      MOV   #PCLKEN,CLKEN
      BR    30$          ; THERE AIN'T NO CLOCK - DEATH!!
      ERRDF 21,EMSG51,ERR1
      GPHARD #0,R1        ;GET P-TAB POINTER FOR THIS UNIT
      BCOMPLETE 35$       ;THIS ONE IS NOT AVAILABLE
      JMP   INICLN
      MOV   (R1)+,UNACSR  ;SAVE CSR
      MOV   (R1)+,UNAVEC  ;SAVE VECTOR
      MOV   (R1)+,UNAPRI  ;SAVE PRIORITY
      SETVEC UNAVEC,#UNAISR,UNAPRI ;SETUP DELUA/DEUNA INTERRUPT VECTOR
      MOV   UNACSR,PCSR0  ;PCSR0
      PCSR0,PCSR1
      ADD   #2,PCSR1
      MOV   PCSR1,PCSR2
      ADD   #2,PCSR2
      MOV   PCSR2,PCSR3
      ADD   #2,PCSR3
      PCSR1,R3            ; get address of PCSR1 in R3
      MOV   (R3),R2          ; move value in PCSR1 into R2
      BIC   #177617,R2        ; isolate device id field of PCSR1
      R2,DEVICE             ; it is bits 4-6
      MOV   R2,DEVICE          ; move value into R2: 0=DEUNA non-0=DELUA
      CALL  MEMMAP           ; setup data structures in extended mem.
      CLR   S.NREC          ; CLEAR SUMMARY DATA COUNTERS
      CLR   S.REC
      CLR   S.LEN
      CLR   S.COMP
      CLR   S.BYTE
      CLR   S.XFER
      MOV   CLKHZ,TIMTCK    ;LOAD TICKS/SEC
      SETVEC CLKVEC,#CLKINT,CLKBR ;SETUP CLOCK INTERRUPT VECTOR
      MOV   CLKEN,CLKCSR    ;SET ENABLE BITS IN THE CLOCK TO START
      SETPRI #PRI00          ;SET PRIORITY=0 TO ALLOW FOR INTERRUPTS
      CALL  UNAINI           ;INITIALIZE THE DELUA/DEUNA
      ;---+
      ; Read the devices default physical address. If successful, print
      ; it out, else, tell user of error and proceed.
      ;---+
      CALL  FUNCT #RDDEFA   ;READ DELUA/DEUNA DEFAULT PHYSICAL ADDRESS
      P$POP R2                ;CHECK FOR ERROR
      BEQ   40$              ;

```

4768 036426
4769 036436 000423
4770 036440
4771 036462
4772
4773
4774
4775
4776
4777 036506
4778 036520
4779 036522 001405
4780 036524
4781 036534 000415
4782
4783 036536 113702 002152'
4784 036542 142702 000300
4785 036546
4786
4787
4788
4789
4790
4791
4792
4793
4794 036570
4795 036610 012703 002626'
4796 036614 012723 000002
4797 036620 012723 003110'
4798 036624 005023
4799 036626 005737 000524'
4800 036632 001404
4801 036634 012723 000002
4802 036640 012723 000030
4803
4804 036644
4805 036656
4806 036660 001405
4807 036662
4808 036672 000524
4809
4810 036674 013703 003110'
4811
4812
4813
4814
4815
4816 036700 005737 000524'
4817 036704 001403
4818 036706 006203
4819 036710 006203
4820 036712 006203
4821
4822 036714 032703 002000
4823 036720 001430
4824 036722 032703 004000

ERRSOFT 22,EMSG52 : INDICATE ERROR
BR 45\$: DON'T TRY TO PRINT
CALL BINHEX #PCBB2,#6,#STRBUF ;PUT ADDRESS INTO HEX FORMAT
PRINTS #HDMMSG1,#STRBUF ;PRINT ADDRESS

40\$: ;---+
; Read ROM firmware version number. If successful, print it out.
; else, tell user of error and proceed

45\$: ;---+
CALL FUNCT #RDSTA ;READ STATUS TO GET ROM VERSION
P\$POP R2 ;CHECK FOR ERROR
BEQ 47\$
ERRSOFT 23,EMSG53 : INDICATE ERROR
BR 50\$: DON'T TRY TO PRINT

47\$: ;---+
MOVB PCBB2,R2 ;ONLY WANT LOWEST 6 BITS
BICB #300,R2
PRINTS #HDMMSG2,R2 ;PRINT ROM VERSION

50\$: ;---+
; Now try to print BOOT select options. The options can be obtained
; by reading an internal location of the device. Unfortunately they
; are neither at the same address nor the same bits of the associated
; word. Some contortions must be gone through to print the info ...
; ... oh well ...

50\$: ;---+
PRINTS #HDMMSG3 ;PRINT MORE HEADER INFO
MOV #UCB20,R3 ;SET UP FUNCTION CONTROL BLOCK
MOV #2,(R3)+ ;MOVE 2 BYTES...
MOV #TEMP,(R3)+ ;INTO LOCATION TEMP...
CLR (R3)+ ;IDBB<17:16>
TST DEVICE ;What kind of device is this?
BEQ 55\$;If zero then DEUNA
MOV #2,(R3)+ ;else, DELUA IDBB<15:0>
MOV #30,(R3)+ ;IDBB<23:16>

55\$: ;---+
CALL FUNCT #DMPMEM ;DUMP INTERNAL MEMORY
P\$POP R2 ;CHECK FOR ERROR
BEQ 60\$;NO ERROR
ERRSOFT 24,EMSG18 ;REPORT ERROR AS SOFT ...
BR 90\$;... AND SKIP STATUS INFO

60\$: ;---+
MOV TEMP,R3 ;PUT RESULT INTO R3

62\$: ;---+
; For the DELUA, the status bits are 15:13 -- the DEUNA 12:10, so
; need to shift right if a DELUA

62\$: ;---+
TST DEVICE ;IS DEVICE DEUNA?
BEQ 62\$;YES, NO SHIFT
ASR R3 ;SHIFT STATUS ...
ASR R3 ;... THREE BITS ...
ASR R3 ;... TO THE RIGHT.

62\$: ;DETERMINE STATUS
BIT #BIT10,R3
BEQ 65\$
BIT #BIT11,R3

```

4825 036726 001441      BEQ    70$          : Is this DEUNA?
4826 036730 005737 000524' TST    DEVICE
4827 036734 001411      BEQ    63$          : YES -- special select for DEUNA
4828 036736          PRINTS #HDMMSG7       : else, remote boot not enabled
4829 036756 000446      BR     80$          :
4830
4831 036760          63$: PRINTS #HDMMSG4   : BIT10!BIT11 = REMOTE AND POWER UP BOOT ENABLED
4832 037000 000435      BR     80$          :
4833
4834 037002 032703 004000 65$: BIT    #BIT11,R3
4835 037006 001422      BEQ    75$          : BIT10 = REMOTE BOOT ENABLED
4836 037010          PRINTS #HDMMSG6       :
4837 037030 000421      BR     80$          :
4838
4839 037032          70$: PRINTS #HDMMSG5   : BIT11 = REMOTE BOOT ENABLED WITH ROM
4840 037052 000410      BR     80$          :
4841
4842 037054          75$: PRINTS #HDMMSG7   : REMOTE BOOT NOT ENABLED
4843
4844
4845          :---+ : Now look at self-test status and print it out
4846
4847 037074 032703 010000 80$: BIT    #BIT12,R3
4848 037100 001411      BEQ    85$          : BIT12 = SELF TEST ENABLED
4849 037102          PRINTS #HDMMSG8       :
4850 037122 000410      BR     90$          :
4851
4852 037124          85$: PRINTS #HDMMSG9   : SELF TEST DISABLED
4853
4854 037144 012737 000000 001170' 90$: MOV    #ALPHA,P$TYPE
4855 037152 012737 001000 001172'  MOV    #512.,P$SIZE
4856 037160 012737 000001 001174'  MOV    #1,P$CPYS
4857
4858 037166 023737 002034' 002044'  CMP    CLKHZ,TIMTCK
4859 037174 001004      BNE    95$          : THESE WON'T BE EQUAL IF CLOCK ...
4860 037176          ERRDF  25,EMSG51.ERR1  : ... CLOCK IS WORKING
4861
4862 037206          95$: CALL   DEVSTOP    : REPORT ERROR AND ABORT
4863
4864 037214 105037 001275' RESTRT: CLR B P$BLD
4865 037220 105037 001276'           CLR B P$HLP
4866 037224 105037 001303'           CLR B P$NCMP
4867 037230 105037 001306'           CLR B P$BONC
4868 037234 105037 001305'           CLR B P$TEXT
4869 037240 005037 002040'           CLR   TIMMIN
4870 037244 005037 002042'           CLR   TIMSEC
4871
4872 037250 013777 002036' 142550 NEW:  MOV    CLKEN,BCLKCSR
4873 037256          READEF #EF.START    : SET ENABLE BITS IN THE CLOCK TO START
4874 037264          BCOMPLETE INIEXI    : If here because of start, exit
4875 037266          SETPRI #PRI00      :
4876 037274 000401          BR     INIEXI    : Else, adjust priority level to enable interrupts
4877 037276          INICLN: DOCLN    : EXIT
4878 037300          INIEXI: EXIT    : ABORT PASS
4879
4880
4881          :---+ : INSERT LOCAL STORAGE THAT IS USED ONLY
4882

```

4883 ; DURING THE INITIALIZE SECTION.
4884 ;
4885 ;
4886 ;
4887 ; INSERT MESSAGES THAT ARE USED ONLY
4888 ; DURING THE INITIALIZE SECTION.
4889 ;
4890 ;
4891 .EVEN
4892
4893
4894 037304
ENDINIT

4896 .SBTTL AUTODROP SECTION
4897
4898
4899 :**
4900 : THIS CODE IS EXECUTED IMMEDIATELY AFTER THE INITIALIZE CODE IF
4901 : THE "ADR" FLAG WAS SET. THE UNIT(S) UNDER TEST ARE CHECKED TO
4902 : SEE IF THEY WILL RESPOND. THOSE THAT DON'T ARE IMMEDIATELY
4903 : DROPPED FROM TESTING.
4904 :--
4905 037306 BGNAUTO
4906
4908 :-----
4909 : INSERT CODE HERE TO CHECK DEVICE(S) TO SEE IF THEY RESPOND.
4910 :-----
4911 : ISSUE A "DODU" FOR THOSE THAT DON'T.
4913
4914 037306 ENDAUTO

```
4916 .SBTTL CLEANUP CODING SECTION
4917
4918
4919 :++: THE CLEANUP CODING SECTION CONTAINS THE CODING THAT IS PERFORMED
4920 : AFTER THE HARDWARE TESTS HAVE BEEN PERFORMED.
4921 :--
4922
4923 037310 BGNCLN
4924
4925 :-----+
4926 : INSERT YOUR CLEANUP CODING. THIS CODING SHOULD
4927 : RESTORE YOUR TEST-DEVICE TO A NEUTRAL STATE.
4928 : THIS CODE WILL BE EXECUTED AFTER EACH PASS AND AFTER THE
4929 : PROGRAM IS INTERRUPTED BY "+C".
4930 :-----
4931
4932
4933
4934 :---+
4935 : Name - Clean up code
4936
4937 : Functional Description:
4938 : The clean-up code is used to leave the DELUA/DEUNA in a
4939 : known state. This will result in the following steps:
4940 :
4941 : 1.) wait one second for all port commands to complete
4942 :
4943 : 2.) Stop the DELUA/DEUNA causing it to transition to the
4944 : ready state
4945 :
4946 : 3.) clear the DELUA/DEUNA's multicast address list, and
4947 :
4948 : 4.) if we have got here after the listen command then take
4949 : the device out of promiscuous mode
4950 :
4951 : Inputs - none
4952 :
4953 : Outputs - none
4954 :
4955 : Calling Procedure: gets called by the DRS
4956 :
4957 : Side Effects - listed above
4958 :
4959 : Subordinate Routines -
4960 : DEVSTOP - stop the DELUA/DEUNA
4961 : FUNCT - issue an ancillary port command
4962 :
4963 : Register Usage -
4964 : R2 - function return status
4965 :
4966 :---+
4967
4968 037310 SETPRI #PRI00 ; Let device and clock interrupt
4969
4970 037316 012737 000062 002046' 5$: MOV #62,TIMER1 ; Set up for one second loop
4971 037324 005737 002046' TST TIMER1 ; Have we timed out?
4972 037330 001375 BNE 5$ ; No, keep looping
4973 037332 005037 003012' CLR DNIFLG ; clear done interrupt flag
4974
```

4975 037336 CALL DEVSTOP ; stop the DELUA/DEUNA
4976 037344 012737 000000 002326' 10\$: MOV #0,\$WDMC+4 ;CLEAR MULTICAST ADDRESS LIST
4977 037352 CALL FUNCT #WDMULA ; WRITE 0 INTO LIST LENGTH
4978 037364 012737 000400 002326' MOV #400,\$WDMC+4 ; RESET FOR 1 ENTRY
4979 037372 P\$POP R2 ;CHECK FOR ERROR
4980 037374 001404 BEQ 15\$; IF OK CONTINUE
4981 037376 ERRDF 26,EMSG25 ; ELSE, REPORT ERROR
4982
4983 037406 105737 001274' 15\$: TSTB P\$LIST ; Did we get here after the listen command?
4984 037412 001426 BEQ 30\$; NO!!
4985 037414 105037 001274' CLRB P\$LIST ; clear listen flag
4986 037420 105037 001253' CLRB SOUFLG ; clear source address filter flag
4987 037424 105037 001254' CLRB DESFLG ; clear destination address filter flag
4988 037430 105037 001255' CLRB PROFLG ; clear protocol type filter flag
4989 037434 012737 000000 002570' MOV #0,\$WDMO+2 ; set up pcb to clear prom. mode
4990 037442 CALL FUNCT #WDMODE ; write mode into device
4991 037454 P\$POP R2 ; check for error
4992 037456 001404 BEQ 30\$; if OK, continue
4993 037460 ERRDF 27,EMSG23 ; else, report error
4994
4995 037470 005077 142332 30\$: CLR #CLKCSR ;DISABLE CLOCK
4996 037474 SETPRI #PRI07 ;SET PROCESSOR PRIORITY BACK TO 7
4997 037502 EXIT CLN
4998
5000 ;*****
5001 ; INSERT LOCAL STORAGE THAT IS USED ONLY
5002 ; DURING THE CLEANUP SECTION.
5003 ;*****
5004
5005 ;*****
5006 ; INSERT MESSAGES THAT ARE USED ONLY
5007 ; DURING THE CLEANUP SECTION.
5008 ;*****
5010
5011
5012
5013 037506 .EVEN
ENDCLN

5015 .SBTTL DROP UNIT SECTION
5016
5017 :
5018 : THE DROP-UNIT SECTION CONTAINS THE CODING THAT CAUSES A DEVICE
5019 : TO NO LONGER BE TESTED.
5020 :--
5021
5022 037510 BGNDU
5023
5025 :-----
5026 : INSERT DROP CODE HERE. THIS CODE WILL BE EXECUTED AFTER
5027 : A "DROP" COMMAND OR A "DODU" MACRO EXECUTION. THE PURPOSE
5028 : OF THIS CODE IS TO DO ANY NECESSARY HOUSEKEEPING AFTER A
5029 : UNIT HAS BEEN DROPPED. THIS SECTION IS OPTIONAL.
5030 :-----
5032
5033 037510 EXIT DU
5034
5036 :-----
5037 : INSERT LOCAL STORAGE THAT IS USED ONLY
5038 : DURING THE DROP-UNIT SECTION.
5039 :-----
5040 :-----
5041 :-----
5042 : INSERT MESSAGES THAT ARE USED ONLY
5043 : DURING THE DROP-UNIT SECTION.
5044 :-----
5046
5047
5048 .EVEN
5049 037514 ENDDU

ADD UNIT SECTION

```
5051          .SBTTL ADD UNIT SECTION
5052
5053
5054          ;+++
5055          ; THE ADD-UNIT SECTION CONTAINS ANY CODE THE PROGRAMMER WISHES
5056          ; TO BE EXECUTED IN CONJUNCTION WITH THE ADDING OF A UNIT BACK
5057          ; TO THE TEST CYCLE.
5058          ;--
5059 037516      BGNAU
5060
5061
5062          ;#####
5063          ;   INSERT ADD CODE HERE. THIS CODE WILL BE EXECUTED AFTER
5064          ;   AN "ADD" COMMAND. THE PURPOSE OF THIS CODE IS TO DO ANY
5065          ;   HOUSEKEEPING THAT MAY BE NECESSARY AFTER A UNIT HAS BEEN ADDED.
5066          ;   THIS SECTION IS OPTIONAL.
5067          ;#####
5068
5069
5070 037516      EXIT    AU
5071
5072
5073          ;#####
5074          ;   INSERT LOCAL STORAGE THAT IS USED ONLY
5075          ;   DURING THE ADD-UNIT SECTION.
5076          ;#####
5077
5078          ;#####
5079          ;   INSERT MESSAGES THAT ARE USED ONLY
5080          ;   DURING THE ADD-UNIT SECTION.
5081          ;#####
5082
5083
5084          .EVEN
5085
5086 037522      ENDAU
5087
5088
5089          .SBTTL TEST 1: NIE
5090          ;--+
5091          ; Name - NIE           Main loop for the NIE
5092
5093          ; Functional Description:
5094          ;   This is the one and only "test" in the program. When
5095          ;   entered, it will take control over user interactions by
5096          ;   presenting a completely separate interface than that of
5097          ;   the DRS. This interface is detailed in the NCSE functional
5098          ;   specification for the NIE.
5099
5100          ;
5101          ; The flow of control of the routine is as follows:
5102          ;
5103          ; REPEAT
5104          ;
5105          ;   CLEAR all variables associated with command parse
5106          ;
5107          ;   READ command line typed by user
5108          ;
5109          ;   PARSE the command line
5110          ;       (* the parse may result in the execution
5111          ;           of certain action routines *)
```

```
5112 : CASE parse_flags OF
5113 :   :
5114 :     P$GDBD : PRINT <error while parsing>
5115 :   :
5116 :     P$NNUF : PRINT <not enough input for parse>
5117 :   :
5118 :     P$HLP : EXECUTE HELP routine
5119 :   :
5120 :     P$BLD : EXECUTE BUILD routine
5121 :   :
5122 :     P$BONC : EXECUTE BOUNCE routine
5123 :   :
5124 :     P$LIST : EXECUTE LISTEN routine
5125 :   :
5126 : END_CASE
5127 : :
5128 : UNTIL (user inputs "EXIT" command)
5129 : :
5130 : NOTE: control will normally return to this routine after
5131 : appropriate actions have been taken to service the input
5132 : command. In some cases control will be grabbed by the DRS,
5133 : such as if a tC is typed, or a device fatal error is encountered
5134 : :
5135 : Inputs - none
5136 : :
5137 : Outputs - none
5138 : :
5139 : Calling Procedure: called by the DRS
5140 : :
5141 : Side Effects -
5142 :   1.) depending on what was input by the user, appropriate
5143 :       routines will be called to service the command.
5144 : :
5145 : Subordinate Routines -
5146 :   P$TRV - parsing routine
5147 :   EXEHELP - execute the help command
5148 :   EXEBLD - execute the build command
5149 :   EXEBNC - execute the bounce command
5150 :   EXELIS - execute the listen command
5151 : :
5152 : Register Usage - None
5153 : :
5154 : ;---+
5155 : :
5156 : 037524
5157 037524          BGNTST
5158
5159 037524 105037 001301'    GETCL: CLR8    P$GDBD      ;CLEAR CMD LINE PARING ERROR FLAG
5160 037530 105037 001300'    CLR8    P$NNUF      ;CLEAR NOT-ENOUGH FLAG
5161 037534 105037 001274'    CLR8    P$LIST      ;CLEAR LISTEN FLAG
5162 037540 105037 001275'    CLR8    P$BLD       ;CLEAR BUILD FLAG
5163 037544 105037 001306'    CLR8    P$BONC      ;CLEAR BOUNCE FLAG
5164 037550 105037 001276'    CLR8    P$HLP       ;CLEAR HELP FLAG
5165 037554          GMANID  CLI$PM,CMDBUF,A,0,1,72.,NO ;GET CMD LINE FROM OPERATOR
5166 037574 012737 000732' 001260'    MOV     #CMDBUF,P$BUFA ;SET UP ...
5167 037602 012737 003430' 001262'    MOV     #CLITRE,P$TREE ;... VARIABLES ...
5168 037610 012737 040012' 001264'    MOV     #CLIACT,P$ACT ;... FOR PARSE.
```

```

5169
5170 037616 005037 002024'           CLR   CFLAG          ;CLEAR QUALIFIER FLAG
5171 037622 004737 034342'           JSR   PC,P$TRV        ;GO PARSE COMMAND TREE
5172
5173 037626 105737 001301'           TSTB  P$GDBD         ;SEE IF PARSED OK, OR AN ERROR
5174 037632 001412                   BEQ   5$              ;IF NOT PRINT ERROR MESSAGE
5175 037634                   PRINTF #CLIERM      ;
5176 037654 000137 037772'           JMP   50$             ;
5177
5178 037660 105737 001300'           5$:   TSTB  P$NNUF        ;SEE IF INCOMPLETE COMMAND TYPED
5179 037664 001412                   BEQ   10$            ;IF NOT PRINT ERROR MESSAGE
5180 037666                   PRINTF #CLINUF      ;
5181 037706 000137 037772'           JMP   50$             ;
5182
5183 037712 105737 001276'           10$:  TSTB  P$HLP           ; help command?
5184 037716 001404                   BEQ   15$            ; branch if not
5185 037720 004737 040250'           JSR   PC,EXEHELP     ; execute it
5186 037724 000137 037772'           JMP   50$             ; get next command
5187
5188 037730 105737 001275'           15$:  TSTB  P$BLD           ; WAS BUILD COMMAND TYPED?
5189 037734 001403                   BEQ   20$            ; BRANCH IF NOT
5190 037736 004737 040644'           JSR   PC,EXEBLD      ; GO EXECUTE BUILD COMMAND
5191 037742 000413                   BR    50$             ; GO GET NEXT COMMAND
5192
5193 037744 105737 001306'           20$:  TSTB  P$BONC          ; bounce command?
5194 037750 001403                   BEQ   40$            ; branch if not
5195 037752 004737 042354'           JSR   PC,EXEBNC      ; execute bounce
5196 037756 000405                   BR    50$             ;
5197 037760                   40$:  TSTB  P$LIST          ; listen command?
5198 037760 105737 001274'           BEQ   50$            ; NAY!!
5199 037764 001402                   JSR   PC,EXELIS      ; execute listen command
5200 037766 004737 056272'           50$:  CMP   #CEXIT,CFLAG      ; WAS EXIT COMMAND TYPED?
5201 037772 022737 000020 002024'   BEQ   70$            ; YES, LEAVE!!
5202 040000 001402                   JMP   GETCL          ; IF NOT GET NEW COMMAND LINE
5203 040002 000137 037524'           70$:  EXIT  TST           ; ELSE EXIT
5204
5205
5206 040006                   .SBttl CLI ACTION TABLE AND ROUTINES
5207
5208
5209
5210
5211 040012                   : USER MUST CLEAR/SET P$GDBD IF USE "CLIBIF" IN CONNECTION WITH ACTION
5212 040012 006302                   : R2 WILL HOLD ACTION CODE FROM PARSING (CLI) NODE
5213 040014 016202 040030'           CLIACT: ASL   R2             ;MULTIPLY ACTION CODE BY 2
5214 040020 062702 040030'           MOV   10$(R2),R2       ;OFFSET VALUE
5215 040024 004712                   ADD   #10$,R2         ;ADD BASE VALUE
5216 040026 000207                   JSR   PC,(R2)        ;GO DO ACTION
5217
5218
5219 040030 000152                   10$:  .WORD ACTNUL-10$    ;BRIEF DESCRIPTION OF ACTION TAKEN
5220 040032 000210                   .WORD ACTHLP-10$    ;0-NULL
5221 040034 000262                   .WORD ACTNOD-10$    ;1-HELP
5222 040036 000600                   .WORD ACTBLD-10$    ;2-NODE
5223 040040 005116                   .WORD ACTRUN-10$    ;3-BUILD
5224 040042 007322                   .WORD ACTPAT-10$    ;4-RUN SPECIFIED TEST
5225 040044 011562                   .WORD ACTSAV-10$    ;5-SET 'MESSAGE PATTERN' TEST FLAG
5226
5227
5228
5229
5230
5231
5232
5233
5234
5235
5236
5237
5238
5239
5240
5241
5242
5243
5244
5245
5246
5247
5248
5249
5250
5251
5252
5253
5254
5255
5256
5257
5258
5259
5260
5261
5262
5263
5264
5265
5266
5267
5268
5269
5270
5271
5272
5273
5274
5275
5276
5277
5278
5279
5280
5281
5282
5283
5284
5285
5286
5287
5288
5289
5290
5291
5292
5293
5294
5295
5296
5297
5298
5299
5300
5301
5302
5303
5304
5305
5306
5307
5308
5309
5310
5311
5312
5313
5314
5315
5316
5317
5318
5319
5320
5321
5322
5323
5324
5325
5326
5327
5328
5329
5330
5331
5332
5333
5334
5335
5336
5337
5338
5339
5340
5341
5342
5343
5344
5345
5346
5347
5348
5349
5350
5351
5352
5353
5354
5355
5356
5357
5358
5359
5360
5361
5362
5363
5364
5365
5366
5367
5368
5369
5370
5371
5372
5373
5374
5375
5376
5377
5378
5379
5380
5381
5382
5383
5384
5385
5386
5387
5388
5389
5390
5391
5392
5393
5394
5395
5396
5397
5398
5399
5400
5401
5402
5403
5404
5405
5406
5407
5408
5409
5410
5411
5412
5413
5414
5415
5416
5417
5418
5419
5420
5421
5422
5423
5424
5425
5426
5427
5428
5429
5430
5431
5432
5433
5434
5435
5436
5437
5438
5439
5440
5441
5442
5443
5444
5445
5446
5447
5448
5449
5450
5451
5452
5453
5454
5455
5456
5457
5458
5459
5460
5461
5462
5463
5464
5465
5466
5467
5468
5469
5470
5471
5472
5473
5474
5475
5476
5477
5478
5479
5480
5481
5482
5483
5484
5485
5486
5487
5488
5489
5490
5491
5492
5493
5494
5495
5496
5497
5498
5499
5500
5501
5502
5503
5504
5505
5506
5507
5508
5509
5510
5511
5512
5513
5514
5515
5516
5517
5518
5519
5520
5521
5522
5523
5524
5525
5526
5527
5528
5529
5530
5531
5532
5533
5534
5535
5536
5537
5538
5539
5540
5541
5542
5543
5544
5545
5546
5547
5548
5549
5550
5551
5552
5553
5554
5555
5556
5557
5558
5559
5560
5561
5562
5563
5564
5565
5566
5567
5568
5569
5570
5571
5572
5573
5574
5575
5576
5577
5578
5579
5580
5581
5582
5583
5584
5585
5586
5587
5588
5589
5590
5591
5592
5593
5594
5595
5596
5597
5598
5599
5600
5601
5602
5603
5604
5605
5606
5607
5608
5609
5610
5611
5612
5613
5614
5615
5616
5617
5618
5619
5620
5621
5622
5623
5624
5625
5626
5627
5628
5629
5630
5631
5632
5633
5634
5635
5636
5637
5638
5639
5640
5641
5642
5643
5644
5645
5646
5647
5648
5649
5650
5651
5652
5653
5654
5655
5656
5657
5658
5659
5660
5661
5662
5663
5664
5665
5666
5667
5668
5669
5670
5671
5672
5673
5674
5675
5676
5677
5678
5679
5680
5681
5682
5683
5684
5685
5686
5687
5688
5689
5690
5691
5692
5693
5694
5695
5696
5697
5698
5699
5700
5701
5702
5703
5704
5705
5706
5707
5708
5709
5710
5711
5712
5713
5714
5715
5716
5717
5718
5719
5720
5721
5722
5723
5724
5725
5726
5727
5728
5729
5730
5731
5732
5733
5734
5735
5736
5737
5738
5739
5740
5741
5742
5743
5744
5745
5746
5747
5748
5749
5750
5751
5752
5753
5754
5755
5756
5757
5758
5759
5760
5761
5762
5763
5764
5765
5766
5767
5768
5769
5770
5771
5772
5773
5774
5775
5776
5777
5778
5779
5780
5781
5782
5783
5784
5785
5786
5787
5788
5789
5790
5791
5792
5793
5794
5795
5796
5797
5798
5799
5800
5801
5802
5803
5804
5805
5806
5807
5808
5809
5810
5811
5812
5813
5814
5815
5816
5817
5818
5819
5820
5821
5822
5823
5824
5825
5826
5827
5828
5829
5830
5831
5832
5833
5834
5835
5836
5837
5838
5839
5840
5841
5842
5843
5844
5845
5846
5847
5848
5849
5850
5851
5852
5853
5854
5855
5856
5857
5858
5859
5860
5861
5862
5863
5864
5865
5866
5867
5868
5869
5870
5871
5872
5873
5874
5875
5876
5877
5878
5879
5880
5881
5882
5883
5884
5885
5886
5887
5888
5889
5890
5891
5892
5893
5894
5895
5896
5897
5898
5899
5900
5901
5902
5903
5904
5905
5906
5907
5908
5909
5910
5911
5912
5913
5914
5915
5916
5917
5918
5919
5920
5921
5922
5923
5924
5925
5926
5927
5928
5929
5930
5931
5932
5933
5934
5935
5936
5937
5938
5939
5940
5941
5942
5943
5944
5945
5946
5947
5948
5949
5950
5951
5952
5953
5954
5955
5956
5957
5958
5959
5960
5961
5962
5963
5964
5965
5966
5967
5968
5969
5970
5971
5972
5973
5974
5975
5976
5977
5978
5979
5980
5981
5982
5983
5984
5985
5986
5987
5988
5989
5990
5991
5992
5993
5994
5995
5996
5997
5998
5999
5999

```

5226 040046 002644	.WORD ACTSUM-10\$:7-PRINT SUMMARY TABLE
5227 040050 003224	.WORD ACTIDT-10\$:10-REQUEST ID
5228 040052 004104	.WORD ACTEXT-10\$:11-EXIT
5229 040054 000144	.WORD ACTNUF-10\$:12-NOT ENOUGH INFO
5230 040056 004114	.WORD ACTXAD-10\$:13-EXTRACT NI NODE ADDRESS FROM INPUT LINE
5231 040060 004212	.WORD ACTSR4-10\$:14-SAVE POINTER TO BEGINING OF ADDRESS STRING
5232 040062 010756	.WORD ACTSND-10\$:15-SET 'NODE' FLAG FOR SHOW COMMAND
5233 040064 004220	.WORD ACTALP-10\$:16-SET 'ALPHA' FLAG
5234 040066 004230	.WORD ACTONE-10\$:17-SET 'ONES' FLAG
5235 040070 004240	.WORD ACTZRO-10\$:20-SET 'ZEROS' FLAG
5236 040072 004250	.WORD ACT1AL-10\$:21-SET '1ALT' FLAG
5237 040074 004260	.WORD ACTOAL-10\$:22-SET '0ALT' FLAG
5238 040076 004270	.WORD ACTCTT-10\$:23-SET 'CCITT' FLAG
5239 040100 004300	.WORD ACTOPR-10\$:24-SET 'OPER SEL' FLAG
5240 040102 004460	.WORD ACTTYP-10\$:25-DETERMINE MESSAGE TYPE
5241 040104 004466	.WORD ACTSZE-10\$:26-DETERMINE MESSAGE SIZE
5242 040106 004544	.WORD ACTCPY-10\$:27-DETERMINE MESSAGE COPIES
5243 040110 004622	.WORD ACTNAD-10\$:30-SET 'NODE/ADDRESS' FLAG
5244 040112 005004	.WORD ACTNAL-10\$:31-SET 'NODE/ALL' FLAG
5245 040114 005252	.WORD ACTRNA-10\$:32-SET 'ALL' FLAG FOR RUN COMMAND
5246 040116 006364	.WORD ACTRNL-10\$:33-SET 'LOOPPAIR' FLAG FOR RUN CMD
5247 040120 007404	.WORD ACTSMS-10\$:34-SHOW CURRENT MESSAGE PARAMETERS
5248 040122 007476	.WORD ACTCMS-10\$:35-RESET MESSAGE PARAMETERS TO DEFAULT
5249 040124 007602	.WORD ACTCNT-10\$:36-SET 'COUNTER' FLAG FOR SHOW COMMAND
5250 040126 011254	.WORD ACTCNL-10\$:37-CLEAR LOGICAL NODE NAMED FROM TABLE
5251 040130 011360	.WORD ACTFCT-10\$:40-INITIATE DELUA/DEUNA PORT COMMAND FUNCTION
5252 040132 000000	.WORD 0	: (was ACTUNS-10\$) 41-UNSAVE NODE TABLE
5253 040134 011430	.WORD ACTCSU-10\$:42-CLEAR SUMMARY TABLE
5254 040136 005720	.WORD ACTDIR-10\$:43-SET 'LOOP DIRECT' FLAG FOR RUN COMMAND
5255 040140 011514	.WORD ACTDFT-10\$:44-LOOK FOR PASS COUNT DEFAULT
5256 040142 012240	.WORD ACTUSF-10\$:45-UNSAVE NODE TABLE FROM A FILE
5257 040144 000154	.WORD ACTSQK-10\$:46-SET QUICK BLD FLAG
5258 040146 000164	.WORD ACTCQK-10\$:47-CLEAR QUICK BLD FLAG
5259 040150 000174	.WORD ACTCMP-10\$:50-NO DATA COMPARISON
5260 040152 000000	.WORD 0	: (* was ACTIBB-10\$ *) 51 - init bounce buffer pointer
5261 040154 002012	.WORD ACTSBB-10\$:52 - fill in address in bounce buffer
5262 040156 001664	.WORD ACTBLG-10\$:53 - calculate address from logical node number
5263 040160 013062	.WORD ACTSOU-10\$:54 - store input address in source filter
5264 040162 013120	.WORD ACTDES-10\$:55 - store input address in destination filter
5265 040164 013172	.WORD ACTPRO-10\$:56 - store protocol type in protocol filter
5266 040166 013156	.WORD ACTLIS-10\$:57 - set listen flag
5267 040170 017342	.WORD ACTSLI-10\$:58 - show listen log
5268 040172 020000	.WORD ACTCLI-10\$:59 - clear listen log

5270
5271
5272 ;ACTION ROUTINE TO INDICATE THAT NOT ENOUGH COMMAND
5273 ;INFORMATION HAS BEEN ENTERED
5274 ;
5275
5276 040174 112737 177777 001300' ACTNUF: MOVB #-1,P\$NNUF ;SET FLAG TO SAY NEED MORE OF COMMAND
5277
5278 ;ACTION ROUTINE TO DO NOTHING
5279 ;
5280
5281
5282 040202 000207 ACTNUL: RTS PC ;RETURN TO PARSER
5283
5284
5285 ;ACTION ROUTINE TO SET QUICK BUILD FLAG
5286 ;
5287
5288 040204 000240 ACTSQK: NOP
5289 040206 105037 CLRB P\$NNUF
5290 040212 000207 001300' RTS PC
5291
5292
5293 ; ACTION ROUTINE TO CLEAR QUICK BUILD FLAG
5294 ;
5295
5296
5297 040214 000240 ACTCQK: NOP
5298 040216 105037 CLRB P\$NNUF
5299 040222 000207 001300' RTS PC
5300
5301 ; ACTION ROUTINE TO SET NOCOMPARE FLAG
5302
5303 ;
5304 040224 105037 001300' ACTCMP: CLRB P\$NNUF
5305 040230 112737 177777 001303' MOVB #-1,P\$NCMP
5306 040236 000207 RTS PC
5307
5308 ; action routine to set help flag
5309
5310 040240 112737 177777 001276' ACTHLP: MOVB #-1,P\$HLP ; set help flag
5311 040246 000207 RTS PC ; return
5312
5313 ;--
5314 ; Name - EXEHELP
5315
5316 ; Functional Description:
5317 ; This routine will print out help to the user
5318
5319 ; Inputs - Implicit
5320 ; HLPTAB - table of addresses of help messages
5321
5322 ; Outputs - Prints out help messages at user's terminal
5323
5324 ; Calling Procedure: JSR PC,EXEHELP
5325
5326 ; Side Effects - none

```

5327
5328
5329
5330
5331
5332
5333 040250
5334 040250
5335 040252 012701 001310'      P$PUSH R1          : save R1
5336                                         MOV #HLPTAB,R1   : point R1 to table of addresses of help
5337 040256 10$: PRINTF (R1)+       : messages
5338 040274 020127 001412'      CMP R1,#HLPEND    : print a line of help message
5339 040300 001366               BNE 10$           : at end of table?
5340
5341 040302 105037 001276'      CLRB P$HLP        : NO, go print more
5342 040306
5343 040310 000207             P$POP R1          : clear the help flag
5344                               RTS PC           : restore R1
5345
5346
5347
5348
5349
5350 040312 105037 001300'      ACTNOD: CLRB P$NNUF      : CLEAR NOTNUF FLAG
5351 040316 004737 035366'      JSR PC,TRVADR    : TRAVERSE ADDRESS, CHECK IF TARGET OR ASSIST
5352 040322 105737 001301'      TSTB P$GDBD      : CHECK IF RESULTS OK
5353 040326 001137               BNE 50$          : IF NOT, RETURN WITH -1 IN P$GDBD
5354 040330 10$: CALL EDPACK CBOADR,#ADRBUF,#6 : GET ADDRESS INTO BUFFER
5355 040352 P$POP R1          : CHECK RESULTS FOR NUMBER OF CHAR.S
5356 040354 001411             BEQ 15$          : IF OK, BRANCH TO 15$
5357 040356 PRINTF #CADRER      : ELSE PRINT ERROR MESSAGE
5358 040376 000513             BR 50$          : AND RETURN
5359 040400 15$: CALL CMPTWO #ADRBUF,#ILLADR,#3 : SEE IF ILLEGAL ADDRESS
5360 040422 P$POP R1          : IF YES, PRINT ERROR MESSAGE
5361 040424 001021             BNE 17$          :
5362 040426 PRINTF #ILADMS     :
5363 040446 PRINTF #ILADM1     :
5364 040466 000457             BR 50$          :
5365 040470 17$: CALL BINHEX #ADRBUF,#6,#STRBUF : CONVERT BINARY ADDRESS
5366                               : INTO ASCII STRING
5367 040512 022737 000001 002024' CMP #CASIST,CFLAG : SEE IF TARGET OR ASSIST
5368 040520 001407             BEQ 20$          :
5369 040522 012737 017536' 001066' MOV #ARGTY7,KEYWD2 : MOVE 'TARGET' INTO KEYWD2
5370 040530 012737 000000 001200' MOV #CTARGT,NODTY : MOVE TARGET INTO NODE TYPE
5371 040536 000406             BR 25$          : MOVE 'ASSIST' INTO KEYWD2
5372 040540 012737 017527' 001066' 20$: MOV #ARGTY6,KEYWD2 : POINT SLOT TO START OF NODE TABLE
5373 040546 012737 000001 001200' MOV #CASIST,NODTY : CALL ROUTINE TO ENTER NODE IN TABLE
5374 040554 012737 100000 001202' 25$: MOV #NODTBL,SLOT : CHECK RESULTS
5375 040562 CALL ENTRND      : IF NODE TABLE FULL, RETURN
5376 040570 P$POP R1          : ELSE, MOVE "NODE" INTO KEYWD1
5377 040572 001015             BNE 50$          : INDICATE IF TARGET OR ASSIST
5378 040574 012737 017434' 001064' MOV #CMDTY7,KEYWD1
5379 040602 PRINTS #MSG2,#STRBUF
5380 040626 000207             RTS PC           :
5381
5382
5383

```

5384 :ACTION ROUTINE TO SET THE BUILD COMMAND FLAG
5385 :
5386 :
5387 040630 112737 177777 001275' ACTBLD: MOVB #1,P\$BLD :SET BUILD FLAG
5388 040636 105037 001300' CLRB P\$NNUF
5389 040642 000207 RTS PC :RETURN
5390
5391 :---+
5392 : Name - EXEBLD
5393 :
5394 : Functional Description
5395 : This routine executes the NIE build function. The build
5396 : function is used to create a node table of those nodes that
5397 : are present on the Ethernet that are conforming to the Ethernet
5398 : specification. Nodes that are not adhering to this spec will
5399 : not necessary be included in the built node table.
5400 : All correctly functioning Ethernet nodes periodically
5401 : transmit a system ID message at approximately ten minute
5402 : intervals. This routine attempts to capture all these IDs
5403 : and, thus, build a picture of the network by constructing
5404 : a node table. Note, the node table will not contain any
5405 : information on the physical position of the nodes with respect
5406 : to each other.
5407 : This routine can run for a maximum of 40 minutes. There
5408 : are three terminating conditions for the routine: 1.) the
5409 : operator may hit a control-C at which point control of the
5410 : diagnostic will be passed to the DRS, 2.) 40 minutes time
5411 : has elapsed since the operator invoked the build command, or 3.)
5412 : 10 minutes time has elapsed since the routine has received a
5413 : new system ID (one which it has not already received and
5414 : logged).
5415 :
5416 : Inputs - none
5417 :
5418 : Outputs - implicit
5419 : NOOTBL - Node Table
5420 : This structure will contain the current physical
5421 : addresses of all the nodes that the routine has
5422 : received a system ID from. It can contain a maximum
5423 : of 512 nodes.
5424 : DEFTBL - Default hardware address table
5425 : This structure will contain the default hardware
5426 : addresses of all the nodes that the routine has
5427 : received a system ID from. It also contains the
5428 : type of device attached to each node (e.g. DELUA,
5429 : DEQNA, etc.). This table can also contain a maximum
5430 : of 512 nodes.
5431 :
5432 : Calling Procedure: JSR PC,EXEBLD
5433 :
5434 : Side Effects - none
5435 :
5436 : Subordinate Routines -
5437 : RELBUF - used to release receive ring entries
5438 : FINDSL - routine to look for empty locations in node table
5439 : RECEIVE - routine to receive frames
5440 : GETRNX - update receive ring pointers

```

5441 : CMPEXT - compare received addresses with node table entries
5442 : MOVEEXT - move data from received frames to node/default table
5443 : GETIDA - get address of a particular field of system ID message
5444 : RETMEM - restore memory mapping to its original state
5445 :
5446 : Register Usage -
5447 : R1, R2, R3, R4 - multiple uses
5448 :
5449 :---+
5450 040644 EXEBLD:
5451 040644 1$:
5452 040644 PRINTS #MSG1 : print 'build' command message
5453 040664 PRINTS #MSG11
5454 040704 PRINTS #MSG12
5455 :
5456 040724 P$PUSH R1,R2,R3,R4 : save registers
5457 :
5458 040734 CALL FINDSL : is table already full?
5459 040742 P$POP R2 : see what find slot has to say
5460 040744 BEQ 3$ : branch if there is an empty slot
5461 040746 001402 041662' JMP 80$ : else, leave
5462 040752 000137 3$:
5463 040752 CALL DEVSTART : start up the DELUA/DEUNA
5464 040760 call funct #wdmule : write multicast address list
5465 040772 P$POP R2 : check for error
5466 040774 001404 beq 10$ : if OK, continue
5467 040776 errdf 28,emsg25,err1 : else report error
5468 041006 005037 003110' 10$: clr temp : clear 'no. nodes in last min.' counter
5469 041012 005037 003112' clr temp1 : clear node type argument (set to target)
5470 041016 005037 003114' clr temp2 : set interval counter
5471 041022 012737 000012 003116' mov #12,temp3 : set 'mins. since last new node' counter
5472 041030 012737 100000 001202' mov #nodtbl,slot : set slot to begining of node table
5473 041036 012737 000074 002052' 19$:
5474 041036 012737 000074 002052' mov #60.,timers
5475 041044 20$:
5476 041044 break : allow for control c interruption
5477 041046 005737 002052' tst timers : see if interval is up
5478 041052 001002 bne 201$ : It's not, keep going
5479 041054 000137 041506' jmp 40$ :
5480 :
5481 041060 201$:
5482 041066 CALL RECEIVE : else, check for reception of id message
5483 041070 001765 P$POP R2 : R2 holds no of messages received
5484 041072 012737 000013 003116' beq 20$ : if none, keep looking
5485 041100 013703 002100' mov #13,temp3 : got one : reset 'mins. since new node'
5486 041104 CALL GETRNX #RRGNXT : save receive ring pointer
5487 041116 016304 000010 MOV 10(R3),R4 : update pointer
5488 :
5489 :---+
5490 : There is a possibility that what was received was a broadcast frame.
5491 : So, check if it is and if so give it the old heave ho.
5492 :---+
5493 :
5494 041122 012702 002332' mov #ucb7,R2 : point R2 to rem. console mult. address
5495 041126 CALL CMPTWO R2,R4,#3 : compare received dest. with
5496 : console mult. address
5497

```

```

5498 041144          P$POP   R1           : Get result of compare
5499 041146 001117    bne     30$          : not equal, throw message away (effectively)
5500 041150 062704 000006    add     #sourcec,R4      : point R4 to node address
5501 041154 012702 100000    mov     #modtbl,R2      : point R2 to node table
5502 041160
5503 041160          21$:    CALL    CMPEXT #ONTAB,R2,#ORRING,R4,#3 ; see if node already on table
5504 041206
5505 041210 001476    22$:    P$POP   R1           : if same, don't add to table
5506 041212
5507 041212 062702 000010    add     #10,R2          : point to next table entry
5508 041216 020227 110000    CMP    R2,#NODEEND    : check to see if end of table
5509 041222 001356    bne     21$          : if no, compare next entry
5510
5511
5512
5513
5514
5515
5516 041224          23$:    After all entries in the node table have been checked and a match
5517 041232          has not been found, try to add the new node address to the table.
5518 041234 001071    24$:    CALL    FINDSL          : Look for an empty entry in the table
5519
5520
5521
5522
5523
5524 041236 013702 001202'    25$:    P$POP   R2           : get table full indicator
5525 041242          mov     slot,R2          : non-zero return means table full
5526
5527
5528
5529
5530
5531 041270 062702 010000    26$:    CALL    MOVEXT #ORRING,R4,#ONTAB,R2,#3 ; move addr. into node table
5532 041274 162704 000006    ADD    #DEFNOD,R2      : point R2 entry in default addr. table
5533 041300          sub     #sourcec,R4      : point R4 back to start of frame
5534 041314          call    getida R4,#7       : get address of default hardware address
5535 041316          p$pop   r1           : r1 points to default hardware address
5536
5537
5538
5539
5540 041344          27$:    CALL    MOVEXT #ORRING,R1,#ONTAB,R2,#3 ; save default address
5541 041360
5542 041362 111101          ADD    #DEFNOD,R2      : point R2 entry in default addr. table
5543 041364          sub     #sourcec,R4      : point R4 back to start of frame
5544 041376 110162 000007    call    getida R4,#144     : get node type address
5545
5546 041402 005237 003110'    p$pop   r1           : r1 points to node type
5547 041406          movb   (r1),r1          : put node type in r1
5548 041416 000612          CALL    REMAP #ONTAB      : allow access to node table
5549
5550 041420          28$:    MOVB   R1,7(R2)        : save node type in default table
5551 041420
5552 041430 012737 000005 002052'    inc    temp           : increment 'nodes in last min.' counter
5553
5554 041436          29$:    CALL    RELBUF R3         : release buffer to DELUA/DEUNA
5555
5556
5557
5558
5559
5560
5561
5562
5563
5564
5565
5566
5567
5568
5569
5570
5571
5572
5573
5574
5575
5576
5577
5578
5579
5580
5581
5582
5583
5584
5585
5586
5587
5588
5589
5590
5591
5592
5593
5594
5595
5596
5597
5598
5599
5600
5601
5602
5603
5604
5605
5606
5607
5608
5609
5610
5611
5612
5613
5614
5615
5616
5617
5618
5619
5620
5621
5622
5623
5624
5625
5626
5627
5628
5629
5630
5631
5632
5633
5634
5635
5636
5637
5638
5639
5640
5641
5642
5643
5644
5645
5646
5647
5648
5649
5650
5651
5652
5653
5654
5655
5656
5657
5658
5659
5660
5661
5662
5663
5664
5665
5666
5667
5668
5669
5670
5671
5672
5673
5674
5675
5676
5677
5678
5679
5680
5681
5682
5683
5684
5685
5686
5687
5688
5689
5690
5691
5692
5693
5694
5695
5696
5697
5698
5699
5700
5701
5702
5703
5704
5705
5706
5707
5708
5709
5710
5711
5712
5713
5714
5715
5716
5717
5718
5719
5720
5721
5722
5723
5724
5725
5726
5727
5728
5729
5730
5731
5732
5733
5734
5735
5736
5737
5738
5739
5740
5741
5742
5743
5744
5745
5746
5747
5748
5749
5750
5751
5752
5753
5754
5755
5756
5757
5758
5759
5760
5761
5762
5763
5764
5765
5766
5767
5768
5769
5770
5771
5772
5773
5774
5775
5776
5777
5778
5779
5780
5781
5782
5783
5784
5785
5786
5787
5788
5789
5790
5791
5792
5793
5794
5795
5796
5797
5798
5799
5800
5801
5802
5803
5804
5805
5806
5807
5808
5809
5810
5811
5812
5813
5814
5815
5816
5817
5818
5819
5820
5821
5822
5823
5824
5825
5826
5827
5828
5829
5830
5831
5832
5833
5834
5835
5836
5837
5838
5839
5840
5841
5842
5843
5844
5845
5846
5847
5848
5849
5850
5851
5852
5853
5854
5855
5856
5857
5858
5859
5860
5861
5862
5863
5864
5865
5866
5867
5868
5869
5870
5871
5872
5873
5874
5875
5876
5877
5878
5879
5880
5881
5882
5883
5884
5885
5886
5887
5888
5889
5890
5891
5892
5893
5894
5895
5896
5897
5898
5899
5900
5901
5902
5903
5904
5905
5906
5907
5908
5909
5910
5911
5912
5913
5914
5915
5916
5917
5918
5919
5920
5921
5922
5923
5924
5925
5926
5927
5928
5929
5930
5931
5932
5933
5934
5935
5936
5937
5938
5939
5940
5941
5942
5943
5944
5945
5946
5947
5948
5949
5950
5951
5952
5953
5954
5955
5956
5957
5958
5959
5960
5961
5962
5963
5964
5965
5966
5967
5968
5969
5970
5971
5972
5973
5974
5975
5976
5977
5978
5979
5980
5981
5982
5983
5984
5985
5986
5987
5988
5989
5990
5991
5992
5993
5994
5995
5996
5997
5998
5999
5999
6000
6001
6002
6003
6004
6005
6006
6007
6008
6009
60010
60011
60012
60013
60014
60015
60016
60017
60018
60019
60020
60021
60022
60023
60024
60025
60026
60027
60028
60029
60030
60031
60032
60033
60034
60035
60036
60037
60038
60039
60040
60041
60042
60043
60044
60045
60046
60047
60048
60049
60050
60051
60052
60053
60054
60055
60056
60057
60058
60059
60060
60061
60062
60063
60064
60065
60066
60067
60068
60069
60070
60071
60072
60073
60074
60075
60076
60077
60078
60079
60080
60081
60082
60083
60084
60085
60086
60087
60088
60089
60090
60091
60092
60093
60094
60095
60096
60097
60098
60099
600100
600101
600102
600103
600104
600105
600106
600107
600108
600109
600110
600111
600112
600113
600114
600115
600116
600117
600118
600119
600120
600121
600122
600123
600124
600125
600126
600127
600128
600129
600130
600131
600132
600133
600134
600135
600136
600137
600138
600139
600140
600141
600142
600143
600144
600145
600146
600147
600148
600149
600150
600151
600152
600153
600154
600155
600156
600157
600158
600159
600160
600161
600162
600163
600164
600165
600166
600167
600168
600169
600170
600171
600172
600173
600174
600175
600176
600177
600178
600179
600180
600181
600182
600183
600184
600185
600186
600187
600188
600189
600190
600191
600192
600193
600194
600195
600196
600197
600198
600199
600200
600201
600202
600203
600204
600205
600206
600207
600208
600209
600210
600211
600212
600213
600214
600215
600216
600217
600218
600219
600220
600221
600222
600223
600224
600225
600226
600227
600228
600229
600230
600231
600232
600233
600234
600235
600236
600237
600238
600239
600240
600241
600242
600243
600244
600245
600246
600247
600248
600249
600250
600251
600252
600253
600254
600255
600256
600257
600258
600259
600260
600261
600262
600263
600264
600265
600266
600267
600268
600269
600270
600271
600272
600273
600274
600275
600276
600277
600278
600279
600280
600281
600282
600283
600284
600285
600286
600287
600288
600289
600290
600291
600292
600293
600294
600295
600296
600297
600298
600299
600300
600301
600302
600303
600304
600305
600306
600307
600308
600309
600310
600311
600312
600313
600314
600315
600316
600317
600318
600319
600320
600321
600322
600323
600324
600325
600326
600327
600328
600329
600330
600331
600332
600333
600334
600335
600336
600337
600338
600339
600340
600341
600342
600343
600344
600345
600346
600347
600348
600349
600350
600351
600352
600353
600354
600355
600356
600357
600358
600359
600360
600361
600362
600363
600364
600365
600366
600367
600368
600369
600370
600371
600372
600373
600374
600375
600376
600377
600378
600379
600380
600381
600382
600383
600384
600385
600386
600387
600388
600389
600390
600391
600392
600393
600394
600395
600396
600397
600398
600399
600400
600401
600402
600403
600404
600405
600406
600407
600408
600409
600410
600411
600412
600413
600414
600415
600416
600417
600418
600419
600420
600421
600422
600423
600424
600425
600426
600427
600428
600429
600430
600431
600432
600433
600434
600435
600436
600437
600438
600439
600440
600441
600442
600443
600444
600445
600446
600447
600448
600449
600450
600451
600452
600453
600454
600455
600456
600457
600458
600459
600460
600461
600462
600463
600464
600465
600466
600467
600468
600469
600470
600471
600472
600473
600474
600475
600476
600477
600478
600479
600480
600481
600482
600483
600484
600485
600486
600487
600488
600489
600490
600491
600492
600493
600494
600495
600496
600497
600498
600499
600500
600501
600502
600503
600504
600505
600506
600507
600508
600509
600510
600511
600512
600513
600514
600515
600516
600517
600518
600519
600520
600521
600522
600523
600524
600525
600526
600527
600528
600529
600530
600531
600532
600533
600534
600535
600536
600537
600538
600539
600540
600541
600542
600543
600544
600545
600546
600547
600548
600549
600550
600551
600552
600553
600554
600555
600556
600557
600558
600559
600560
600561
600562
600563
600564
600565
600566
600567
600568
600569
600570
600571
600572
600573
600574
600575
600576
600577
600578
600579
600580
600581
600582
600583
600584
600585
600586
600587
600588
600589
600590
600591
600592
600593
600594
600595
600596
600597
600598
600599
600600
600601
600602
600603
```

CLI ACTION TABLE AND ROUTINES

5555 041436		CALL P\$POP	RECEIVE R2	; keep fetching frames until they stop	
5556 041444		BEQ	38\$; branch if none received	
5557 041446 001413	013703 002100'	MOV	RRGNXT,R3	; point R3 to received entry	
5558 041450		CALL	RELBUF R3	; release buffer to DELUA/DEUNA	
5559 041454		CALL	GETRNX #RRGNXT	; update ring pointer	
5560 041464					
5561 041476		38\$:	TST	TIMERS	; is time up?
5562 041476 005737	002052'	BNE	36\$; branch if time is not up	
5563 041502 001355		BR	50\$; yes, leave	
5564 041504 000431					
5565 041506		40\$:	dec	temp3	; see if 10 mins since last node
5566 041506 005337	003116'	beq	50\$; if yes, exit	
5567 041512 001426		inc	temp2	; see if time is up	
5568 041514 005237	003114'	cmp	temp2,#40.		
5569 041520 023727	003114' 000050	beq	50\$; if yes, exit	
5570 041526 001420		PRINTS	#bldmsg,temp,temp2	; else, print "still working" message	
5571 041530		clr	temp		
5572 041560 005037	003110'	JMP	19\$; do it again	
5573 041564 000137	041036'				
5574 041570		50\$:	PRINTS	#blddon,temp2	; print "build complete" message
5575 041570 012737	000000 002326'	mov	\$0,\$wdmc+4	; clear multicast address list	
5576 041614		call	funct #WDMULA	; write 0 into list length	
5577 041622		P\$POP	R2	; check for error	
5578 041634		beq	55\$; continue if ok	
5579 041636 001404		errdf	29.emsg25,err1	; else, report error	
5580 041640					
5581 041650		55\$:	jsr	pc,actand	; print node table
5582 041650 004737	051006'	mov	\$400,\$wdmc+4	; reset multicast list for 1 entry	
5583 041654 012737	000400 002326'				
5584 041662		80\$:	CLRB	P\$BLD	; clear build flag
5585 041662 105037	001275'	CALL	DEVSTOP	; stop the DELUA/DEUNA	
5586 041666		CALL	RETMEM	; return memory to original mapping	
5587 041674		P\$POP	R1,R2,R3,R4	; restore registers	
5588 041702		RTS	PC		
5589 041712 000207					
5590					
5591					
5592					
5593 041714		ACTBLG: P\$PUSH	R2	;SAVE R2	
5594 041716		CALL	REMAPP #ONTAB	; allow access to node table	
5595 041730 013702	001270'	MOV	P\$NUM,R2	;PUT NODE LOGICAL NUMBER INTO R2	
5596 041734 006302		ASL	R2	;MULTIPLY BY 8	
5597 041736 006302		ASL	R2	;NODE TABLE ADDRESS =	
5598 041740 006302		ASL	R2	; (LOG. NO. X 8) + #NODTBL	
5599 041742 062702	100000	ADD	#NODTBL,R2	;ADD OFFSET	
5600					
5601 041746 020227	110000	CMP	R2,#NODEND	; Does R2 point past the end of node table	
5602 041752 003002		BGT	5\$; Yes, an incorrect node has been specified	
5603 041754 005712		TST	(R2)	; is there an address here?	
5604 041756 001014		BNE	10\$; branch if there is	
5605					
5606 041760		5\$:	PRINTF	#EMSG46	; report it
5607 042000 112737	177777 001301'	MOVB	0-1,P\$GDBD	; set error	
5608 042006 000410		BR	20\$; leave	
5609 042010					
5610 042010 012237	001070'	MOV	(R2)+,ADRBUF	; put it in the address buffer	
5611 042014 012237	001072'	MOV	(R2)+,ADRBUF+2	; put it in the address buffer	

```

5612 042020 011237 001074'          MOV    (R2),ADRBUF+4      ; put it in the address buffer
5613 042024 105037 001302'          CLR8   P$AERR           ; clear address error flag
5614 042030                               20$:              P$POP  R2             ; restore regs
5615 042030                               CALL   RETMEM          ; restore memory mapping
5616 042032
5617 042040 000207                   RTS    PC              ; continue

5618
5619
5620
5621
5622
5623
5624
5625
5626
5627
5628
5629
5630
5631
5632
5633
5634
5635
5636
5637
5638
5639
5640
5641
5642
5643
5644
5645
5646
5647
5648
5649
5650 042042 023737 002076' 002072'  ACTSBB:::          CMP    XRGNXT,XRGCUR      ; has a buffer been allocated?
5651 042042 023737 002076' 002072'          BNE    10$             ; Yes, call ACTFBB
5652 042050 001003                               JSR    PC,ACTIBB        ; Else, call ACTIBB
5653
5654 042052 004737 042066'          BR     20$             ; ... and exit
5655 042056 000402
5656
5657 042060 004737 042224'          10$:              JSR    PC,ACTFBB        ; ....
5658 042064 000207                   20$:              RTS    PC              ; DONE!!
5659
5660
5661
5662
5663
5664
5665
5666
5667
5668

;---+
; Name - ACTSBB          Initialize the bounce buffer
; Functional Description:
; This action routine is called to initialize a transmit
; buffer to be used in the BOUNCE command. Also, it
; initializes some pointers that the BOUNCE routine must
; know about.

```

```

5669          : Inputs - Implicit
5670          :         ADRBUF - contains six bytes of destination address
5671          :
5672          : Outputs - none
5673          :
5674          : Calling Procedure: JSR PC.ACTIB8
5675          :
5676          : Side Effects -
5677          :         1.) Transmit buffer pointed to by XRGNXT is initialized for
5678          :             bounce command
5679          :         2.) Variables initialized:
5680          :             BNCCBUF - pointer to beginning of transmit buffer
5681          :             BNCCNT - number of loop information bytes -- set to 2 for
5682          :                 skip count
5683          :
5684          :
5685          : Subordinate Routines -
5686          :         REMAP - remap virtual memory
5687          :         RETMEM - restore memory mapping
5688          :
5689          : Register Usage -
5690          :         R1 - pointer to transmit buffer
5691          :
5692          :---+
5693 042066 ACTIB8:::
5694 042066      P$PUSH   R1          : Save R1
5695 042070      CALL      DEVSTART    : start up the DELUA/DEUNA
5696 042076      CALL      REMAP #OTRING : allow access to transmit ring
5697 042110 013701 002076'      MOV      XRGNXT,R1 : point R1 to next entry in ring
5698 042114 016137 000010 002062'    MOV      10(R1),BNCCBUF : save pointer to transmit buffer
5699 042122 016101 000010          MOV      10(R1),R1 : point R1 to transmit buffer
5700          :
5701 042126 013711 001070'      MOV      ADRBUF,(R1) : store six ...
5702 042132 013761 001072' 000002  MOV      ADRBUF+2,2(R1) : ... bytes of destination address ...
5703 042140 013761 001074' 000004  MOV      ADRBUF+4,4(R1) : ... in transmit buffer
5704          :
5705 042146 013761 003034' 000014  MOV      PROTO0,PROTOT(R1) : fill in protocol type
5706          :
5707 042154 005061 000016          CLR      16(R1) : skip count equals zero
5708 042160 012737 000002 002064'    MOV      #2,BNCCNT : two bytes of data are in data
5709          : field (skip count)
5710          :
5711 042166 112737 177777 001306'    MOVB    #-1,P$BONC : indicate that we are to do BOUNCE
5712 042174          P$POP    R1          : restore R1
5713 042176          CALL    GETXNX #XRGNXT : point XRGNXT to next ring entry
5714 042210          CALL    RETMEM     : restore memory mapping
5715 042216 105037 001300'          CLRB    P$NNUF : clear not enough flag
5716 042222 000207          RTS     PC       : all done!!
5717          :
5718          :---+
5719          : Name - ACTFB8           Fill bounce buffer
5720          :
5721          : Functional Description:
5722          :         This routine is used to fill in forwarding addresses into
5723          :             the loopback portion of a loopback message.
5724          :
5725          : Inputs - Implicit -

```

5726 : ADRBUF - contains the address to forward to
5727 :
5728 : Outputs - none
5729 :
5730 : Calling Procedure: JSR PC,ACTFB8
5731 :
5732 : Side Effects -
5733 : 1.) A forward function is added to the buffer pointed to by
5734 : BNCCNT
5735 : 2.) BNCCNT is update to reflect the addition of data to the
5736 : buffer
5737 :
5738 : Subordinate Routines -
5739 : REMAP - remap a portion of virtual memory
5740 : RETMEM - restore memory mapping
5741 :
5742 : Register Usage -
5743 : R2 - pointer to transmit buffer
5744 :
5745 :---+
5746 042224 ACTFB8:
5747 042224 P\$PUSH R2 ; save R2
5748 042226 CALL REMAP #0TRING ; allow access to transmit ring
5749 042240 013702 002062' MOV BNCCNT,R2 ; point R2 to transmit buffer
5750 042244 062702 000016 ADD #16,R2 ; point R2 past header info
5751 042250 063702 002064' ADD BNCCNT,R2 ; point R2 past info already in data field
5752 :---+
5753 :---+
5754 : Update count of information contained in this bounce buffer.
5755 : If the result is greater than the message size then abort attempt
5756 :---+
5757 042254 062737 000010 002064' ADD #10,BNCCNT ; update bounce count
5758 042262 023737 002064' 001172' CMP BNCCNT,P\$SIZE ; Is this greater than message size
5759 042270 003414 BLE 10\$; NO!
5760 042272 112737 177777 001301' MOVB #-1,P\$GDBD ; indicate bad command to parser
5761 042300 PRINTF #EMSG45 ; Tell user of problem
5762 042320 000410 BR 20\$; and take off
5763 :---+
5764 042322 012722 000002 10\$: MOV #2, (R2)+ ; set forward function code
5765 042326 013722 001070' MOV ADRBUF, (R2)+ ; set 6 bytes of forwarding address
5766 042332 013722 001072' MOV ADRBUF+2, (R2)+
5767 042336 013722 001074' MOV ADRBUF+4, (R2)+
5768 :---+
5769 042342 20\$: CALL RETMEM ; restore memory mapping
5770 042350 P\$POP R2 ; restore R2
5771 042352 000207 RTS PC ; return
5772 :
5773 :---+
5774 : Name - EXEBNC Execute bounce command
5775 :
5776 : Functional Description:
5777 : This routine is called to carry out the Bounce command
5778 : of the NI Exercisor. The bounce command is a function supplied
5779 : to the user so that he/she may choose any path of nodes
5780 : on the NI to loop a packet through.
5781 : To carry out this function a loop request message
5782 :

5783 : ;
5784 : ; is created with each of the nodes specified in the input
5785 : ; command line used as a forwarding field of the message.
5786 : ; To complete the loop the last forwarding field along with
5787 : ; the reply field is set to our own address. For example, if
5788 : ; the following command were input:
5789 : ;
5790 : ; NIE > bounce/AA0004000010,N3,N5,N7
5791 : ; then this loop request message would result:
5792 : ;
5793 : ;
5794 : ;
5795 : ;
5796 : ;
5797 : ;
5798 : ;
5799 : ;
5800 : ;
5801 : ;
5802 : ;
5803 : ;
5804 : ;
5805 : ;
5806 : ;
5807 : ;
5808 : ;
5809 : ;
5810 : ;
5811 : ;
5812 : ;
5813 : ;
5814 : ;
5815 : ;
5816 : ;
5817 : ; After the message is created and transmitted, an attempt is
5818 : ; made to receive the message. The message will be looped back
5819 : ; to our node if and only if all nodes on the specified path
5820 : ; forward the message properly. If the message is not received
5821 : ; the user will be notified as such and can then take further
5822 : ; steps to isolate the problem.
5823 : ;
5824 : ; NOTE: 1.) logical node names can be mixed with ethernet
5825 : ; addresses, and
5826 : ; 2.) the node order specified in the command line
5827 : ; dictates the path that the message will follow
5828 : ;
5829 : ; Inputs - Implicit
5830 : ; The buffer pointed to by BNCFBUF has an incomplete loop
5831 : ; request message in it. It contains all necessary information
5832 : ; except the last forwarding address and the reply address (both
5833 : ; our own)
5834 : ;
5835 : ; Outputs - none
5836 : ;
5837 : ; Calling Procedure: JSR PC,EXEBNC
5838 : ;
5839 : ; Side Effects -

5840 : 1.) loop request message is completed and transmitted
5841 : 2.) The status of the reception of the message is indicated
5842 : to the user
5843 :
5844 : Subordinate Routines -
5845 : REMAP - remap virtual memory
5846 : RETMEM - restore memory mapping
5847 : BLDBUF - fill the transmit buffer with data patterns
5848 : XMIT - transmit the loop request message
5849 : RUNCOM - Do receive
5850 :
5851 : Register Usage -
5852 : R2 - pointer to transmit buffer
5853 :
5854 :---
5855 042354 EXBNC: P\$PUSH R2 ; save r2 and r3
5856 042356 CALL REMAP #OTRING ; allow access to transmit ring
5857 :---
5858 :---
5859 : Position the pointer to the transmit buffer so that it points to
5860 : where more loop info should be added.
5861 :---
5862 042370 1157CE 002062' MOV BNCCNT,R2 ; let R2 point to transmit buffer
5863 042374 0C~02 600015 ADD #16,R2 ; point R2 past header info
5864 042400 063702 002064' ADD BNCCNT,R2 ; point R2 past loop data already in
5865 : buffer
5866 :---
5867 : Update the count of loop information in the bounce buffer. If it
5868 : is greater than the message size (P\$SIZE) then abort this command
5869 :---
5870 042404 062737 000020 002064' ADD #20,BNCCNT ; let bounce count reflect what will
5871 : be added
5872 042412 023737 002064' 001172' CMP BNCCNT,P\$SIZE ; TOO MUCH LOOP INFO ???
5873 042420 003414 BLE 10\$; NAY LADDIE!!
5874 042422 112737 177777 001301' MOVB #-1,P\$GDBD ; indicate error to parser
5875 042430 PRINTF #EMSG45 ; report error to user
5876 042450 000465 BR 50\$; and partake of the exit
5877 :---
5878 042452 10\$:
5879 :---
5880 : Add last forward address and the reply message to the bounce buffer.
5881 : They will both be the device's physical address.
5882 :---
5883 :---
5884 042452 012722 000002 MOV #2, (R2)+ ; put our address as forwarding address
5885 042456 013722 002244' MOV PHYADR, (R2)+
5886 042462 013722 002246' MOV PHYADR+2, (R2)+
5887 042466 013722 002250' MOV PHYADR+4, (R2)+
5888 042472 012722 000001 MOV #1, (R2)+ ; set reply message
5889 042476 013722 002244' MOV PHYADR, (R2)+ ; put our address in here
5890 042502 013722 002246' MOV PHYADR+2, (R2)+ ; 6 bytes worth
5891 042506 013722 002250' MOV PHYADR+4, (R2)+
5892 :---
5893 042512 CALL BLDBUF BNCCNT,BNCCNT ; fill the buffer with data patterns
5894 :---
5895 042530 CALL XMIT P\$POP R2 ; transmit the buffer
5896 042536 : error?

5897 042540 001404 BEQ 30\$; branch if okay
5898 042542 112737 177777 001301' MOVB #-1,P\$GDBD ; set error flag
5899 042550 000425 BR 50\$
5900
5901 042552 30\$: CALL RUNCOM ; execute common receive
5902 042552 P\$POP R2 ; get results
5903 042560 BEQ 40\$; branch if no error
5904 042562 001410 177777 001301' MOVBL #-1, P\$GDBD ; set error flag
5905 042564 112737 30,EMSG34
5906 042572 ERRSOFT
5907 042602 000410 BR 50\$; leave
5908 042604
5909 042604 PRINTF #OK ; say it arrived a okay
5910 042624 50\$:
5911
5912 :--+
5913 : A consequence of calling RUNCOM is the updating of certain summary
5914 : data counters. This routine does not add to the summary, but
5915 : must clear the counters, so that they are not misread by future
5916 : action routines.
5917 :--+
5918 042624 005037 002770' CLR S.NREC ; CLEAR SUMMARY DATA COUNTERS
5919 042630 005037 002766' CLR S.REC
5920 042634 005037 002772' CLR S.LEN
5921 042640 005037 002774' CLR S.COMP
5922 042644 005037 002776' CLR S.BYTE
5923 042650 005037 003000' CLR S.XFER
5924
5925 042654 CALL RETMEM ; restore memory mapping
5926 042662 CALL DEVSTOP ; stop the DELUA/DEUNA
5927 042670 P\$POP R2 ; restore R2
5928 042672 000207 RTS PC ; bye
5929
5930 :--+
5931 : Name - ACTSUM Print summary data
5932 :
5933 : Functional Description:
5934 : This action routine is called to print out the summary
5935 : data counters kept by the NIE.
5936 :
5937 : Inputs - Implicit - STATBL - table containing the summary data
5938 :
5939 : Outputs -
5940 : 1.) summary data is printed at the user terminal
5941 :
5942 : Calling Procedure: JSR PC,ACTSUM
5943 :
5944 : Side Effects - none
5945 :
5946 : Subordinate Routines -
5947 : BINHEX - convert binary data to HEX character string
5948 : BINDEC - convert binary data to decimal character string
5949 : REMAP - used to map summary table into page registers
5950 : RETMEM - restore memory mapping
5951 :
5952 : Register Usage -
5953 :

```

5954          : R1      - pointer to summary table
5955          : R2,R3,R4 - summary data
5956          :
5957          ;--+
5958
5959 042674 105037 001300' ACTSUM: CLR8  P$NNUF      ;CLEAR NOTNUF FLAG
5960 042700          CALL  REMAP #0$TAB    ; allow access to summary table
5961 042712          P$PUSH R1,R2,R3,R4
5962 042722 012701 100000  mov  #statbl,R1   ; move address of table to R1
5963 042726 005711          tst  (R1)       ; see if table empty
5964 042730 001013          bne  $      ; if not, cont.
5965 042732          printf #tabemt,#summ  ; else print 'table empty' message
5966 042756 000526          br   30$      ; exit
5967
5968 042760          5$:   printf #summs1    ; print the ...
5969 043000          printf #summs2    ; ... header info
5970
5971 043020 020127 126000 10$:   cmp   R1,#STAEND  ; See if at end of table
5972 043024 001503          beq   30$      ; if yes, exit
5973 043026 005711          tst   (R1)       ; see if rest of table empty
5974 043030 001501          beq   30$      ; if yes, exit
5975 043032          call   binhex R1,#6,#strbuf  ; print summary data
5976 043052 016102 000006  mov   6(R1),R2    ; RX not complete
5977 043056 016103 000010  mov   10(R1),R3   ; RX complete
5978 043062 016104 000012  mov   12(R1),R4   ; length errors
5979 043066          printf #summs3,#strbuf,R3,R2,R4; print them out
5980 043120 016102 000014  mov   14(R1),R2   ; compare errors
5981 043124 062701 000016  add   #16,R1    ; bytes compared
5982 043130          call   bindec R1     ; put into ascii string
5983 043140          printf #summs5,R2,#decstr  ; print them out
5984 043166 062701 000004  add   #4,R1     ; bytes transferred
5985 043172          call   bindec R1     ; put into ascii string
5986 043202          printf #summs6,#decstr  ; print
5987 043226 062701 000004  add   #4,R1     ; point R1 to next table entry
5988 043232 000672          br   10$      ; do it all again
5989 043234          30$:  CALL  RETMEM   ; restore memory mapping
5990 043242          P$POP R1,R2,R3,R4
5991 043252 000207          RTS   PC

5992
5993
5994
5995          ;ACTION ROUTINE TO INITIATE THE REQUEST ID TEST TO THE SPECIFIED NODE
5996          ;
5997
5998          ;--+
5999          : Functional Description
6000          : This subroutine builds and transmits Request ID frames
6001          : to the node specified by the operator in the command line.
6002          : The system ID info of the specified node is then displayed.
6003          : If the node does not respond before 60 seconds have passed
6004          : an error is reported to the operator.
6005
6006          : Inputs - Implicit - The specified node address is located in ADRBUF.
6007
6008          : Outputs - System ID info or error message printed to operator.
6009
6010          : Calling procedure - JSR PC, ACTIDT

```

```

6011
6012 : Side effects - XRGNXT pointer is updated by a call to BLDREQ sub.
6013
6014 : Register Usage - R1 - points to $WDMO for write mode operations.
6015 : R2 - is scratch.
6016 : R3 - points to the received message buffer.
6017 : R4 - scratch
6018
6019 :---+
6020
6021 043254 105737 001302' ACTIDT: TSTB P$AERR :SEE IF ADDRESS ENTERED WAS VALID
6022 043260 001402 BEQ 5$ : IF NOT, EXIT ACTION ROUTINE
6023 043262 000137 044026' JMP 70$ :--+
6024
6025 043266 105037 001300' 5$: P$PUSH R1,R2,R3,R4 : save registers
6026 043276 CLRB P$NNUF :CLEAR NOTNUF FLAG
6027 043302 CALL CMPTWO #ADRBUFF,ILLADR,#3 : see if illegal address
6028 043324 P$POP R1
6029 043326 001012 bne 10$ : if no, continue
6030 043330 PRINTF #ILADMS : else print illegal address message
6031 043350 000137 044026' jmp 70$ :--+
6032
6033 043354 10$: CALL CMPTWO #ADRBUFF,#PHYADR,#3 : see if address is own (host node)
6034 043376 P$POP R1 :
6035 043400 001563 beq 55$ :
6036 043402 012737 177776 003114' mov #-2,temp2 : set counter for no. of times tried
6037 043410 012701 002566' mov $WDMO,R1 : set up to write mode
6038 043414 012761 010000 000002 mov #10000,2(R1) : 10000: TPAD =1 (pad transmit buffers)
6039 043422 CALL FUNCT #WDMODE : write mode
6040 043434 P$POP R2 : check for error
6041 043436 001402 beq 15$ : br if error
6042 043440 000137 043772' jmp 60$ :--+
6043
6044 043444 15$: CALL DEVSTART : start up the DELUA/DEUNA
6045 043452 CALL BLDREQ : build Request ID message frame
6046 043460 CALL XMIT : transmit request
6047 043466 P$POP R2 : get results, R2 = success/failure
6048 043470 001402 beq 20$ : if OK branch
6049 043472 000137 044002' jmp 65$ : else exit routine
6050
6051 043476 005737 003024' 20$: tst retrys : see if failed due to excessive collisions
6052 043502 001412 beq 25$ : if no, cont.
6053 043504 printf #rtryer : yes, print 'excessive collisions' message
6054 043524 000137 043750' jmp 55$ : exit
6055
6056 043530 012704 002052' 25$: mov #timers,R4 : set up for 10 second timeout
6057 043534 012714 000012 mov #10.,(R4)
6058
6059 043540 30$: break : see if time has expired
6060 043542 005714 tst (R4) : if yes, branch
6061 043544 001431 beq 35$ : check for answer
6062 043546 CALL RECEIVE : R2 holds no. of buffers received
6063 043554 P$POP R2 : if no buffers received, loop
6064 043556 001770 beq 30$ :--+
6065
6066 043560 013703 002100' mov RRGNXT,R3 : get receive ring pointer
6067 043564 CALL GETRNX #RRGNXT : update pointer

```

```

6068 043576 016304 000010      mov    10(R3),R4          ; point R4 to message buffer
6069 043602 026427 000022 051115   cmp    $ircpt(R4),#MR   ; see if message received is in reply to one sent
6070 043610 001421                 beq    40$              ; if yes, branch to 25$
6071 043612                           CALL   RELBUF R3       ; release buffer to DELUA/DEUNA
6072 043622 005237 003114'          inc    temp2           ; increment retry counter
6073 043626 001344                 bne    30$              ; if no, look for correct reply message
6074
6075 043630                           35$:  errsoft 31.emsg22 ; else, report error
6076 043640 005237 002770'          inc    s.nrec            ; update summary data
6077 043644 012704 001070'          mov    #adrbuf,R4     ; point R4 to node that did not respond
6078 043650 000137 043720'          jmp    52$              ; and exit
6079
6080 043654 005237 002766'          40$:  inc    s.rec             ; increment 'received messages' counter
6081 043660 062737 000056 003000'   add    #46.,s.xfer      ; update 'bytes transferred' counter
6082
6083 043666                           call   prntid r4       ; Print the system id info
6084
6085 043676                           50$:  CALL   REMAP #0RRING ; allow access to receive ring
6086 043710 016304 000010          MOV    10(R3),R4       ; point R4 to received message again
6087 043714 062704 000006          ADD    #6,R4            ; point R4 to source address
6088 043720                           call   writes #1,R4,#0rring ; update summary table
6089 043740                           CALL   RELBUF R3       ; release buffer to DELUA/DEUNA
6090
6091 043750 005061 000002          55$:  clr    2(R1)            ; disable transmit padding
6092 043754                           CALL   FUNCT #WDMODE ; check for error
6093 043766                           P$POP R2             ; ain't none
6094 043770 001404                 BEQ    65$              ; error -- can't write mode
6095 043772                           60$:  errdf 32.emsg23,err1
6096
6097 044002                           65$:  CALL   RETMEM ; restore memory mapping
6098 044010                           CALL   DEVSTOP ; stop the DELUA/DEUNA
6099 044016                           P$POP R1,R2,R3,R4 ; restore registers
6100
6101 044026 000207                 70$:  RTS   PC             ; ACTION ROUTINE TO CHECK FOR ADDITION PARAMETER CHANGE INPUTS
6102
6103
6104
6105 ;AND PRINT OUT NEW PARAMETER INFO WHEN ALL INPUT ARE PROCESSED
6106
6107
6108
6109 044030 105714               ACTMSG: TSTB (R4)          ;CHECK FOR ADDITIONAL INPUT
6110 044032 001037                 BNE   50$              ; Branch if none
6111 044034 012737 017424' 001064' 12$:  MOV    #CMDTY6,KEYWD1 ;GET MESSAGE TYPE ASCII STRING ADDRESS
6112 044042 013701 001170'          MOV    P$TYPE,R1        ;INTO R1
6113 044046 006301                 ASL    R1
6114 044050 062701 001414'          ADD    #MSGTAB,R1      ;PRINT 'MESSAGE' COMMAND MESSAGE
6115 044054                           PRINTF #MSGPRM ;PRINT MSG PARAMETERS
6116 044074                           PRINTF #MSG4,(R1),P$SIZE,P$CPYS
6117 044126 105037 001300'          CLRB  P$NNUF          ;CLEAR NOTNUF FLAG
6118 044132 000207                 50$:  RTS   PC             ; ACTION ROUTINE TO RETURN CONTROL TO THE SUPERVISOR
6119
6120
6121
6122
6123
6124

```

6125 044134 012737 000020 002024' ACTEXT: MOV #CEXIT,CFLAG ;SET EXIT FLAG
6126 044142 000207 RTS PC

6127
6128
6129
6130 ;ACTION ROUTINE TO TAKE NI NODE ADDRESS FROM INPUT STRING BUFFER
6131 ;AND STORE IT IN THE BUFFER CALLED ADRBUF
6132 ;
6133
6134 044144 004737 053322' ACTXAD: JSR PC,XSTRIN ; put node address in CB0BUF
6135 044150 CALL EDPACK #CB0BUF,#ADRBUF,\$6 ;PUT NODE ADDRESS INTO ADRBUF
6136 044172 P\$POP R0
6137 044174 110037 001302' MOVB R0,P\$AERR ;SET ADDRESS=12 CHAR. GOOD/BAD FLAG
6138 044200 105737 001302' TSTB P\$AERR ;IF GOOD, RETURN
6139 044204 001415 BEQ 10\$
6140 044206 PRINTF #CADRER ;ELSE, PRINT ERROR MESSAGE
6141 044226 105037 001300' CLR8 P\$NNUF ;; AND CLEAR 'NOT ENOUGH' FLAG
6142 044232 112737 177777 001301' MOVB #-1,P\$GDBD ; set bogus command flag
6143 044240 000207 10\$: RTS PC
6144
6145
6146 ;ACTION ROUTINE TO STORE POINTER TO BEGINING OF OPERATOR INPUT ADDRESS
6147 ;IN COMMAND INPUT BUFFER
6148 ;
6149
6150 044242 010437 001166' ACTSR4: MOV R4,CBOADR ;SAVE STRING POINTER
6151 044246 000207 10\$: RTS PC
6152
6153
6154
6155 ;ACTION ROUTINE TO SET MESSAGE TYPE = ALPHA FLAG
6156 ;
6157
6158 044250 012737 000000 001170' ACTALP: MOV #ALPHA,P\$TYPE ;SET MESSAGE TYPE
6159 044256 000207 RTS PC
6160
6161
6162 ;ACTION ROUTINE TO SET MESSAGE TYPE = ALL ONES FLAG
6163 ;
6164
6165
6166 044260 012737 000001 001170' ACTONE: MOV #ONES,P\$TYPE ;SET MESSAGE TYPE
6167 044266 000207 RTS PC
6168
6169
6170 ;ACTION ROUTINE TO SET MESSAGE TYPE = ALL ZEROS FLAG
6171 ;
6172
6173
6174
6175 044270 012737 000002 001170' ACTZRO: MOV #ZEROS,P\$TYPE ;SET MESSAGE TYPE
6176 044276 000207 RTS PC
6177
6178
6179
6180 ;ACTION ROUTINE TO SET MESSAGE TYPE = ALTERNATING ONES FLAG
6181 ;

6182
6183 044300 012737 000003 001170' ACT1AL: MOV #ONEALT,P\$TYPE :SET MESSAGE TYPE
6184 044306 000207 RTS PC

6185
6186
6187 ;
6188 ;ACTION ROUTINE TO SET MESSAGE TYPE = ALTERNATING ZEROS FLAG
6189 ;
6190
6191 044310 012737 000004 001170' ACTOAL: MOV #ZROALT,P\$TYPE :SET MESSAGE TYPE
6192 044316 000207 RTS PC

6193
6194
6195 ;
6196 ;ACTION ROUTINE TO SET MESSAGE TYPE = CCITT FLAG
6197 ;
6198
6199 044320 012737 000005 001170' ACTCTT: MOV #CCITT,P\$TYPE :SET MESSAGE TYPE
6200 044326 000207 RTS PC

6201
6202
6203 ;
6204 ;ACTION ROUTINE TO SET MESSAGE TYPE = OPERATOR SELECTED INPUT
6205 ;
6206
6207 044330 105037 001304' ACTOPR: CLRB P\$MERR :CLEAR MESSAGE ERROR FLAG
6208 044334 112737 177777 001305' MOVB #-1,P\$TEXT : indicate text
6209 044342 004737 035366' JSR PC,TRVADR : process string
6210 044346 105037 001305' CLRB P\$TEXT : clear text flag
6211 044352 105737 001301' TSTB P\$GDBD : good string?
6212 044356 001403 BEQ 10\$: continue if it is
6213 044360 105037 001301' CLRB P\$GDBD : clear error flag
6214 044364 000425 BR 20\$: and report error

6215
6216 044366 022737 000006 002024' 10\$: CMP #OPRSEL,CFLAG : was it a user defines text?
6217 044374 001021 BNE 20\$: no, we have an error
6218 044376 012737 000006 001170' MOV #OPRSEL,P\$TYPE : yes, good user string, set type
6219 044404 CALL SELMSG R4 : and process it

6220
6221 ;---+
6222 ;---+ Make R4 point past string in input command line
6223 ;---+
6224 044414 P\$PUSH R2 : save R2 for now
6225 044416 012702 001722' MOV #OPSLBF,R2 : point R2 to selected message
6226 044422 122227 000000 15\$: CMPB (R2)+,#0 : reached the end of string yet?
6227 044426 001402 BEQ 18\$: YES,
6228 044430 005204 INC R4 : point past character of message
6229 044432 000773 BR 15\$: continue 'til all the way past

6230
6231 044434 18\$: P\$POP R2 : restore R2
6232 044436 000423 BR 50\$: and branch

6233
6234 044440 022737 000000 002024' 20\$: CMP #CTARGT,CFLAG : see if target flag set
6235 044446 001011 BNE 30\$: branch if it is
6236 044450 PRINTF #UNBOND : print unbounded error message
6237 044470 000406 BR 50\$: and branch

6238

```

6239 044472 105737 001304'      30$:   TSTB    P$MERR          ; see if unbounded string
6240 044476 001003                BNE     50$              ; branch if not
6241 044500 112737 177777 001301'  MOVP    #-1,P$GDBD        ; set error in good/bad flag
6242
6243 044506 000207                50$:   RTS     PC               ; return
6244
6245
6246 ;ACTION ROUTINE TO CHECK FOR MORE INPUT AFTER MESSAGE TYPE HAS BEEN
6247 ;ALTERED
6248 ;
6249
6250 044510 004737 044030'      ACTTYP: JSR    PC,ACTMSG        ;CHECK FOR ADDITIONAL COMMANDS
6251 044514 000207                RTS     PC
6252
6253
6254
6255 ;ACTION ROUTINE TO INPUT MESSAGE SIZE PARAMETER, CHECK TO SEE IF
6256 ;IT IS WITHIN LEGAL LIMITS, CHANGE PARAMETER AND THEN RETURN TO
6257 ;SEE IF MORE INPUT EXISTS
6258 ;
6259
6260 044516 023727 001270' 000037 ACTSZ:  CMP    P$NUM,#31.       ;CHECK FOR VALID SIZE RANGE
6261 044524 003410                BLE    10$              ;IF VALID CONTINUE
6262 044526 022737 002673 001270'  CMP    #1467.,P$NUM        ;SET MESSAGE SIZE
6263 044534 003404                BLE    10$              ;PRINT SIZE LIMIT EXCEEDED MESSAGE
6264 044536 013737 001270' 001172'  MOV    P$NUM,P$SIZE
6265 044544 000410                BR    20$              ;CHECK FOR ADDITIONAL COMMANDS
6266 044546                      10$:  PRINTF  #SIZLMT
6267 044566 004737 044030'      20$:  JSR    PC,ACTMSG
6268 044572 000207                RTS     PC
6269
6270
6271 ;ACTION ROUTINE TO INPUT COPIES PARAMETER, CHECK TO SEE IF IT IS
6272 ;WITHIN LEGAL LIMITS, CHANGE PARAMETER AND THEN RETURN TO SEE IF
6273 ;MORE INPUT PARAMETERS EXIST
6274 ;
6275
6276
6277 044574 023727 001270' 000000 ACTCPY: CMP    P$NUM,00          ;CHECK FOR VALID COPIES RANGE
6278 044602 003410                BLE    10$              ;IF VALID, CONTINUE
6279 044604 022737 000400 001270'  CMP    #256.,P$NUM        ;SET MESSAGE COPIES
6280 044612 003404                BLE    10$              ;PRINT COPY LIMIT EXCEEDED MESSAGE
6281 044614 013737 001270' 001174'  MOV    P$NUM,P$CPYS
6282 044622 000410                BR    20$              ;CHECK FOR ADDITIONAL COMMANDS
6283 044624                      10$:  PRINTF  #CPYLMT
6284 044644 004737 044030'      20$:  JSR    PC,ACTMSG
6285 044650 000207                RTS     PC
6286
6287
6288 ;ACTION ROUTINE TO CLEAR NODE SPECIFIED BY PHYSICAL ADDRESS FROM NODE TABLE
6289 ;
6290
6291
6292 044652 105037 001300'      ACTNAD: CLR8    P$NNUF          ;CLEAR NOTNUF FLAG
6293 044656 105737 001302'      TSTB    P$AERR          ;SEE IF ADDRESS ENTERED WAS VALID
6294 044662 001063                BNE    35$              ; IF NOT, EXIT ACTION ROUTINE
6295 044664                      P$PUSH R2,R3          ;SAVE R2 AND R3

```

```

6296 044670 012702 001070'      MOV    #ADRBUF,R2          ;MOVE ADDRESS OF ADDRESS INTO R2
6297 044674 012703 100000       MOV    #NODTBL,R3          ;MOVE ADDRESS OF NODE TABLE INTO R3
6298 044700                           CALL   REMAP #ONTAB        ; allow access to node table
6299
6300 044712                           21$:  CALL   CMPTWO R2,R3,#3      ;SEE IF ADDRESSES MATCH
6301 044730                           P$POP R1
6302 044732 001416                   BEQ   25$              ;IF YES, BR 25$
6303 044734 062703 000010       ADD    #10,R3           ;ELSE POINT R3 TO NEXT ENTRY
6304 044740 020327 110000       CMP    R3,#NODEND        ;ARE WE AT END OF NODE TABLE?
6305 044744 001362                   BNE   21$              ;IF NOT, COMPARE NEXT ENTRY
6306 044746                           PRINTF #NOCMPR        ;ELSE, PRINT ADDRESS DOESN'T COMPARE MSG.
6307 044766 000414                   BR    30$              ;RETURN
6308
6309 044770 005023                   25$:  CLR   (R3)+          ;ELSE, CLEAR NODE FROM TABLE
6310 044772 005023                   CLR   (R3)+          ;CLEAR (R3)+
6311 044774 005023                   CLR   (R3)+          ;CLEAR (R3)+
6312 044776 005013                   CLR   (R3)            ;CLEAR (R3)
6313 045000                           PRINTF #ADRDEL        ;PRINT NODE DELETED FROM TABLE MESSAGE
6314
6315 045020                           30$:  CALL   RETMEM          ; restore memory mapping
6316 045026                           P$POP R2,R3          ;RESTORE R2 AND R3
6317 045032 000207                   35$:  RTS   PC             ;RETURN
6318
6319
6320
6321 :ACTION ROUTINE TO CLEAR NODE TABLE
6322 :
6323
6324 045034                           ACTNAL: P$PUSH R2          ; save R2
6325 045036                           CALL   REMAP #ONTAB        ;ALLOW ACCESS TO THE NODE TABLE
6326 045050 012702 100000       MOV    #NODTBL,R2          ;MOVE NODE TABLE ADDRESS INTO R2
6327 045054 005022                   10$:  CLR   (R2)+          ;CLEAR WORD IN NODE/DEFAULT TABLE
6328 045056 020227 120000       CMP    R2,#DEFEND         ;ANY MORE?
6329 045062 001374                   BNE   10$              ;CONTINUE UNTIL DONE
6330 045064                           PRINTF #TABCLR,#NOD        ;PRINT NODE TABLE CLEARED MESSAGE
6331 045110 105037 001300'       CLRB  P$NNUF           ;CLEAR NOTNUF FLAG
6332 045114                           P$POP R2             ;RESTORE R2
6333 045116                           CALL   RETMEM          ;RESTORE MEMORY MAPPING
6334 045124 000207                   RTS   PC
6335
6336
6337 : Functional Description
6338 : This routine is used to calculate the logical node name
6339 : of a node.
6340
6341 : Inputs - P1 - pointer to a node in the node table
6342
6343 : Outputs - P2 - Integer representing the logical node name
6344
6345 : Calling Procedure - CALL LOGNAM P1
6346 : P$POP P2
6347
6348 : Side effects - None
6349
6350 : Subordinate routines - None
6351
6352 : Register Usage - R1 - scratch

```

```

6353
6354
6355 045126          ;--+
6356 045126          LOGNM::: P$POP R1      ; Get address of node
6357 045130 162701 100000   SUB  #NODTBL,R1    ; Make it an offset from base
6358 045134 006201      ASR   R1      ; DIVIDE
6359 045136 006201      ASR   R1      ; BY
6360 045140 006201      ASR   R1      ; EIGHT
6361 045142          RETURN R1      ; return the logical value
6362
6363
6364
6365          ;--+
6366          : Name - ACTRUN           Run a specified test
6367          : Functional Description:
6368          : This routine is called by the parse routine to run
6369          : the user specified test. It looks at the variable
6370          : KEYWD1 to determine which test it should call up, then
6371          : invokes the appropriate test. Also, it keeps track
6372          : of the pass count and calls the specified test the
6373          : appropriate number of times.
6374
6375          : Inputs - Implicit -
6376          : KEYWD1 - contains integer representing a test number
6377          : P$PASS - number of times to invoke test
6378
6379          : Outputs - none
6380
6381          : Calling Procedure: JSR PC,ACTRUN
6382
6383          : Side Effects -
6384          : 1.) invokes test specified by KEYWD1, P$PASS times
6385
6386          : Subordinate Routines -
6387          : DEVSTART - start up the DELUA/DEUNA
6388          : RUNALL - run the ALLNODE test
6389          : RUNLUP - run the looppair test
6390          : RUNDIR - run the direct loop test
6391          : RUNPAT - run the pattern test
6392          : DEVSTOP - stop the DELUA/DEUNA
6393
6394          : Register Usage - none
6395
6396          :--+
6397
6398 045146 105037 001300' ACTRUN: CLRB  P$NNUF      ; CLEAR 'NOT ENOUGH' FLAG
6399 045152 013737 001270' 001176' MOV   P$NUM,P$PASS
6400 045160          CALL  DEVSTART
6401 045166 022737 000032 001064' 5$: CMP  #CRNALL,KEYWD1 ; start up the DELUA/DEUNA
6402 045174 001004          BNE   10$      ; SEE IF 'ALL' TEST
6403 045176          CALL  RUNALL     ; IF NO, CONTINUE
6404 045204 000423          BR    30$      ; IF YES, DO ALLNODE
6405 045206 022737 000033 001064' 10$: CMP  #CLUPPR,KEYWD1 ; IS IT 'LOOPPAIR' TEST
6406 045214 001004          BNE   15$      ; IF NO, CONTINUE
6407 045216          CALL  RUNLUP     ; IF YES, DO LOOPPAIR
6408 045224 000413          BR    30$      ; IS IT 'DIRECT' TEST
6409 045226 022737 000043 001064' 15$: CMP  #CDIR,KEYWD1

```

```

6410 045234 001004          BNE    20$           : IF NO, CONTINUE
6411 045236                   CALL   RUNDIR        : IF YES, DO DIRECT
6412 045244 000403          BR    30$           :
6413 045246                   20$: CALL  RUNPAT       : ELSE, ITS 'PATTERN' TEST
6414 045254 023727 001176' 177777 30$: CMP  P$PASS, #1  : SEE IF PASS SET FOR INDEFINATE
6415 045262 001741          BEQ   5$            : IF YES, LOOP
6416 045264 005337 001176'          DEC  P$PASS       : HAVE WE DONE ALL PASSES?
6417 045270 001336          BNE   5$            : IF NO, LOOP
6418 045272                   CALL  DEVSTOP      : stop the DELUA/DEUNA
6419 045300 000207          RTS   PC             :
6420
6421
6422          : ACTION ROUTINE TO SET 'RUN ALL' FLAG
6423
6424
6425 045302 012737 000032 001064' ACTRNA: MOV    #CRNALL,KEYWD1 : SET FLAG
6426 045310 000207          RTS   PC             :
6427
6428          :--+
6429          : Name - RUNALL                      run ALLNODE test
6430
6431          : Functional Description:
6432          : This routine implements the NIE ALLNODE loop test.
6433          : This is a two part test. First, the direct loop
6434          : test is run. If all nodes respond to the direct loop
6435          : request, then a packet is looped between each pair of nodes
6436          : in the node table to establish the connectivity of
6437          : the two nodes at the farthest ends of the NI.
6438
6439          : Inputs - Implicit -
6440          :         1.) all nodes in the node table
6441
6442          : Outputs - Implicit -
6443          :         1.) adds or modifies entries in the summary table
6444
6445          : Calling Procedure: CALL RUNALL
6446
6447          : Side Effects - none
6448
6449          : Subordinate Routines -
6450          :         DIRCOM  - run the direct loop test
6451          :         FULSLT - find a valid entry in the node table
6452          :         BLDFAS - build a full assist message
6453          :         XMIT   - transmit the loopback packet
6454          :         REMAP   - allow access to the node table
6455          :         BINHEX - convert binary data to HEX character string
6456          :         LOGNM   - determine logical node name of a node
6457          :         RUNCOM - receive the loopback packet
6458          :         WRITES - write summary information to summary table
6459
6460          : Register Usage -
6461          :         R1    - pointer to target node
6462          :         R2    - pointer to assist node
6463          :         R3    - logical node number for target node
6464          :         R4    - logical node number for assist node
6465
6466          :--+

```

```

6467
6468 045312          RUNALL: CALL    DIRCOM      ; run loopdirect test
6469 045320          P$POP   R1           ; check results
6470 045322 001415      beq     5$           ; if OK, branch
6471 045324 022701 000001      cmp     #1,R1        ; else, was table empty?
6472 045330 001410      beq     3$           ; if yes, don't print abort message
6473 045332          prints  #pasabt      ; else abort test and print message
6474 045352 000137 045746' 3$:    jmp     32$          ; move node table address to slot
6475 045356 012737 100000 001202' 5$:    mov     #nodtbl.slot    ; find first entry
6476 045364          CALL    FULSLT      ; and put target address into R1
6477 045372 013701 001202'      mov     slot,R1        ; set up loop for no. of copies
6478 045376 013737 001174' 003122' 10$:   mov     P$CPYS.cpycnt
6479 045404 062737 000010 001202'      add     #10,slot      ; update slot
6480 045412          CALL    FULSLT      ; get next assist node from table
6481 045420 013702 001202'      mov     slot,R2        ; see if at end of table
6482 045424 022737 177777 001202'      cmp     #-1,slot      ; if yes, br
6483 045432 001530          15$:   beq     25$          ; build full assist message
6484 045434          CALL    BLDFAS  R1,slot      ; transmit message
6485 045450          CALL    XMIT         ; check results
6486 045456          P$POP   R3           ; transmit failed -- try next pair
6487 045460 001346          BNE    10$          ; allow access to node table
6488
6489 045462          17$:   CALL    REMAP   #ONTAB      ; set up buffers for error print ...
6490 045474          call    binhex R1,#6,#strbuf
6491 045514          call    binhex r2,#6,#strbuf
6492
6493 045534          CALL    LOGNM  R1           ; put the logical node name for ...
6494 045544          P$POP   R3           ; ... target into R3
6495 045546          CALL    LOGNM  R2           ; put the logical node name for ...
6496 045556          P$POP   R4           ; ... assist into R4
6497
6498 045560          printb #tstms4,#argty7,r3,#argty6,r4 ; assist node =
6499 045614          CALL    RUNCOM      ; do receive loop
6500 045622          P$POP   R4           ; check results
6501 045624 001405          beq    21$          ; if OK, loop some more
6502
6503 045626          20$:   errsoft 33,emsg42,ERR3      ; ... and print failing nodes
6504 045636 000410          br     101$          ; decrement 'copies' counter
6505
6506 045640          21$:   printb #okfu
6507 045660 005337 003122' 101$:   dec     cpycnt
6508 045664 001263          bne    15$          ; if more to do, loop
6509 045666          CALL    WRITES #2,R1,slot,#ontab; else, update summary table
6510 045712 000631          br     10$          ; point R1 to next target node
6511 045714 062701 000010          add     #10,R1
6512 045720 010137 001202'          mov     R1,slot      ; update slot
6513 045724          CALL    FULSLT      ; get address from table
6514 045732 013701 001202'          MOV    SLOT,R1
6515 045736 022737 177777 001202'      cmp     #-1,slot      ; see if end of table
6516 045744 001214          bne    10$          ; if no, continue else, finished
6517 045746          32$:   RETURN
6518
6519          ;ACTION ROUTINE TO SET 'RUN LOOP DIRECT' FLAG
6520          ;
6521          ;
6522          ;
6523 045750 012737 000043 001064' ACTDIR: MOV    #CDIR,KEYWD1      ; SET FLAG

```

6524 045756 000207 RTS PC

6525

6526 045760 RUNDIR: CALL DIRCOM : call common code

6527 045766 P\$POP R1

6528 045770 10\$: RETURN

6529

6530 :---

6531 : Name - DIRCOM direct loop test common code

6532

6533 : Functional Description:

6534 : This routine implements the NIE Direct Loop Test.

6535 : In this test a packet is looped directly to all nodes

6536 : in the node table

6537

6538 : Inputs - Implicit

6539 : 1.) nodes in the node table

6540

6541 : Outputs - Explicit -

6542 : P1 - return status of routine

6543

6544 : Implicit

6545 : 1.) add or modify entries in the summary table

6546

6547 : Calling Procedure: CALL DIRCOM

6548 P\$POP P1

6549

6550 : Side Effects - none

6551

6552 : Subordinate Routines -

6553 : FULSLT - find a valid entry in the node table

6554 : BLDLD - build loop direct packet

6555 : XMIT - transmit the loopback packet

6556 : REMAP - allow access to the node table

6557 : BINHEX - convert binary data to HEX character string

6558 : LOGNM - determine logical node name of a node

6559 : RUNCOM - receive the loopback packet

6560 : WRITES - write summary information to summary table

6561

6562 : Register Usage -

6563 : R1 - return status

6564 : R2 - return status of transmit

6565 : R3 - logical node number

6566 : R4 - return status of receive

6567

6568 :---

6569 045772 005001 DIRCOM: clr R1 : clear results register

6570 045774 012737 100000 001202' mov #nodtbl.slot : move node table address to slot

6571 046002 CALL FULSLT : see if table empty

6572 046010 022737 177777 001202' cmp #-1.slot

6573 046016 001015 bne 9\$: if no continue

6574 046020 printf #tabemt,#nod : else, print "table empty" message

6575 046044 012701 000001 mov #1,R1 : put 'table empty' indicator in R1

6576 046050 000554 br 32\$

6577 046052 012737 100000 001202' 9\$: mov #nodtbl.slot

6578 046060 013737 001174' 003122' 10\$: mov P\$CPYS,cpycnt : set up for no. of copies

6579 046066 CALL FULSLT : get next node in table

6580 046074 022737 177777 001202' cmp #-1.slot : see if at end of table

```

6581 046102 001537          beq    32$      : if yes, exit
6582
6583 046104          CALL    LOGNM   SLOT      : Get logical node name pointed to ...
6584 046116          P$POP   R3       : ... by slot and store in R1
6585 046120          CALL    REMAP   #ONTAB    : allow access to node table
6586 046132          CALL    BINHEX  SLOT,#6,#STRBUF : STRBUF holds address of node that will
6587                           : be looped directly to
6588
6589 046154          15$:    printb  #TSTMS2,#DIRECT,R3 : node address
6590 046202 022737 000005 001064'  CMP    #CPATRN,KEYWD1
6591 046210 001016          BNE    16$      :
6592 046212 013701 001170'    MOV    P$TYPE,R1
6593 046216 006301          ASL    R1       :
6594 046220 062701 001414'    ADD    #MSGTAB,R1
6595 046224          PRINTB  #MESP1,(R1)
6596
6597 046246          16$:    CALL    BLDLD   slot      : call build loopdirect subroutine
6598 046260          CALL    XMIT      : transmit loopdirect messages
6599 046266          P$POP   R2       : get results, R2 = success/failure
6600 046270 001273          bne    10$      : failed to transmit -- try next node
6601
6602 046272          26$:    CALL    RUNCOM   :
6603 046300          P$POP   R4       : do receive loop
6604 046302 001407          beq    29$      : get results
6605                           : if no errors, continue
6606 046304          ERRSOFT 34,EMSG48,ERR2
6607 046314 012701 177777          mov    #1,R1      : put error indicator into R1
6608 046320 000410          BR     101$      :
6609
6610 046322          29$:    PRINTB  #OK      : response ok
6611
6612 046342 005337 003122'  101$:  dec    cpycnt   :
6613 046346 001302          bne    15$      : if more to do, loop
6614 046350          CALL    WRITES #1,slot,#ontab : else, update summary table
6615
6616 046372 062737 000010 001202' 30$:  add    #10,slot   : increment to next node table entry
6617 046400 000627          br     10$      :
6618
6619 046402          32$:    CALL    RETMEM   : restore memory mapping
6620 046410          return  R1       :
6621
6622
6623
6624          :ACTION ROUTINE TO SET 'RUN LOOPPAIR' FLAG
6625          :
6626
6627 046414 012737 000033 001064' ACTRL:  MOV    #CLUPPR,KEYWD1 : SET FLAG
6628 046422 000207          RTS    PC       :
6629
6630          :--+
6631          : Function description
6632          : This routine implements the looppair function as described
6633          : by the NIE functional specification.
6634
6635          : Inputs - None
6636
6637          : Outputs - None

```

```

6638
6639 : Calling Procedure - CALL RUNLUP
6640
6641 : Side effects - The user sees information on the success or failure of each
6642 : attempted looping of a frame.
6643
6644 : Register Usage -
6645 : R1 - Pointer into the node table. This node will be used to
6646 : assist in the looping.
6647 : R2 - Pointer into the node table. This node will be used as
6648 : the target of the looping.
6649 : R3 - Integer representing the logical node name of the assist
6650 : node.
6651 : R4 - Integer representing the logical node name of the target
6652 : node.
6653 :
6654 :--+
6655 046424 012737 100000 001202' RUNLUP: MOV #NODTBL,SLOT      ; move node table address to slot
6656 046432          CALL FULSLT           ; see if table empty
6657 046440 022737 177777 001202'   CMP #-1,SLOT
6658 046446 001014          BNE 5$           ;
6659 046450          PRINTF #TABEMT,#NOD  ; if no, continue
6660 046474 000137 047054'   JMP 50$           ; else, print "Table empty" message
6661
6662 046500 012737 100000 001202' 5$: MOV #NODTBL,SLOT      ; move node table address to slot
6663 046506          CALL FULSLT           ; get first node in node table
6664 046514 013737 001202' 003112'  MOV SLOT,TEMP1        ; save first node to pair with last
6665
6666 046522 013737 001174' 003122' 10$: MOV P$CPYS.CPYCNT    ; set up for no. of copies
6667 046530 013701 001202'          MOV SLOT,R1           ; R1 points to assist node
6668 046534 062737 000010 001202'  ADD #10,SLOT           ; point SLOT to next entry in node table
6669 046542          CALL FULSLT           ; get next node in table
6670 046550 022737 177777 001202'  CMP #-1,slot           ; see if at end of table
6671 046556 001003          BNE 15$           ;
6672 046560 013702 003112'          MOV TEMP1,R2           ; Use first node in node table as target
6673 046564 000402          BR 20$            ; This will be the last loop tested
6674
6675 046566 013702 001202' 15$: MOV SLOT,R2           ; R2 Points to target node
6676
6677 046572          20$: CALL BLDFA$ R2,R1       ; build full assist message
6678 046604          CALL XMIT             ; transmit message
6679 046612          P$POP R4             ; check results
6680 046614 001077          BNE 35$           ; transmit failed -- try next pair
6681
6682 046616          25$: CALL LOGNM R1           ; get logical node name for assist ...
6683 046626          P$POP R3             ; ... and put it in R3
6684 046630          CALL LOGNM R2           ; get logical node name for target ...
6685 046640          P$POP R4             ; ... and put it in R4
6686 046642          PRINTB #TSTMS4,#ARGTY7,R4,#ARGTY6,R3 ; assist node =
6687
6688
6689 : Set up STRBUF, STRBU1 with addresses of the two nodes involved in this test
6690
6691 046676          CALL REMAP #ONTAB        ; allow access to node table
6692 046710          CALL BINHEX R2,#6,#STRBUF   ; STRBUF has target node
6693 046730          CALL BINHEX R1,#6,#STRBU1   ; STRBU1 has assist node
6694

```

6695 046750 CALL RUNCOM : do receive loop
6696 046756 P\$POP R3 : check results
6697 046760 001405 BEQ 30\$: if no errors, cont
6698
6699 046762 ERRSOFT 35.EMSG42.ERR3 : ... else, print failing nodes
6700 046772 000410 BR 35\$
6701
6702 046774 30\$: PRINTB #OKFU
6703
6704 047014 005337 003122' 35\$: DEC CPYCNT : decrement 'copies' counter
6705 047020 001264 BNE 20\$: if more to do, loop
6706 047022 CALL WRITES #2,R1,R2,#ONTAB : else, update summary table
6707
6708 047044 022737 177777 001202' CMP #-1,SLOT : Are we through?
6709 047052 001223 BNE 10\$: NAY!
6710
6711 047054 50\$: CALL RETMEM : restore memory mapping
6712 047062 RETURN
6713
6714 :---+
6715 : Name - RUNCOM Common receive code
6716
6717 : Functional Description:
6718 : This routine will perform the reception of loopback
6719 : messages transmitted by any of the loopback tests.
6720 : It will wait for ten seconds for the reply to the loopback
6721 : message. If it successfully receives the message, it
6722 : performs a data comparison on what was transmitted to what
6723 : was received.
6724 : The success of these operations will be returned
6725 : to the caller.
6726
6727 : Inputs - none
6728
6729 : Outputs - P1 - 0 = successful reception of loop message/ -1 = no success
6730
6731 : Calling Procedure: CALL RUNCOM
6732 : P\$POP P1
6733
6734 : Side Effects -
6735 : 1.) summary data counters are modified on error
6736
6737 : Subordinate Routines -
6738 : RECEIVE - receive a frame
6739 : GETRNX - update receive ring pointer
6740 : DATCMP - data compare routine
6741 : RELBUF - release a receive buffer to the DELUA/DEUNA
6742 : RETMEM - restore memory mapping
6743
6744 : Register Usage -
6745 : R1 - scratch
6746 : R2 - return status of this routine
6747 : R3 - pointer to receive ring
6748 : R4 - holds timer address
6749
6750
6751 :---+

```

6752 047064 005737 003024' RUNCOM: tst      retries          ; see if failed due to excessive collisions
6753 047070 001402 047330' beq      34$              ; if not, then try to receive
6754 047072 000137 047330' jmp      50$              ; else, take off
6755
6756 047076 012704 002052' 34$:   mov      #timers,R4        ; set up for 10 second timeout
6757 047102 012714 000012    mov      #10..(R4)
6758 047106 005002          clr      R2               ; clear results register
6759 047110
6760 047112 005714          35$:   break             ; see if time has expired
6761 047114 001475          tst      (R4)             ; if yes, branch
6762 047116
6763 047124
6764 047126 001770          beq      35$              ; check for answer
6765 047130 063737 003120' 003000' add      xfer.s.xfer    ; R2 holds no. of buffers received
6766 047136 005237 002766'          inc      s.rec             ; if no buffers received, loop
6767 047142 013703 002100'          mov      RRGNXT,R3       ; update bytes transferred sum. counter
6768 047146
6769 047160 016301 000006          mov      6(R3),R1        ; update frames received sum. counter
6770 047164 042701 170000          bic      #170000,R1     ; get receive ring pointer
6771 047170 162701 000004          sub      #4,R1            ; update pointer
6772 047174 020137 003126'          cmp      R1,buflen       ; get frame length from descriptor
6773 047200 001416
6774 047202 005237 002772'          beq      37$              ; zero out excess infor
6775 047206
6776 047234 000435          inc      s.len             ; subtract crc bytes
6777
6778 047236 016301 000010          37$:   mov      10(R3),R1      ; check for length error
6779 047242 062701 000016          add      #16,R1           ; if OK, br
6780 047246 005011
6781 047250 063737 001172' 002776'          clr      (R1)             ; else, update length errors counter
6782 047256
6783 047276
6784 047300 001413          add      P$SIZE,s.byte    ; print length error message
6785 047302 060137 002774'          CALL    DATCMP P$SIZE,CMPBUF,R1 ; and exit
6786 047306 000410          br      50$              ; point R1 to message buffer
6787
6788 047310 005237 002770'          40$:   mov      R1,s.comp       ; point R1 past header info
6789 047314 012737 017233' 001066'          inc      s.nrec            ; clear skip count for compare
6790 047322 012702 177777          mov      #noresp,keywd2  ; update bytes compared summary counter
6791 047326 000404          br      60$              ; check for data compare errors
6792
6793 047330
6794 047340
6795 047346
6796
6797
6798
6799
6800
6801
6802 047352 012737 000005 001064' ACTPAT: MOV      #CPATRN,KEYWD1 ; release buffer to DELUA/DEUNA
6803 047360 000207          RTS      PC                ; restore memory mapping
6804
6805
6806
6807
6808
;-----+
; Name - RUNPAT
;-----+
; ACTION ROUTINE TO SET 'RUN PATTERN' FLAG
;
;SET FLAG
run pattern test

```

```

6809
6810      ; Functional Description:
6811          This routine implements the NIE pattern test. It is
6812          identical to the loop direct test with the exception that
6813          it will loop a frame containing each of the defined data
6814          types.
6815
6816      ; Inputs - none
6817
6818      ; Outputs - none
6819
6820      ; Calling Procedure: CALL RUNPAT
6821
6822      ; Side Effects - none
6823
6824      ; Subordinate Routines -
6825          DIRCOM - direct loop test for each pattern
6826
6827      ; Register Usage -
6828          R1      - return status of DIRCOM
6829
6830
6831 047362      RUNPAT: P$PUSH P$TYPE      ; save type parameter
6832 047366 005037 001170'      clr    P$TYPE      ; set type to first type
6833 047372      5$:    CALL    dircom      ; send messages
6834 047400      P$POP   R1      ; get results to keep stack in order
6835 047402 001403      beq    10$      ; if OK, cont
6836 047404 022701 000001      cmp    #1,R1      ; else, was table empty
6837 047410 001406      beq    15$      ; if yes, return
6838 047412 005237 001170'      10$:   inc    P$TYPE      ; set to next type
6839 047416 022737 000005 001170'      cmp    #5,P$TYPE      ; see if done all of them
6840 047424 002362      bge    5$      ; if not, do more
6841 047426      15$:   P$POP   P$TYPE      ; restore message type
6842 047432      return
6843
6844
6845      ; ACTION ROUTINE TO SHOW THE CURRENT MESSAGE PARAMETERS
6846
6847
6848 047434 013701 001170'      ACTSMS: MOV    P$TYPE,R1      ; GET MESSAGE TYPE INTO R1
6849 047440 006301      ASL    R1      ; MULTIPLY BY 2
6850 047442 062701 001414'      ADD    #MSGTAB,R1      ; ADD MESSAGE TABLE OFFSET
6851 047446      PRINTF   #MSGPRM      ; PRINT MESSAGE PARAMETER MESSAGE
6852 047466      PRINTF   #MSG4,(R1),P$SIZE,P$CPYS      ; PRINT PARAMETERS
6853 047520 105037 001300'      CLR8    P$NNUF
6854 047524 000207      RTS    PC
6855
6856
6857
6858      ; ACTION ROUTINE TO CLEAR THE CURRENT MESSAGE PARAMETERS AND
6859      ; RESET THEM TO THE DEFAULT VALUE
6860
6861
6862 047526 012737 000000 001170'  ACTCMS: MOV    #ALPHA,P$TYPE      ; RESET TYPE
6863 047534 012737 001000 001172'  MOV    #512.,P$SIZE      ; RESET SIZE
6864 047542 012737 000001 001174'  MOV    #1,P$CPYS      ; RESET COPIES
6865 047550      PRINTF   #CLRMSG      ; PRINT MESSAGE PARAMETERS RESET MESSAGE

```

```

6866 047570
6867 047624 105037 001300'
6868 047630 000207      PRINTF #MSG4,MSGTAB,P$SIZE,P$CPYS ;PRINT PARAMETERS
                           CLRBL P$NNUF ;CLEAR NOTNUF FLAG
                           RTS PC

6869
6870
6871
6872      ;ACTION ROUTINE TO SET SHOW COUNTERS FLAG
6873
6874
6875 047632      ACTCNT: CALL DEVSTART ; start up the DELUA/DEUNA
6876 047640      CALL FUNCT #RDCNTS ;READ COUNTERS
6877 047652      P$POP R1 ;CHECK RESULT
6878 047654 001402 BEQ 21$ ;BRANCH IF ERROR
6879 047656 000137 050762' JMP 40$ ;PRINT COUNTER INFO

6880
6881
6882 047662      21$: CALL BINHEX #PHYADR,#6,#STRBUF ;GET ADDRESS INTO ASCII
6883 047704      PRINTF #CNTR00,#STRBUF
6884 047730      PRINTF #CNTR01,UCB12+2
6885 047754      CALL BINDEC #UCB12+4
6886 047766      PRINTF #CNTR02,#DECSTR
6887 050012      CALL BINDEC #UCB12+10
6888 050024      PRINTF #CNTR03,#DECSTR
6889 050050      PRINTF #CNTR04,UCB12+14
6890 050074      PRINTF #CNTR05,UCB12+16
6891 050120      CALL BINDEC #UCB12+20
6892 050132      PRINTF #CNTR06,#DECSTR
6893 050156      CALL BINDEC #UCB12+24
6894 050170      PRINTF #CNTR07,#DECSTR
6895 050214      PRINTF #CNTR08,UCB12+30
6896 050240      PRINTF #CNTR09,UCB12+32
6897 050264      CALL BINDEC #UCB12+34
6898 050276      PRINTF #CNTR10,#DECSTR
6899 050322      CALL BINDEC #UCB12+40
6900 050334      PRINTF #CNTR11,#DECSTR
6901 050360      CALL BINDEC #UCB12+44
6902 050372      PRINTF #CNTR12,#DECSTR
6903 050416      CALL BINDEC #UCB12+50
6904 050430      PRINTF #CNTR13,#DECSTR
6905 050454      CALL BINDEC #UCB12+54
6906 050466      PRINTF #CNTR14,#DECSTR
6907 050512      CALL BINDEC #UCB12+60
6908 050524      PRINTF #CNTR15,#DECSTR
6909 050550      CALL BINDEC #UCB12+64
6910 050562      PRINTF #CNTR16,#DECSTR
6911 050606      PRINTF #CNTR17,UCB12+70
6912 050632      PRINTF #CNTR18,UCB12+72
6913 050656      PRINTF #CNTR19,UCB12+74
6914 050702 005737 000524' TST DEVICE ; find out what devie we are talking to
6915 050706 001431 BEQ 50$ ; It's a DEUNA -- all done here
6916 050710      PRINTF #CNTR20,UCB12+100 ; ELSE DELUA -- print babble counter
6917 050734      PRINTF #CNTR21,UCB12+102 ; ... and port driver error counter
6918 050760 000404 BR 50$ ; stop the DELUA/DEUNA
6919
6920 050762      40$: ERRDF 36.EMSG31
6921
6922 050772      50$: CALL DEVSTOP ; stop the DELUA/DEUNA

```

6923 051000 105037 001300' CLRB P\$NNUF
 6924 051004 000207 RTS PC

6925
 6926
 6927
 6928 ;ACTION ROUTINE TO PRINT OUT THE NODE TABLE
 6929 ;
 6930
 6931 051006 105037 001300' ACTSND: CLRB P\$NNUF
 6932 051012 012737 100000 001202' MOV #NODTBL,SLOT
 6933 051020 CALL FULSLT
 6934 051026 022737 177777 001202' CMP #-1,SLOT
 6935 051034 001510 BEQ 15\$
 6936 051036 PRINTF #NTBHDR
 6937 051056 10\$: CALL FULSLT
 6938 051064 022737 177777 001202' CMP #-1,SLOT
 6939 051072 001503 BEQ 20\$
 6940 051074 CALL NTEXTI
 6941 051102 PRINTF #NODADR,#STRBUF
 6942 051126 PRINTF #DEFADR,#STRBU1
 6943 051152 PRINTF #LOGNAM,LOGVAL
 6944 051176 PRINTF #NETADR,AREA,DECNET
 6945 051226 PRINTF TYPADR
 6946 051246 062737 000010 001202' ADD #8.,SLOT
 6947 051254 000700 BR 10\$
 6948 051256 15\$: PRINTF #TABEMT,#NOD
 6949 051302 000207 20\$: RTS PC
 ;RETURN

6950
 6951
 6952
 6953
 6954 ;ACTION ROUTINE TO CLEAR A NODE SPECIFIED BY NODE LOGICAL NAME
 6955 ;FROM THE NODE TABLE
 6956 ;
 6957
 6958 051304 ACTCNL: P\$PUSH R2
 6959 051306 CALL REMAP #ONTAB
 6960 051320 013702 001270' MOV P\$NUM,R2
 6961 051324 006302 ASL R2
 6962 051326 006302 ASL R2
 6963 051330 006302 ASL R2
 6964 051332 062702 100000 ADD #NODTBL,R2
 6965 051336 005022 CLR (R2)+
 6966 051340 005022 CLR (R2)+
 6967 051342 005022 CLR (R2)+
 6968 051344 005012 CLR (R2)
 6969 051346 P\$POP R2
 6970 051350 105037 001300' CLRB P\$NNUF
 6971 051354 PRINTF #LOGDEL,P\$NUM
 6972 051400 CALL RETMEM
 6973 051406 000207 RTS PC
 ;RETURN

6974
 6975
 6976 ;ACTION ROUTINE TO INITIATE A DELUA/DEUNA PORT COMMAND
 6977 ;
 6978
 6979 051410 105037 001300' ACTFCT: CLRB P\$NNUF
 ;CLEAR NOTNUF FLAG

```

6980 051414          CALL    DEVSTART      ; start up the DELUA/DEUNA
6981 051422          CALL    FUNCT P$NUM   ;CALL FUNCTION ROUTINE WITH FUNCTION CODE
6982 051434          P$POP  R1           ;CHECK RESULTS
6983 051436 001404    BEQ    1$           ; IF OK EXIT
6984 051440          ERRDF  37,EMSG30   ; ELSE REPORT ERROR
6985 051450          1$:    CALL    DEVSTOP    ; STOP THE DELUA/DEUNA
6986 051456 000207    RTS    PC

6987
6988
6989          ;ACTION ROUTINE TO CLEAR SUMMARY TABLE
6990          ;
6991
6992 051460 105037 001300'      ACTCSU: CLR8  P$NNUF      ;CLEAR 'NOT ENOUGH' COUNTER
6993 051464          P$PUSH R2           ;SAVE R2
6994 051466          CALL    REMAP  #0$TAB    ;ALLOW ACCESS TO SUMMARY TABLE
6995 051500 012702 100000        MOV    #STATBL,R2   ;MOVE SUMMARY TABLE ADDRESS TO R2
6996 051504 005022          5$:    CLR    (R2)+     ;CLEAR FIRST WORD
6997 051506 020227 126000        CMP    R2,#STAEND  ;ANY MORE TO CLEAR?
6998 051512 001374          BNE    5$           ; IF YES, DO IT
6999 051514          PRINTF #TABCLR,#SUMM  ;ELSE, PRINT 'TABLE CLEARED' MESSAGE
7000 051540          P$POP  R2           ;AND RESTORE R2
7001 051542 000207          RTS    PC

7002
7003
7004          ;ACTION ROUTINE TO CHECK FOR PASS DEFAULT VALUE
7005          ;
7006
7007 051544          ACTDFT:          ;
7008 051544 121427 000040        1$:    CMPB  (R4),#40  ;SEE IF SPACES
7009 051550 001002          BNE    2$           ; IF NO, CONT.
7010 051552 005204          INC    R4           ; ELSE, POINT TO NEXT CHAR
7011 051554 000773          BR    1$           ; AND CHECK AGAIN
7012 051556 121427 000000        2$:    CMPB  (R4),#0  ;SEE IF DEFAULT VALUE
7013 051562 001007          BNE    10$          ; IF NO, BR
7014 051564 012763 000054 000002  MOV    #54,2(R3)  ; IF YES, POINT R3 TO SKIP CHECK PASS COUNT
7015 051572 012737 000001 001270'  MOV    #1,P$NUM   ;SET DEFAULT TO 1
7016 051600 000403          BR    15$          ;RETURN
7017 051602 012763 000004 000002  10$:   MOV    #4,2(R3)  ;POINT R3 TO CHECK FOR PASS COUNT
7018 051610 000207          15$:   RTS    PC

7019
7020
7021          ;--- Functional description
7022          This subroutine is used to save the current node table to
7023          the load device medium. For each entry that is filled in the
7024          node table, an entry will be made in a file including: the
7025          current address for a node, its default address, its logical
7026          name, and the type of device connected to the Ethernet at
7027          that node address. This information is formatted, then
7028          sequentially stored on a file resident on the load medium.
7029          When an empty slot in the node table is encountered, an
7030          appropriate message will be printed to the file.
7031
7032          ; Input - Implicit -
7033          The routine NTEXTI extracts information from the node
7034          table and leaves it in specific global variables. These
7035          are used by this routine. For their names and meanings,
7036          see the documentation on NTEXTI.

```

```

7037
7038 ; Outputs - file on load medium is created or appended to with the
7039 ; the information mentioned above
7040
7041 ; Calling procedure - JSR PC,ACTSAV
7042
7043 ; Side effects - none
7044
7045 ; Subordinate routines - FULSLT - Find a full slot
7046 ; OUTBLK - output a block of bytes
7047 ; FORLOG - format a logical name
7048 ; NTEXTI - extract info from node table
7049
7050 ; Register Usage -
7051 ; R2 - pointer to node table
7052
7053 051612 ACTSAV: P$PUSH R2,R3 ; Save some registers
7054 051616 OPEN CBOADR,W ; Open the specified file
7055
7056 051624 BNCOMPLETE 30$ ; Leave if the file can't be opened
7057
7058 051626 012737 100000 001202' 10$: MOV #NODTBL,SLOT ; point SLOT to beginning of node table
7059 051634 013702 001202' ; MOV SLOT,R2 ; point R2 to current node table entry
7060 051640 CALL FULSLT ; point SLOT to full entry in node table
7061 051646 022737 177777 001202' CMP #-1,SLOT ; Are we at the end of the node table
7062 051654 001522 BEQ 30$ ; Yes, done with this command
7063
7064
7065 ;---+ Check to see if the slot is full. If it isn't then print
7066 ; "EMPTY SLOT" to the save file
7067 ;---+
7068
7069 051656 020237 001202' 15$: CMP R2,SLOT ; Was slot pointed to by R2 full?
7070 051662 001412 BEQ 20$ ; Yes, go output info for this slot
7071 051664 CALL OUTBLK #EMPSLT,#14 ; No, output empty slot message
7072 051702 062702 000010 ADD #8,,R2 ; point R2 to next slot ...
7073 051706 000763 BR 15$ ; ... and keep trying
7074
7075
7076 ;---+ A full slot has been found. The following block writes the
7077 ; info to the save file
7078 ;---+
7079
7080 051710 20$: CALL NTEXTI ; set locations with node entry info
7081 051716 CALL OUTBLK #STRBUF,#21 ; output current node address for entry
7082 051734 CALL OUTBLK #SPACES,#4 ; output some spaces
7083 051752 CALL OUTBLK #STRBU1,#21 ; output default node address for entry
7084 051770 CALL OUTBLK #SPACES,#4 ; output some spaces
7085 052006 CALL FORLOG ; format the logical node name
7086 052014 P$POP R3 ; get number of characters in ...
7087 052016 CALL OUTBLK #STRBUF,R3 ; ... logical node name string and output
7088 052032 CALL OUTBLK #SPACES,#4 ; output some spaces
7089
7090
7091 ;---+ TYPADR points to a PRINTF formatted string. Just add 2 to the address
7092 ; to point past the formatting info
7093 ;---+

```

7094 052050 062737 000002 001164' ADD #2,TYPADR : point TYPADR to device description
7095 052056 CALL OUTBLK TYPADR, #5 : output device type for this entry
7096 052074 CALL OUTBLK #NEWLI2.#2 : <CR><LF> to file
7097
7098 052112 062737 000010 001202' ADD #8..SLOT : point SLOT to next node table entry
7099 052120 000645 BR 10\$: keep processing
7100
7101 052122 30\$: CLOSE : close up the file
7102 052124 P\$POP R2,R3 : restore register ...
7103 052130 105037 001300' CLRBL P\$NNUF : clear not enough flag
7104 052134 000207 RTS PC : ... and return
7105
7106
7107 :--+:
7108 : Functional Description:
7109 : This routine is designed to take a string of ascii text
7110 : and store it on the load medium. The file that is being
7111 : written is assumed to be already open.
7112 : Inputs - P1 - Address of a character string
7113 : P2 - Number of characters to be output to the load medium
7114 : Outputs - outputs P2 bytes from string P1 to load medium
7115 : Calling Procedure - CALL OUTBLK P1,P2
7116 : Side effects - None
7117 : Subordinate routines - None
7118 : Register Usage -
7119 : R1 - pointer to character string
7120 : R2 - count of bytes to output
7121 :--+:
7122 7127 052136 OUTBLK: P\$POP R1,R2 : get input parameters
7123
7124 7129 052142 10\$: PUTBYT (R1) : output a byte
7125
7126 7131 052150 005201 INC R1 : point R1 to next byte
7127 052152 005302 DEC R2 : decrement number of bytes to output
7128 7133 052154 001372 BNE 10\$: go on if there's more to do
7129
7130
7131
7132
7133
7134
7135 052156 RETURN : ALL DONE!!
7136
7137
7138 :--+:
7139 : Name - FORLOG
7140 :
7141 : Functional Description:
7142 : This routine is used to convert an integer representing a
7143 : logical node number (octal) into an ascii character string of
7144 : the form "N*", where "*" is a character string representing the
7145 : integer value. The node table can contain a maximum of
7146 : 2000(0) node entries, thus the length of the character string
7147 : will not exceed five ("N" + 4 digits).
7148 :
7149 7150 : Inputs - Implicit LOGVAL - word containing the logical node name to be formatted

```
7151 ; Outputs - Explicit
7152 ; P1 - the number of characters in the formatted string
7153 ;
7154 ;
7155 ; - Implicit
7156 ; STRBUF - will contain the formatted output string
7157 ;
7158 ; Calling Procedure - CALL FORLOG
7159 ; P$POP P1
7160 ;
7161 ; Side effects - STRBUF is modified
7162 ;
7163 ; Subordinate Routines - None
7164 ;
7165 ; Register Usage -
7166 ; R1 - Value to format
7167 ; R2 - scratch
7168 ; R3 - digit counter
7169 ; R4 - scratch
7170 ;--+
7171 052160 112737 000116 001116' FORLOG: MOVB #116,STRBUF ; put an 'N' in STRBUF
7172 052166 013701 001162' MOV LOGVAL,R1 ; get value to format
7173 ;--+
7174 ; Determine how many digits are needed to represent the logical
7175 ; node number. This can be ascertained by comparing the number
7176 ; to powers of eighth. For example, if the number is less than
7177 ; 8-squared (100(0)), it can be represented in two digits.
7178 ;--+
7179 052172 012703 000001 MOV #1,R3 ; there will be at least one digit
7180 052176 020127 000010 CMP R1,#10 ; represent # w/ 1 digit?
7181 052202 002411 BLT 10$ ; YES
7182 ;
7183 052204 005203 INC R3 ; NO, add one to digit count
7184 052206 020127 000100 CMP R1,#100 ; represent # w/ 2 digits?
7185 052212 002405 BLT 10$ ; YES
7186 ;
7187 052214 005203 INC R3 ; NO, add one to digit count
7188 052216 020127 001000 CMP R1,#1000 ; represent # w/ 3 digits?
7189 052222 002401 BLT 10$ ; YES
7190 ;
7191 052224 005203 INC R3 ; add one to digit count, MAX = 4 digits
7192 ;
7193 ;--+
7194 ; Convert the logical node number to its ascii equivalent string
7195 ;--+
7196 ;
7197 052226 010302 10$: MOV R3,R2 ; put digit count in R2
7198 ;
7199 052230 010104 20$: MOV R1,R4 ; put logical value in R4
7200 052232 042704 177770 BIC #177770,R4 ; isolate least significant 3 bits
7201 ;
7202 ;
7203 ;--+
7204 ; Adding 60(0) to a single digit creates its ascii representation
7205 ;
7206 052236 062704 000060 ADD #060,R4 ; create ascii value ...
7207 052242 110462 001116' MOVB R4,STRBUF(R2) ; ... move it into its string position
```

```

7208 052246 005302          DEC    R2           ; decrement digit count
7209 052250 001404          BEQ    30$         ; if no more digits, return
7210 052252 006201          ASR    R1           ; move next ...
7211 052254 006201          ASR    R1           ; ... 3 bits ...
7212 052256 006201          ASR    R1           ; ... into position
7213 052260 000763          BR     20$         ; and continue formatting
7214
7215 052262 005203          30$:   INC    R3           ; R3 = digit count + 1 for 'N'
7216 052264                  RETURN R3           ; back where we came from!!
7217
7218
7219 :--+
7220 : Name - ACTUSF          ACTION ROUTINE TO UNSAVE THE NODE TABLE
7221 :
7222 : Functional Description
7223 : This routine is used to restore the node table from a file
7224 : located on the load medium. It assumes that the file will
7225 : be in the following format:
7226 :
7227 : CURRENT ADDRESS DEFAULT ADDRESS LOGICAL NAME DEVICE
7228 :
7229 : The file is sequential read with each valid entry resulting
7230 : in the addition of a node to the node table. If a line is
7231 : of an invalid form or it reads "empty slot", a slot in the
7232 : node table will be left empty. This is to preserve the
7233 : original structure of the node table and also the correspon-
7234 : dence of logical node names to node addresses.
7235 :
7236 : Inputs - Implicit - Address of a string that names the file is in CBOADR
7237 :           - Explicit - Takes input from a file on the load medium
7238 :
7239 : Outputs - Implicit - The node table is restored from the file
7240 :
7241 : Calling Procedure - JSR PC,ACTUSF
7242 :
7243 : Side effects - The old node table will be wiped out in lieu of the new one
7244 :
7245 : Subordinate Routines
7246 :           RDLIN - read line of an open file
7247 :           NXTDEL - find next delimiter in a string
7248 :           NXTNDL - find next non-delimiter in a string
7249 :           EDPACK - edit data frame
7250 :           ENTRND - enter node into node table
7251 :
7252 : Register Usage
7253 :           R1 - Scratch
7254 :           R2 - Node type - target or assist
7255 :           R3 - Pointer to line of input from file
7256 :           R4 - pointer to node table
7257 :
7258 052270
7259 052270
7260 052300
7261 052312 012704 077770
7262 052316
7263 052324
7264 052326
ACTUSF:
P$PUSH R1,R2,R3,R4          ; save registers
CALL REMAP #ONTAB            ; allow access to node table
MOV #NODTBL-10,R4             ; let R4 point to node table
OPEN CBOADR                  ; open file, name=asciz string
BCOMPLETE 1$                   ; return if successful
PRINTF #OPNERR,CBOADR        ; else print "open error"

```

CLI ACTION TABLE AND ROUTINES

```

7265 052352 000137 053020'           : 1$:   JMP    30$      ; ... and leave
7266 052356 062704 000010             : ADD    #10,R4    ; point R4 to next node in table
7267 052362 012703 000526'           : MOV    #FILLIN,R3 ; point R3 to buffer for input line
7268 052366                                         CALL    RDLIN    ; read a line at a time
7269 052374                                         P$POP   R1       ; Get success of read in R1
7270 052376 001402                   BEQ    2$       ; non-zero means EOF
7271 052400 000137 053020'           JMP    30$      ; ... and leave
7272
7273 052404 020427 110000             : 2$:   CMP    R4,#NODEND ; check if the node table is full
7274 052410 001012                   BNE    3$       ; NOT this time
7275 052412                                         PRINTF  #NTBLOV ; print node table truncated ...
7276 052432 000137 053020'           JMP    30$      ; ... and take off
7277
7278 052436                                         CALL    NXTNDL   R3      ; Point R3 to current address
7279 052446                                         P$POP   R3       ; get updated pointer
7280 052450                                         CALL    EDPACK   R3,#ADRBUF,#6 ; Put address into binary
7281
7282
7283 ;---+ ; If results of call to EDPACK are unsuccessful, assume "Empty slot".
7284 ;---+
7285 052470                                         P$POP   R1       ; Get results of call
7286 052472 001403                   BEQ    20$      ; Success, go add entry
7287 052474 012714 000000             MOV    #0,(R4)  ; leave an empty slot in the node table
7288 052500 000726                   BR     1$       ; ... and move on
7289
7290
7291 ;---+ ; Store address in node table
7292 ;---+
7293
7294 052502 013714 001070'           20$:  MOV    ADRBUF,(R4) ; first two bytes
7295 052506 013764 001072' 000002   MOV    ADRBUF+2,2(R4) ; second two bytes
7296 052514 013764 001074' 000004   MOV    ADRBUF+4,4(R4) ; last two bytes
7297
7298 052522                                         CALL    NXTDEL   R3      ; point R3 past current address
7299 052532                                         P$POP   R3       ; get updated pointer
7300 052534                                         CALL    NXTNDL   R3      ; point R3 to default address
7301 052544                                         P$POP   R3       ; get updated pointer
7302 052546                                         CALL    EDPACK   R3,#ADRBUF,#6 ; get default address in ADRBUF
7303 052566                                         P$POP   R1       ; ERROR is a don't care - but clean stack
7304
7305 052570 010401                   MOV    R4,R1      ; point R1 to corresponding ...
7306 052572 062701 010000             ADD    #DEFNOD,R1 ; ... default node address
7307
7308 052576 013721 001070'           MOV    ADRBUF,(R1)+ ; ... and store the default address
7309 052602 013721 001072'           MOV    ADRBUF+2,(R1)+ ;
7310 052606 013721 001074'           MOV    ADRBUF+4,(R1)+ ;
7311
7312 052612                                         CALL    NXTDEL   R3      ; point R3 past current address
7313 052622                                         P$POP   R3       ; get updated pointer
7314 052624                                         CALL    NXTNDL   R3      ; point R3 to logical name
7315 052634                                         P$POP   R3       ; get updated pointer
7316 052636                                         CALL    NXTDEL   R3      ; and skip by it
7317 052646                                         P$POP   R3       ; get updated pointer
7318 052650                                         CALL    NXTNDL   R3      ; point R3 to device type (i.e. DEUNA)
7319 052660                                         P$POP   R3       ; get updated pointer
7320
7321 :

```

7322 : Now we want to extract the type of device attached to the node. Since
 7323 : there is just a description of the node in the file, we'll have to figure
 7324 : it out from there. It is possible to distinguish between types by looking
 7325 : at the third letter of the description (i.e. the 'U' in 'DEUNA').
 7326 :

7327 052662 062703 000002	ADD #2,R3	: point R3 to third letter of description	
7328			
7329 052666 121327 000125	CMPB (R3),#'U	: Is this a DEUNA?	
7330 052672 001005	BNE 22\$: NO	
7331 052674 112761 000001 000001	MOVB #IDTUNA,1(R1)	: put DEUNA identifier in table	
7332 052702 000137 052356'	JMP 1\$: through with line of input	
7333			
7334 052706 121327 000114	22\$: CMPB (R3),#'L	: Is this a DELUA?	
7335 052712 001005	BNE 23\$: NO	
7336 052714 112761 000011 000001	MOVB #IDTLUA,1(R1)	: put DELUA identifier in table	
7337 052722 000137 052356'	JMP 1\$: through with line of input	
7338			
7339 052726 121327 000121	23\$: CMPB (R3),#'Q	: Is this a DEQNA?	
7340 052732 001005	BNE 24\$: NO	
7341 052734 112761 000005 000001	MOVB #IDTQNA,1(R1)	: put DEQNA identifier in table	
7342 052742 000137 052356'	JMP 1\$: through with line of input	
7343			
7344 052746 122327 000103	24\$: CMPB (R3),#'C	: Is this a DECserver or DECNA	
7345 052752 001015	BNE 26\$: NO	
7346 052754 121327 000163	CMPB (R3),#'S	: IS This a DECserver?	
7347 052760 001005	BNE 25\$: NOPE!	
7348 052762 112761 000021 000001	MOVB #IDTSRV,1(R1)	: put DECserver identifier in table	
7349 052770 000137 052356'	JMP 1\$: through with line of input	
7350			
7351 052774 112761 000003 000001	25\$: MOVB #IDTCNA,1(R1)	: put DECNA identifier in table	
7352 053002 000137 052356'	JMP 1\$		
7353			
7354 053006 112761 177777 000001	26\$: MOVB #I-1,1(R1)	: move unknown identifier into table	
7355 053014 000137 052356'	JMP 1\$		
7356			
7357 053020	30\$: CLOSE	: close the open file	
7358 053022	P\$POP	: restore registers	
7359 053030	RETURN	R1,R2,R3	
7360			
7361 053032	NXTNDL: P\$POP	R1	: get pointer to string
7362 053034 121127 000040	5\$: CMPB (R1),#040	: Does R1 point to a space?	
7363 053040 001002	BNE 10\$: NO, go look for a tab	
7364 053042 005201	INC R1	: YES, point past the space	
7365 053044 000773	BR 5\$: keep checking	
7366 053046 121127 000011	10\$: CMPB (R1),#011	: Does R1 point to a tab?	
7367 053052 001002	BNE 15\$: NO, return	
7368 053054 005201	INC R1	: YES, point past the tab	
7369 053056 000766	BR 5\$: keep checking	
7370			
7371 053060	15\$: RETURN	R1	
7372			
7373 053064	NXTDEL: P\$POP	R1	: get pointer to string
7374 053066 121127 000040	5\$: CMPB (R1),#040	: does R1 point to a space	
7375 053072 001405	BEQ 15\$: YES, return	
7376 053074 121127 000011	CMPB (R1),#011	: does R1 point to a tab	
7377 053100 001402	BEQ 15\$: YES, return	
7378 053102 005201	INC R1	: point to next character	

7379 053104 000770		BR	5\$: keep checking
7380				
7381 053106 013737	001070' 001076'	15\$: ACTSOU:	RETURN R1 MOV ADRBUF,SOUFIL MOV ADRBUF+2,SOUFIL+2 MOV ADRBUF+4,SOUFIL+4	: return results : store 6 bytes of source filter
7382 053112 013737	001072' 001100'		MOV	:
7383 053120 013737	001074' 001102'		MOV	:
7384 053126 013737	177777 001253'		MOV	:
7385 053134 112737			MOVB #1,SOUFLG	: set source filter presence flag
7386 053142 105037	001300'		CLRB P\$NNUF	: clear not enough flag
7387 053146 000207			RTS PC	
7388				
7389 053150 013737	001070' 001104'	ACTDES:	MOV ADRBUF,DESFIL MOV ADRBUF+2,DESFIL+2 MOV ADRBUF+4,DESFIL+4	: store 6 bytes of destination filter
7390 053156 013737	001072' 001106'		MOV	:
7391 053164 013737	001074' 001110'		MOV	:
7392 053172 112737	177777 001254'		MOVB #1,DESFLG	: set destination filter presence flag
7393 053200 105037	001300'		CLRB P\$NNUF	: clear not enough flag
7394 053204 000207			RTS PC	
7395				
7396 053206		ACTLIS::		
7397 053206 112737	177777 001274'		MOVB #1,P\$LIST CLRB P\$NNUF RTS PC	: set listen command flag : clear "not enough" flag
7398 053214 105037	001300'			
7399 053220 000207				
7400				
7401 053222 004737	053322'	ACTPRO:	JSR PC,XSTRIN CALL EDPACK #CB0BUF,#PROFIL,#2 ;STORE PROTOCOL FILTER	: Put protocol type in CB0BUF
7402 053226			P\$POP R0	: get return status
7403 053250			TSTB R0	: was this a successful call?
7404 053252 105700			BEQ 5\$: yes, take off!
7405 053254 001416			PRINTF #CPEROER	: else print error
7406 053256			CLRB P\$NNUF	: clear "not enough" flag
7407 053276 105037	001300'		MOVB #1,P\$GDBD	: set bogus command flag
7408 053302 112737	177777 001301'		BR 10\$: exit!
7409 053310 000403			MOVB #1,PROFLG	: set protocol filter presence flag
7410 053312 112737	177777 001255'	5\$:	RTS PC	
7411 053320 000207		10\$:		
7412				
7413 053322		XSTRIN:	P\$PUSH R1,R2,R3	: save these registers
7414 053330 013701	001166'		MOV CBOADR,R1	: get address of string to extract
7415 053334 012702	001042'		MOV #CB0BUF,R2	: get address of buffer to hold it
7416 053340 121127	000057	10\$:	CMPB (R1),#57	: Is this char. a "/"?
7417 053344 001407			BEQ 20\$: Yes!!
7418 053346 121127	000054		CMPB (R1),#54	: Or a comma?
7419 053352 001404			BEQ 20\$: Yes!!
7420 053354 105711			TSTB (R1)	: Or is it the end of command line?
7421 053356 001402			BEQ 20\$: Yes!!
7422 053360 112122			MOVB (R1)+,(R2)+	: buffer the character
7423 053362 000766			BR 10\$: go look at next character in command line
7424 053364 105012		20\$:	CLRB (R2)	: put a null character at end of extracted string
7425 053366 010104			MOV R1,R4	: point command line pointer past what
7426				: we just grabbed
7427 053370			P\$POP R1,R2,R3	: restore registers
7428 053376 000207			RTS PC	: LATER!
7429				
7430		.SBTTL READ LINE OF OPENED FILE		
7431				
7432				
7433				: THIS ROUTINE GETS BYTES FROM AN OPENED FILE UNTIL A CR IS ENCOUNTERED
7434				: "EOF" AND "BAD" FLAGS ARE SET IF END-OF-FILE OR ERRORS ARE ENCOUNTERED
7435				

```

7436      : NOTE: ASSUMING A ASCII TEXT FILE IS BEING READ, FOR EXAMPLE:
7437          : AA-00-03-00-01-AB<CR><LF>
7438          :
7439          : AA-00-03-00-01-AB<CR><LF>
7440
7441          : WHAT YOU SEE READ BYTE-BY-BYTE IS:
7442          : "A..-AB<CR><LF>A..-AB<CR><LF>..<0><0><0>....???
7443          : SO I MADE ASSUMPTION THAT SINCE SEE "0-PADDING" AFTER LAST CHAR TO
7444          : END-OF-FILEBLOCK, ANY CHARACTER THAT IS NOT "SPACE OR GREATER" OR A
7445          : <CR> OR <LF> THEN I'LL TAKE THAT AS END-OF-FILE(TEXT), SET EOF-FLAG
7446          : AND LEAVE.
7447
7448          : INPUTS:
7449          :     FILLIN  BUFFER TO HOLD LINE OF BYTES READ FROM OPENED FILE
7450          :             (CR NOT INCLUDED, 0-BYTE TERMINATED)
7451          : OUTPUTS:
7452          :     BAD      IF NON-ZERO, ERROR IN READING A BYTE FROM FILE
7453          :     EOF      IF NON-ZERO, END OF FILE WAS ENCOUNTERED
7454          :     FILLIN   ASCIZ STRING THAT WAS READ AS CHAR-CR-LF STRING
7455          :             (CR-LF REMOVED)
7456
7457 053400 012702 000526' RDLIN: MOV #FILLIN,R2           :POINT R2 TO A LINE BUFFER
7458 053404 005001           CLR R1                         ; set success indicator to true
7459
7460          :*****+
7461          : THE FOLLOWING TWO LINES ARE EQUIVALENT TO DRS GETBYTE CALL. THEY HAVE
7462          : ERROR RIGHT NOW -- SHOULD DO A MOVB AND THEY ARE DOING A MOV OF RESULT
7463          :*****+
7464 053406 104426 1$: TRAP C$GETB
7465 053410 110012           MOVB R0,(R2)
7466
7467          :*****+
7468          : THIS SHOULD BE A BCOMPLETE. CALL DOESN'T SEEM TO BE SETTING CARRY
7469          : CORRECTLY -- 5/24/85
7470          :*****+
7471 053412 BCOMPLETE 2$: BR IF READ-BYTE SUCESSFUL
7472 053414 012701 177777    MOV #1,R1                 ; put EOF in R1
7473 053420 000416           BR 5$
7474
7475 053422 122712 000000 2$: CMPB #0,(R2)           ;IS this char is a null byte?
7476 053426 001003           BNE 3$                   ; br if not (look for <CR><LF>)
7477 053430 012701 177777    MOV #1,R1                 ; ... put EOF in R1
7478 053434 000410           BR 5$                   ; ... and leave!
7479 053436 122712 000015 3$: CMPB #15,(R2)          ;IS THE CHARACTER A <CR>
7480 053442 001761           BEQ 1$                   ; BR IF YES (GO BACK TO GET <LF>)
7481 053444 122712 000012           CMPB #12,(R2)          ;IS THE CHARACTER A <LF>
7482 053450 001402           BEQ 5$                   ; BR IF YES (TERMINATE AND LEAVE)
7483 053452 005202           INC R2                  ; IF NO, LEAVE CHAR IN BUFFER
7484 053454 000754           BR 1$                   ; AND GO GET MORE CHARS
7485
7486 053456 105012 5$: CLRB (R2)
7487 053460           RETURN R1
7488
7489
7490
7491          :--+
7492          : Name - SELMSG

```

OPERATOR SELECTED MESSAGE STORAGE

```

7493          ; Functional Description
7494          ; This routine will take the operator selected message from the
7495          ; command line input string buffer and put it into a buffer at
7496          ; location OPSLBF.
7497
7498          ; Inputs - P1 - ADDRESS OF OPERATOR SELECTED MESSAGE IN
7499          ; INPUT STRING
7500          ; Outputs - Implicit -
7501          ; The buffer at OPSLBF will contain the ASCII operator selected
7502          ; input string followed by a null character
7503
7504          ; Side Effects - none
7505
7506          ; Subordinate Routines - none
7507
7508          ; Calling Procedure: CALL SELMSG P1
7509
7510          ; Register Usage -
7511          ; R1 - address of input string
7512          ; R2 - address of output string
7513
7514          ;--+
7515
7516 053464          SELMSG: P$POP    R1      ;PUT ADDRESS OF OPR. SEL ASCII STRING INTO R1
7517 053466 012702 001722'    MOV     #OPSLBF,R2   ;PUT ADDRESS OF OUTPUT BUFFER INTO R2
7518 053472 122711 000045      CMPB    #45,(R1)  ; IS IT HEX DATA (first char a '#')?
7519 053476 001034          BNE     4$       ; branch if not
7520 053500 005201          INC     R1       ; point past data type indicator
7521 053502 010103          MOV     R1, R3   ; point to source string
7522 053504 105713          TSTB    (R3)    ; look for end of string
7523 053506 001405          BEQ     3$       ; branch if end
7524 053510 122713 000057      CMPB    #57,(R3)  ; is it a "/" delimiter
7525 053514 001402          BEQ     3$       ; branch if yes
7526 053516 005203          INC     R3       ; bump pointer
7527 053520 000771          BR     1$       ; continue counting
7528 053522 160103          SUB    R1, R3   ; calculate number of bytes
7529 053524          CALL    HXFORM  R1, #OPSLBF, R3 ; convert to hex
7530 053542          P$POP   R0,R4   ; get return status
7531 053546 001420          BEQ     12$      ; branch if success
7532 053550 112737 177777 001301'      MOVB    #-1,P$GDBD  ; set error flag
7533 053556          ERSSOFT 38,EMSG44
7534 053566 000412          BR     13$      ;--+
7535 053570          4$:
7536 053570 005003          CLR    R3       ;CLEAR CHARACTER COUNTER
7537 053572 105711          5$:
7538 053574 001403          TSTB    (R1)    ;CHECK FOR END OF STRING
7539 053576 112122          BEQ     10$      ;GO TO 10$ IF END
7540 053600 005203          MOVB    (R1)+,(R2)+ ;ELSE, MOVE BYTE TO OUTPUT BUFFER
7541 053602 000773          INC     R3       ;COUNT NUMBER OF CHARACTERS IN INPUT BUFFER
7542 053604 112712 000000 10$: MOVB    #0,(R2)  ;GO DO MORE CHARACTERS
7543 053610 010337 001446' 12$: MOV     R3,MSG6C ;PUT ZERO AT END OF OUTPUT BUFFER
7544 053614          13$: RETURN ;STORE NUMBER OF CHARACTERS FOR USE IN BUF. BUILDING
7545
7546
7547          ;--+
7548          ; Name - ENTRND          ENTER NODE IN TABLE
7549          ; Functional Description

```

7550 ; This routine is used to enter a node in the node table.
7551 ;
7552 ; Inputs - Implicit -
7553 ; ADRBUF - contains the node address to add to the node table
7554 ;
7555 ; Outputs - Explicit -
7556 ; P1 - zero if successful, -1 if table is full already
7557 ;
7558 ; Calling Procedure: CALL ENTRND
7559 ; P\$POP P1
7560 ;
7561 ; Side Effects - none
7562 ;
7563 ; Subordinate Routines -
7564 ; FINDSL - used to find empty slot in node table
7565 ; REMAP - map node table into memory
7566 ; RETMEM - restore memory mapping
7567 ;
7568 ; Register Usage -
7569 ; R1 - pointer to node table
7570 ; R2 - pointer to node address to be added to the node table
7571 ; R3 - loop control
7572 ;
7573 ;---+
7574
7575 053616 ENTRND: CALL FINDSL ;FIND AVAILABLE SLOT IN TABLE
7576 053624 P\$POP R1 ;CHECK IF TABLE FULL
7577 053626 001403 BEQ 5\$;IF NOT FULL BR TO 5\$
7578 053630 P\$PUSH #-1 ;ELSE PUT FULL INDICATION ON STACK
7579 053634 000426 BR 20\$;RETURN
7580 053636 5\$: CALL REMAP #ONTAB ;allow access to node table
7581 053650 012703 000003 MOV #3,R3 ;SET INCR. COUNTER TO 6 (BYTES)
7582 053654 013701 001202' MOV SLOT,R1 ;MOV ADDRESS OF AVAILABLE SLOT TO R1
7583 053660 012702 001070' MOV #ADRBUF,R2 ;MOV ADDRESS OF NODE ADDRESS TO R2
7584 053664 012221 10\$: MOV (R2)++,(R1)+ ;MOV BYTE OF ADDRESS
7585 053666 005303 DEC R3 ;DECR. COUNTER
7586 053670 001375 BNE 10\$;CONTINUE UNTIL 6 BYTES TRANSFERRED
7587 053672 005201 INC R1 ;SET POINTER TO NODE TYPE LOCATION
7588 053674 113711 001200' MOVB NODTY,(R1) ;MOVE NODE TYPE INTO TABLE
7589 053700 CALL RETMEM ;restore memory mapping
7590 053706 P\$PUSH #0 ;PUT ADDRESS ADDED INDICATION ON STACK
7591 053712 20\$: RETURN ;RETURN
7592 ;---+
7593 ; Name - FINDSL FIND EMPTY SLOT IN NODE TABLE
7594 ;
7595 ; Functional Description
7596 ; This routine is used to find an empty slot in the node table.
7597 ;
7598 ; Inputs - none
7599 ;
7600 ; Outputs - Explicit -
7601 ; P1 - zero if found a slot, -1 if no room in the node table
7602 ;
7603 ; Implicit -
7604 ; SLOT - contains address of empty slot in node table
7605 ;
7606 ;

```

7607 : Calling Procedure: CALL FINDSL
7608 : P$POP P1
7609 :
7610 :
7611 :
7612 : Side Effects - none
7613 : Subordinate Routines -
7614 : REMAP - map node table into memory
7615 : RETMEM - restore memory mapping
7616 :
7617 : Register Usage -
7618 : R2 - pointer into node table
7619 :
7620 053714
7621 053726 012702 100000
7622 053732 022712 000000
7623 053736 001422
7624 053740 062702 000010
7625 053744 020227 110000
7626 053750 001370
7627 053752
7628 053776
7629 054002 000404
7630 054004 010237 001202'
7631 054010
7632 054014
7633 054022
7634
7635
7636 :---+
7637 : Name - FULSLT
7638 : Functional Description
7639 : This routine is used to locate an entry in the node table
7640 : that contains a valid node address.
7641 :
7642 : Inputs - none
7643 :
7644 : Outputs - Implicit
7645 : SLOT - contains either an address of a node address or
7646 : -1 if the end of the node table has been reached
7647 : Calling Procedure: CALL FULSLT
7648 :
7649 : Side Effects - none
7650 :
7651 : Subordinate Routines -
7652 : REMAP - map node table into memory
7653 : RETMEM - restore memory mapping
7654 :
7655 : Register Usage -
7656 : R1 - pointer into node table
7657 :
7658 :
7659 :
7660 054024
7661 054036 013701 001202'
7662 054042 020127 110000
7663 054046 001406
7664
7665
7666
7667
7668
7669
7670
7671
7672
7673
7674
7675
7676
7677
7678
7679
7680
7681
7682
7683
7684
7685
7686
7687
7688
7689
7690
7691
7692
7693
7694
7695
7696
7697
7698
7699
7700
7701
7702
7703
7704
7705
7706
7707
7708
7709
7710
7711
7712
7713
7714
7715
7716
7717
7718
7719
7720
7721
7722
7723
7724
7725
7726
7727
7728
7729
7730
7731
7732
7733
7734
7735
7736
7737
7738
7739
7740
7741
7742
7743
7744
7745
7746
7747
7748
7749
7750
7751
7752
7753
7754
7755
7756
7757
7758
7759
7760
7761
7762
7763
7764
7765
7766
7767
7768
7769
7770
7771
7772
7773
7774
7775
7776
7777
7778
7779
7780
7781
7782
7783
7784
7785
7786
7787
7788
7789
7790
7791
7792
7793
7794
7795
7796
7797
7798
7799
7799
7800
7801
7802
7803
7804
7805
7806
7807
7808
7809
7810
7811
7812
7813
7814
7815
7816
7817
7818
7819
7820
7821
7822
7823
7824
7825
7826
7827
7828
7829
7830
7831
7832
7833
7834
7835
7836
7837
7838
7839
7840
7841
7842
7843
7844
7845
7846
7847
7848
7849
7850
7851
7852
7853
7854
7855
7856
7857
7858
7859
7860
7861
7862
7863
7864
7865
7866
7867
7868
7869
7870
7871
7872
7873
7874
7875
7876
7877
7878
7879
7880
7881
7882
7883
7884
7885
7886
7887
7888
7889
7890
7891
7892
7893
7894
7895
7896
7897
7898
7899
7899
7900
7901
7902
7903
7904
7905
7906
7907
7908
7909
7910
7911
7912
7913
7914
7915
7916
7917
7918
7919
7920
7921
7922
7923
7924
7925
7926
7927
7928
7929
7930
7931
7932
7933
7934
7935
7936
7937
7938
7939
7940
7941
7942
7943
7944
7945
7946
7947
7948
7949
7950
7951
7952
7953
7954
7955
7956
7957
7958
7959
7960
7961
7962
7963
7964
7965
7966
7967
7968
7969
7970
7971
7972
7973
7974
7975
7976
7977
7978
7979
7980
7981
7982
7983
7984
7985
7986
7987
7988
7989
7990
7991
7992
7993
7994
7995
7996
7997
7998
7999
7999
8000
8001
8002
8003
8004
8005
8006
8007
8008
8009
8010
8011
8012
8013
8014
8015
8016
8017
8018
8019
8020
8021
8022
8023
8024
8025
8026
8027
8028
8029
8030
8031
8032
8033
8034
8035
8036
8037
8038
8039
8040
8041
8042
8043
8044
8045
8046
8047
8048
8049
8050
8051
8052
8053
8054
8055
8056
8057
8058
8059
8060
8061
8062
8063
8064
8065
8066
8067
8068
8069
8070
8071
8072
8073
8074
8075
8076
8077
8078
8079
8080
8081
8082
8083
8084
8085
8086
8087
8088
8089
8090
8091
8092
8093
8094
8095
8096
8097
8098
8099
8099
8100
8101
8102
8103
8104
8105
8106
8107
8108
8109
8110
8111
8112
8113
8114
8115
8116
8117
8118
8119
8120
8121
8122
8123
8124
8125
8126
8127
8128
8129
8130
8131
8132
8133
8134
8135
8136
8137
8138
8139
8140
8141
8142
8143
8144
8145
8146
8147
8148
8149
8150
8151
8152
8153
8154
8155
8156
8157
8158
8159
8160
8161
8162
8163
8164
8165
8166
8167
8168
8169
8170
8171
8172
8173
8174
8175
8176
8177
8178
8179
8180
8181
8182
8183
8184
8185
8186
8187
8188
8189
8190
8191
8192
8193
8194
8195
8196
8197
8198
8199
8199
8200
8201
8202
8203
8204
8205
8206
8207
8208
8209
8210
8211
8212
8213
8214
8215
8216
8217
8218
8219
8220
8221
8222
8223
8224
8225
8226
8227
8228
8229
8229
8230
8231
8232
8233
8234
8235
8236
8237
8238
8239
8239
8240
8241
8242
8243
8244
8245
8246
8247
8248
8249
8249
8250
8251
8252
8253
8254
8255
8256
8257
8258
8259
8259
8260
8261
8262
8263
8264
8265
8266
8267
8268
8269
8269
8270
8271
8272
8273
8274
8275
8276
8277
8278
8279
8279
8280
8281
8282
8283
8284
8285
8286
8287
8288
8289
8289
8290
8291
8292
8293
8294
8295
8296
8297
8298
8299
8299
8300
8301
8302
8303
8304
8305
8306
8307
8308
8309
8309
8310
8311
8312
8313
8314
8315
8316
8317
8318
8319
8319
8320
8321
8322
8323
8324
8325
8326
8327
8328
8329
8329
8330
8331
8332
8333
8334
8335
8336
8337
8338
8339
8339
8340
8341
8342
8343
8344
8345
8346
8347
8348
8349
8349
8350
8351
8352
8353
8354
8355
8356
8357
8358
8359
8359
8360
8361
8362
8363
8364
8365
8366
8367
8368
8369
8369
8370
8371
8372
8373
8374
8375
8376
8377
8378
8379
8379
8380
8381
8382
8383
8384
8385
8386
8387
8388
8389
8389
8390
8391
8392
8393
8394
8395
8396
8397
8398
8399
8399
8400
8401
8402
8403
8404
8405
8406
8407
8408
8409
8409
8410
8411
8412
8413
8414
8415
8416
8417
8418
8419
8419
8420
8421
8422
8423
8424
8425
8426
8427
8428
8429
8429
8430
8431
8432
8433
8434
8435
8436
8437
8438
8439
8439
8440
8441
8442
8443
8444
8445
8446
8447
8448
8449
8449
8450
8451
8452
8453
8454
8455
8456
8457
8458
8459
8459
8460
8461
8462
8463
8464
8465
8466
8467
8468
8469
8469
8470
8471
8472
8473
8474
8475
8476
8477
8478
8479
8479
8480
8481
8482
8483
8484
8485
8486
8487
8488
8489
8489
8490
8491
8492
8493
8494
8495
8496
8497
8498
8499
8499
8500
8501
8502
8503
8504
8505
8506
8507
8508
8509
8509
8510
8511
8512
8513
8514
8515
8516
8517
8518
8519
8519
8520
8521
8522
8523
8524
8525
8526
8527
8528
8529
8529
8530
8531
8532
8533
8534
8535
8536
8537
8538
8539
8539
8540
8541
8542
8543
8544
8545
8546
8547
8548
8549
8549
8550
8551
8552
8553
8554
8555
8556
8557
8558
8559
8559
8560
8561
8562
8563
8564
8565
8566
8567
8568
8569
8569
8570
8571
8572
8573
8574
8575
8576
8577
8578
8579
8579
8580
8581
8582
8583
8584
8585
8586
8587
8588
8589
8589
8590
8591
8592
8593
8594
8595
8596
8597
8598
8599
8599
8600
8601
8602
8603
8604
8605
8606
8607
8608
8609
8609
8610
8611
8612
8613
8614
8615
8616
8617
8618
8619
8619
8620
8621
8622
8623
8624
8625
8626
8627
8628
8629
8629
8630
8631
8632
8633
8634
8635
8636
8637
8638
8639
8639
8640
8641
8642
8643
8644
8645
8646
8647
8648
8649
8649
8650
8651
8652
8653
8654
8655
8656
8657
8658
8659
8659
8660
8661
8662
8663
8664
8665
8666
8667
8668
8669
8669
8670
8671
8672
8673
8674
8675
8676
8677
8678
8679
8679
8680
8681
8682
8683
8684
8685
8686
8687
8688
8689
8689
8690
8691
8692
8693
8694
8695
8696
8697
8698
8699
8699
8700
8701
8702
8703
8704
8705
8706
8707
8708
8709
8709
8710
8711
8712
8713
8714
8715
8716
8717
8718
8719
8719
8720
8721
8722
8723
8724
8725
8726
8727
8728
8729
8729
8730
8731
8732
8733
8734
8735
8736
8737
8738
8739
8739
8740
8741
8742
8743
8744
8745
8746
8747
8748
8749
8749
8750
8751
8752
8753
8754
8755
8756
8757
8758
8759
8759
8760
8761
8762
8763
8764
8765
8766
8767
8768
8769
8769
8770
8771
8772
8773
8774
8775
8776
8777
8778
8779
8779
8780
8781
8782
8783
8784
8785
8786
8787
8788
8789
8789
8790
8791
8792
8793
8794
8795
8796
8797
8798
8799
8799
8800
8801
8802
8803
8804
8805
8806
8807
8808
8809
8809
8810
8811
8812
8813
8814
8815
8816
8817
8818
8819
8819
8820
8821
8822
8823
8824
8825
8826
8827
8828
8829
8829
8830
8831
8832
8833
8834
8835
8836
8837
8838
8839
8839
8840
8841
8842
8843
8844
8845
8846
8847
8848
8849
8849
8850
8851
8852
8853
8854
8855
8856
8857
8858
8859
8859
8860
8861
8862
8863
8864
8865
8866
8867
8868
8869
8869
8870
8871
8872
8873
8874
8875
8876
8877
8878
8879
8879
8880
8881
8882
8883
8884
8885
8886
8887
8888
8889
8889
8890
8891
8892
8893
8894
8895
8896
8897
8898
8899
8899
8900
8901
8902
8903
8904
8905
8906
8907
8908
8909
8909
8910
8911
8912
8913
8914
8915
8916
8917
8918
8919
8919
8920
8921
8922
8923
8924
8925
8926
8927
8928
8929
8929
8930
8931
8932
8933
8934
8935
8936
8937
8938
8939
8939
8940
8941
8942
8943
8944
8945
8946
8947
8948
8949
8949
8950
8951
8952
8953
8954
8955
8956
8957
8958
8959
8959
8960
8961
8962
8963
8964
8965
8966
8967
8968
8969
8969
8970
8971
8972
8973
8974
8975
8976
8977
8978
8979
8979
8980
8981
8982
8983
8984
8985
8986
8987
8988
8989
8989
8990
8991
8992
8993
8994
8995
8996
8997
8998
8999
8999
9000
9001
9002
9003
9004
9005
9006
9007
9008
9009
9009
9010
9011
9012
9013
9014
9015
9016
9017
9018
9019
9019
9020
9021
9022
9023
9024
9025
9026
9027
9028
9029
9029
9030
9031
9032
9033
9034
9035
9036
9037
9038
9039
9039
9040
9041
9042
9043
9044
9045
9046
9047
9048
9049
9049
9050
9051
9052
9053
9054
9055
9056
9057
9058
9059
9059
9060
9061
9062
9063
9064
9065
9066
9067
9068
9069
9069
9070
9071
9072
9073
9074
9075
9076
9077
9078
9079
9079
9080
9081
9082
9083
9084
9085
9086
9087
9088
9089
9089
9090
9091
9092
9093
9094
9095
9096
9097
9098
9099
9099
9100
9101
9102
9103
9104
9105
9106
9107
9108
9109
9109
9110
9111
9112
9113
9114
9115
9116
9117
9118
9119
9119
9120
9121
9122
9123
9124
9125
9126
9127
9128
9129
9129
9130
9131
9132
9133
9134
9135
9136
9137
9138
9139
9139
9140
9141
9142
9143
9144
9145
9146
9147
9148
9149
9149
9150
9151
9152
9153
9154
9155
9156
9157
9158
9159
9159
9160
9161
9162
9163
9164
9165
9166
9167
9168
9169
9169
9170
9171
9172
9173
9174
9175
9176
9177
9178
9179
9179
9180
9181
9182
9183
9184
9185
9186
9187
9188
9189
9189
9190
9191
9192
9193
9194
9195
9196
9197
9198
9199
9199
9200
9201
9202
9203
9204
9205
9206
9207
9208
9209
9209
9210
9211
9212
9213
9214
9215
9216
9217
9218
9219
9219
9220
9221
9222
9223
9224
9225
9226
9227
9228
9229
9229
9230
9231
9232
9233
9234
9235
9236
9237
9238
9239
9239
9240
9241
9242
9243
9244
9245
9246
9247
9248
9249
9249
9250
9251
9252
9253
9254
9255
9256
9257
9258
9259
9259
9260
9261
9262
9263
9264
9265
9266
9267
9268
9269
9269
9270
9271
9272
9273
9274
9275
9276
9277
9278
9279
9279
9280
9281
9282
9283
9284
9285
9286
9287
9288
9289
9289
9290
9291
9292
9293
9294
9295
9296
9297
9298
9299
9299
9300
9301
9302
9303
9304
9305
9306
9307
9308
9309
9309
9310
9311
9312
9313
9314
9315
9316
9317
9318
9319
9319
9320
9321
9322
9323
9324
9325
9326
9327
9328
9329
9329
9330
9331
9332
9333
9334
9335
9336
9337
9338
9339
9339
9340
9341
9342
9343
9344
9345
9346
9347
9348
9349
9349
9350
9351
9352
9353
9354
9355
9356
9357
9358
9359
9359
9360
9361
9362
9363
9364
9365
9366
9367
9368
9369
9369
9370
9371
9372
9373
9374
9375
9376
9377
9378
9379
9379
9380
9381
9382
9383
9384
9385
9386
9387
9388
9389
9389
9390
9391
9392
9393
9394
9395
9396
9397
9398
9399
9399
9400
9401
9402
9403
9404
9405
9406
9407
9408
9409
9409
9410
9411
9412
9413
9414
9415
9416
9417
9418
9419
9419
9420
9421
9422
9423
9424
9425
9426
9427
9428
9429
9429
9430
9431
9432
9433
9434
9435
9436
9437
9438
9439
9439
9440
9441
9442
9443
9444
9445
9446
9447
9448
9449
9449
9450
9451
9452
9453
9454
9455
9456
9457
9458
9459
9459
9460
9461
9462
9463
9464
9465
9466
9467
9468
9469
9469
9470
9471
9472
9473
9474
9475
9476
9477
9478
9479
9479
9480
9481
9482
9483
9484
9485
9486
9487
9488
9489
9489
9490
9491
9492
9493
9494
9495
9496
9497
9498
9499
9499
9500
9501
9502
9503
9504
9505
9506
9507
9508
9509
9509
9510
9511
9512
9513
9514
9515
9516
9517
9518
9519
9519
9520
9521
9522
9523
9524
9525
9526
9527
9528
9529
9529
9530
9531
9532
9533
9534
9535
9536
9537
9538
9539
9539
9540
9541
9542
9543
9544
9545
9546
9547
9548
9549
9549
9550
9551
9552
9553
9554
9555
9556
9557
9558
9559
9559
9560
9561
9562
9563
9564
9565
9566
9567
9568
9569
9569
9570
9571
9572
9573
9574
9575
9576
9577
9578
9579
9579
9580
9581
9582
9583
9584
9585
9586
9587
9588
9589
9589
9590
9591
9592
9593
9594
9595
9596
9597
9598
9599
9599
9600
9601
9602
9603
9604
9605
9606
9607
9608
9609
9609
9610
9611
9612
9613
9614
9615
9616
9617
9618
9619
96
```

7664 054050 022711 000000 CMP #0,(R1) ;CHECK IF EMPTY
7665 054054 001407 BEQ 20\$;IF YES, BR 20\$
7666 054056 010137 001202' MOV R1.SLOT ;ELSE PUT EMPTY LOC. ADDRESS INTO SLOT
7667 054062 000407 BR 30\$;RETURN
7668 054064 012737 177777 001202' 15\$: MOV #-1.SLOT ;PUT -1 INTO SLOT TO SHOW END OF TABLE
7669 054072 000403 BR 30\$;RETURN
7670 054074 062701 000010 ADD #8,,R1 ;INCR. POINTER TO NEXT LOCATION
7671 054100 000760 BR 10\$;CHECK NEXT LOC.
7672 054102 30\$: CALL RETMEM ;RESTORE MEMORY MAPPING
7673 054110 RETURN ;RETURN

7674
7675 ;---+
7676 : Name - CMPTWO COMPARE TWO BUFFERS
7677
7678 : Functional Description
7679 : This routine does a word by word comparison of two buffers
7680 : of arbitrary length. It will report the likeness of the
7681 : two buffers.
7682
7683 : Inputs - Explicit -
7684 : P1 - address of first buffer
7685 : P2 - address of second buffer
7686 : P3 - number of words to compare
7687
7688 : Outputs - Explicit -
7689 : P4 - 0 = buffers contained exact same data; -1 = they differed
7690
7691 : Calling Procedure: CALL CMPTWO P1,P2,P3
7692 : P\$POP P4
7693
7694 : Side Effects - none
7695
7696 : Subordinate Routines - none
7697
7698 : Register usage -
7699 : R1 - comparison indicator
7700 : R2 - pointer to first buffer
7701 : R3 - pointer to second buffer
7702 : R4 - number of words to compare
7703
7704 ;---+
7705 054112 CMPTWO: P\$POP R2,R3,R4 ;PUT ADDRESS OF STRING TO BE COMPARED IN R2 AND R3
7706 054120 022223 10\$: CMP (R2)+,(R3)+ ;DO TWO BYTE COMPARE?
7707 054122 001004 BNE 20\$; IF NO, EXIT W/ERROR
7708 054124 005304 DEC R4 ; DECREMENT NUMBER OF WORDS TO COMPARE
7709 054126 001374 BNE 10\$; KEEP GOING IF WE HAVE MORE TO DO
7710 054130 005001 CLR R1 ; INDICATE EQUALS!
7711 054132 000402 BR 30\$; AND LEAVE
7712 054134 012701 177777 20\$: MOV #-1,R1 ;PUT NO COMPARISON INDICATOR IN R1
7713 054140 30\$: RETURN R1

7714
7715 ;---+
7716 : Name - NTEXTI Extract Node table information
7717
7718 : Functional Description
7719 : This routine will take the information on one node in
7720 : the node table and default address table, format it and

7721 :
7722 : set up a "record" of information on that particular node.
7723 : Included in the information will be: current physical address,
7724 : default physical address, device type attached to the node,
7725 : logical node name, and DECnet address (AREA.NODE_NUMBER).
7726 :
7727 : Inputs - Implicit -
7728 : SLOT - contains address of node to work on
7729 :
7730 : Outputs - Implicit -
7731 : STRBUF - contains current physical address of node
7732 : STRBU1 - contains default physical address of node
7733 : LOGVAL - integer representing logical node number
7734 : DECNET - DECnet node number
7735 : AREA - DECnet area number
7736 :
7737 : Calling Procedure: CALL NTEXTI
7738 :
7739 : Side Effects - none
7740 :
7741 : Subordinate Routines -
7742 : BINHEX - convert node address into ascii string
7743 : GETTYP - set device type attached to node
7744 : REMAP - map node table into memory
7745 : RETMEM - restore memory mapping
7746 :
7747 : Register Usage -
7748 : R1, R2, R3 - scratch
7749 :
7750 054144 :---
7751 : NTEXTI:
7752 :---+
7753 : Setup the current node address in the buffer STRBUF
7754 :---+
7755 054144 CALL REMAP #ONTAB ;ALLOW ACCESS TO NODE TABLE
7756 054156 CALL BINHEX SLOT,#6,#STRBUF ;PUT ASCII ADDRESS INTO BUFFER
7757 :---+
7758 :---+
7759 : Setup the default hardware address in the buffer STRBU1
7760 :---+
7761 054200 013703 001202' MOV SLOT,R3 ;GET POINTER TO NODE TABLE
7762 054204 062703 010000 ADD #DEFNOD,R3 ;POINT R3 TO DEFAULT HARDWARE ADDR.
7763 054210 CALL BINHEX R3,#6,#STRBU1 ;CONVERT BINARY ADDRESS TO ASCII
7764 :---+
7765 :---+
7766 :---+
7767 : Call GETTYP to setup a string describing the device type in TYPADR
7768 :---+
7769 :---+
7770 054230 062703 000007 ADD #7,R3 ;POINT TO BYTE WITH NODE TYPE
7771 054234 CALL GETTYP R3 ;GET NODE TYPE!!
7772 :---+
7773 :---+
7774 : Setup the logical node number in the variable LOGVAL
7775 :---+
7776 054244 013702 001202' MOV SLOT,R2 ;POINT R2 TO NODE TABLE

7778 054250 162702 100000 SUB #NODTAB,R2 ;CALCULATE THE LOGICAL NAME ...
7779 054254 006202 ASR R2
7780 054256 006202 ASR R2 ;... LOG. NAM = (SLOT-#NODTAB)/8
7781 054260 006202 ASR R2
7782 054262 010237 001162' MOV R2,LOGVAL ;SAVE LOGICAL NAME
7783
7784 ;---+
7785 ; Setup the DECnet address in the variables AREA and DECNET
7786 ;---+
7787
7788 054266 013701 001202' MOV SLOT,R1 ;address of node binary > R1
7789 054272 062701 000002 ADD #2,R1 ;point to DECnet indicator
7790 054276 121127 000004 CMPB (R1),#04 ;is this a DECnet node?
7791 054302 001405 BEQ 30\$;branch if it is
7792 054304 005037 002054' CLR DECNET ;otherwise clear area.number..
7793 054310 005037 002056' CLR AREA
7794 054314 000422 BR 40\$;and exit
7795 054316 062701 000002 30\$: ADD #2, R1 ; point to decnet address
7796 054322 011137 002054' MOV (R1),DECNET ; and buffer it
7797 054326 042737 176000 002054' BIC #176000,DECNET ;clear area number
7798 054334 011137 002056' MOV (R1), AREA
7799 054340 042737 001777 002056' BIC #1777,AREA ;clear node number
7800 054346 012701 000012 MOV #10.,R1
7801 054352 006037 002056' 35\$: ROR AREA ;shift it into position for print
7802 054356 005301 DEC R1
7803 054360 001374 BNE 35\$
7805
7806 054362 40\$: RETURN ;RETURN
7807
7808
7809 ;---+
7810 ; Functional Description
7811 ; This subroutine prints the information contained in a reply
7812 ; system id message, in English.
7813 ;
7814 ; Inputs - P1 - the address of a buffer that contains a reply system
7815 ; id message.
7816 ;
7817 ; Outputs - System id information
7818 ;
7819 ; Calling procedure - Call PRNTID P1
7820 ;
7821 ; Side effects - None
7822 ;
7823 ; Subordinate routines -
7824 ; GETIDA - get address of a particular field in the sys. ID msg.
7825 ; GETTYP - set up the device type
7826 ; REMAP - map node table into memory
7827 ; RETMEM - restore memory mapping
7828 ;
7829 ; Register Usage -
7830 ; R1 - used to hold field type identifier for sys. id
7831 ; R2 - scratch
7832 ; R3 - scratch
7833 ;
7834 ;---+

7835
 7836 054364 PRNTID: p\$pop R1 : Get address of system id
 7837 054366 CALL REMAP #0RRING : allow access to receive ring
 7838 054400 010137 003110' mov R1,temp : save it in TEMP
 7839
 7840 054404 062701 000006 add #sourcc,R1 : point R1 to source address
 7841 054410 call binhex R1,#6,#strbuf : put address in strbuf
 7842 054430 printf #simsg1,#strbuf : print remote node current address
 7843
 7844 054454 013701 003110' mov temp,R1 : restore address of system id
 7845 054460 016137 000016 003112' mov siccou(R1),temp1 : save char. count
 7846 054466 162737 000004 003112' sub #4,temp1 : skip code, pad, and receipt number
 7847
 7848 054474 call getida temp,#144 : get address of device type
 7849 054512 p\$pop R2 : save address in R2
 7850 054514 PRNTF #SIMSG7 : print device field label
 7851 054534 call GETTYP R2 : get the device type
 7852 054544 PRNTF TYPADR : print the device type
 7853
 7854 054564 062701 000024 add #siffid,R1 : let R1 point to first field identifier
 7855 054570 116102 000002 5\$: movb 2(R1),R2 : get field length in R2
 7856 054574 160237 003112' sub R2,temp1 : sub. field len. from char. count
 7857 054600 162737 000003 003112' sub #3,temp1 : sub. id and length fields from char. count
 7858
 7859 :---+
 7860 : To avoid word references on odd-byte boundaries, a field will be
 7861 : extracted from the system id, then justified on an even byte boundary.
 7862 : Also, the length field will be extended from a byte to a word with the
 7863 : upper byte being null.
 7864 :---+
 7865 054606 012703 003040' mov #tempbl,R3 : point R3 to temporary storage
 7866 054612 112123 movb (R1)++,(R3)++ : save two bytes for the identifier
 7867 054614 112123 movb (R1)++,(R3)++ : save two bytes for the identifier
 7868 054616 112123 movb (R1)++,(R3)++ : save the field length
 7869 054620 112723 000000 movb #0,(R3)+ : add a null byte to keep alignment
 7870
 7871 054624 112123 8\$: movb (R1)++,(R3)++ : save a byte of field value
 7872 054626 005302 dec R2 : any more bytes left for value
 7873 054630 003375 bgt 8\$: yes, indeed!!
 7874 054632 012703 003040' mov #tempbl,R3 : point R3 back to the beginning of field
 7875
 7876 054636 022713 000144 cmp #144,(R3) : was this the device type field?
 7877 054642 001002 bne 10\$: no
 7878 054644 000137 055434' jmp 100\$: if so skip it
 7879
 7880 054650 022713 000000 10\$: cmp #0,(R3) : This is an illegal field type
 7881 054654 001002 bne 11\$: this ain't it!!
 7882 054656 000137 055446' jmp 101\$: on illegal type - exit
 7883
 7884 054662 022713 000001 11\$: cmp #1,(R3) : Is this maintenance version field?
 7885 054666 001043 bne 20\$: Nay!
 7886 054670 116302 000004 movb 4(R3),R2 : get version number
 7887 054674 printf #simsg3,R2 : and print it
 7888 054716 116302 000005 movb 5(R3),R2 : get ECO number
 7889 054722 printf #simsg4,R2 : and print it
 7890 054744 116302 000006 movb 6(R3),R2 : get user ECO number
 7891 054750 printf #simsg5,R2 : and print it

7892	054772	000137	055434'		jmp	100\$: done with this field
7893							
7894	054776	022713	000002	20\$:	cmp	#2,(R3)	: is this the function field?
7895	055002	001015			bne	30\$: Nay!
7896	055004	016302	000004		mov	4(R3),R2	: get function code
7897	055010				printf	#simsg6,R2	: and print it
7898	055032	000137	055434'		jmp	100\$: done with this field
7899							
7900	055036	022713	000003	30\$:	cmp	#3,(R3)	: is this console user field?
7901	055042	001026			bne	40\$: Nay!
7902	055044	010302			mov	R3,R2	: get address of system address
7903	055046	062702	000004		add	#4,R2	:
7904	055052				call	binhex R2,#6,#strbuf	: put it into STRBUF
7905	055072				printf	#simsg8,#strbuf	: and print it
7906	055116	000546			br	100\$: done with this field
7907							
7908	055120	022713	000004	40\$:	cmp	#4,(R3)	: Is this reservation timer field?
7909	055124	001014			bne	50\$: Nay!
7910	055126	016302	000004		mov	4(R3),R2	: get reservation timer value
7911	055132				printf	#simsg9,R2	: and print it
7912	055154	000527			br	100\$: done with this field
7913							
7914	055156	022713	000005	50\$:	cmp	#5,(R3)	: is this console command size?
7915	055162	001014			bne	60\$: nay!
7916	055164	016302	000004		mov	4(R3),R2	: get console command size
7917	055170				printf	#msg10,R2	: and print it
7918	055212	000510			br	100\$: done with this field
7919							
7920	055214	022713	000006	60\$:	cmp	#6,(R3)	: is this console response size?
7921	055220	001014			bne	70\$: Nay!
7922	055222	016302	000004		mov	4(R3),R2	: get console response size
7923	055226				printf	#msg11,R2	: and print it
7924	055250	000471			br	100\$: done with this field
7925							
7926	055252	022713	000007	70\$:	cmp	#7,(R3)	: is this hardware address field?
7927	055256	001026			bne	80\$: Nay!
7928	055260	010302			mov	R3,R2	: get address
7929	055262	062702	000004		add	#4,R2	: of default hardware address
7930	055266				call	binhex R2,#6,#strbuf	: convert to readable form
7931	055306				printf	#msg12,#strbuf	: and print it
7932	055332	000440			br	100\$: done with this field
7933							
7934	055334	022713	000010	80\$:	cmp	#10,(R3)	: is this system time stamp
7935	055340	001023			bne	90\$: Nay!
7936	055342				printf	#msg13,4(R3),6(R3),10(R3),12(R3),14(R3)	; dump 10 bytes in octal
7937	055406	000412			br	100\$: done with this field
7938							
7939	055410	021327	000310	90\$:	cmp	(R3),#200.	: See if we've got communications
7940	055414	002007			bge	100\$: device specific information
7941	055416	021327	000144		cmp	(R3),#100.	: this will be in the range ...
7942	055422	003404			ble	100\$: ... 101 <= n <= 199
7943							
7944							
7945							
7946							
7947							
7948							

:---+
:---+
:---+
:---+
:---+
The field that is being looked at is relevant only to POSEIDON
communication servers at present. If further COM devices make use
of this field then this section will have to be expanded accordingly

```

7949 055424          CALL    POSEIDON R3      : call routine to handle this field
7950
7951 055434 005737 003112'   100$:  tst     temp1      : are we through w/ this message?
7952 055440 001402          beq    101$      : yes
7953 055442 000137 054570'   jmp    5$       : nope!
7954
7955 055446          101$:  CALL    RETMEM      : restore memory mapping
7956 055454          return           : good bye
7957
7958
7959          ;---+; Name - POSEIDON          print POSEIDON specific system ID fields
7960
7961          ; Functional Description:
7962          ; This routine is used to print out information contained in
7963          ; the communication device specific field of a system ID message,
7964          ; specifically for the DECserver 100 (POSEIDON) communications
7965          ; device. The values of the TYPE INFO field for these fields will
7966          ; be in the range 101 <= N <= 199 (decimal).
7967
7968          ; Inputs - P1 - pointer to block containing a device specific field
7969
7970          ; Outputs - none
7971
7972          ; Calling Procedure: CALL POSEIDON P1
7973
7974          ; Side Effects -
7975          ; 1.) Prints out the information contained in the field
7976
7977          ; Subordinate Routines - none
7978
7979          ; Register Usage -
7980          ; R1 - pointer to block containing a device specific field
7981
7982
7983 055456          ;---+; POSEIDON:::
7984 055456          P$POP  R1      : get pointer to system ID field
7985
7986 055460 021127 000145          CMP    (R1),#101.      : Is this the Diagnostic Status field?
7987 055464 001036          BNE    10$       : NO, branch.
7988 055466          PRINTF #POSDS      : print diagnostic header
7989 055506          PRINTF #POSDS0,4(R1)  : print word 0 of status
7990 055532          PRINTF #POSDS1,6(R1)  : print word 1 of status
7991 055556 000137 056016'   JMP    POSEXIT      : all through with field
7992
7993 055562 021127 000150          10$:  CMP    (R1),#104.      : Is this the Server Number
7994 055566 001014          BNE    20$       : NO, branch.
7995 055570          PRINTF #POSSN,4(R1)  : print the server number ...
7996 055614 000137 056016'   JMP    POSEXIT      : ... and leave
7997
7998 055620 021127 000146          20$:  CMP    (R1),#102.      : Is this ROM version number?
7999 055624 001011          BNE    30$       : NO, branch.
8000 055626          PRINTF #POSRVN      : Print field identifier message
8001 055646 000443          BR     60$       : ... and go print value
8002
8003 055650 021127 000147          30$:  CMP    (R1),#103.      : Is this Software Version number?
8004 055654 001011          BNE    40$       : NO, branch.
8005 055656          PRINTF #POSSVN      : print field identifier message ...

```

```

8006 055676 000427           BR    60$          ; ... and go print value
8007
8008 055700 021127 000151     40$:  CMP   (R1),#105.   ; Is this the Server's name?
8009 055704 001011           BNE   50$          ; NO, branch.
8010 055706                   PRINTF #POSNAM      ; print field identifier message ...
8011 055726 000413           BR    60$          ; ... and go print value
8012
8013 055730 021127 000152     50$:  CMP   (R1),#106.   ; Is this the Server's Location?
8014 055734 001030           BNE   POSEXIT      ; NO, didn't find match ... just exit
8015 055736                   PRINTF #POSLOC      ; print field identifier message
8016
8017
8018
8019
8020
8021
8022
8023 055756 062701 000004     60$:  ADD   #4,R1        ; point R1 past TYPE and LENGTH fields
8024 055762 010102           MOV   R1,R2        ; make R2 point there
8025 055764 066202 177776     ADD   -2(R2),R2      ; point R2 past VALUE field
8026 055770 112712 000000     MOVB #0,(R2)      ; stuff a NULL byte at end of string
8027
8028 055774                   PRINTF #POSSTR,R1    ; print the string
8029
8030 056016                   POSEXIT:RETURN      ; hasta la vista, brother!!
8031
8032 .sbttl GETIDA  get the address of a system id field
8033
8034
8035
8036
8037
8038
8039
8040
8041
8042
8043
8044
8045
8046
8047
8048
8049
8050
8051
8052
8053
8054
8055
8056
8057 056020
8058 056020
8059 056024
8060 056030
8061
8062 056042 016103 000016

GETIDA:
          p$pop  R1,R2        ; get address of string to search for
          p$push temp          ; need a temporary var., so save 'temp'
          CALL   REMAP #ORRING  ; allow access to receive ring
          mov    siccou(R1),R3    ; save character count in R3

```

```

8063 056046 162703 000004      sub    #4,R3          ; dec. char count to skip code, pad, and
8064                                add    #siffid,R1   ; receipt number
8065 056052 062701 000024      10$:   mov    #temp,R4    ; point R1 to first field ID
8066                                movb   (R1)+,(R4)+ ; let R4 point to temporary storage
8067 056056 012704 003110'      movb   (R1)+,(R4)+ ; save a byte of field identifier
8068 056062 112124              movb   (R1)+,(R4)+ ; save a byte of field identifier
8069 056064 112124              cmp    temp,R2      ; have we found the desired field?
8070 056066 023702 003110'      beq    20$           ; yes, return it
8071 056072 001412              movb   (R1)+,R4    ; get byte that has length field
8072                                sub    #3,R3          ; decrement character count for fields
8073 056074 112104              sub    R4,R3          ; keep going if more characters
8074                                bne   15$           ; didn't find it
8075 056076 162703 000003      mov    #0,R1          ; return error indicator
8076 056102 160403              br    22$           ; return address
8077 056104 001003              mov    #0,R1          ; let R1 point to next field
8078 056106 012701 000000      br    10$           ; continue to look
8079 056112 000404              15$:   add    R4,R1    ; point R1 to field value
8080                                br    10$           ; restore value in 'temp'
8081 056114 060401              20$:   add    #1,R1          ; return address
8082 056116 000757              22$:   p$pop temp
8083                                return R1          ; print the device type
8084 056120 062701 000001      .sbttl PRTTYP
8085 056124
8086 056130
8087
8088
8089
8090
8091
8092
8093
8094
8095
8096
8097
8098
8099
8100 056134
8101 056134
8102 056136 122712 000001
8103 056142 001004
8104 056144 012737 012746' 001164'
8105 056152 000446
8106 056154 122712 000005
8107 056160 001004
8108 056162 012737 012756' 001164'
8109 056170 000437
8110 056172 122712 000011
8111 056176 001004
8112 056200 012737 012766' 001164'
8113 056206 000430
8114 056210 122712 000003
8115 056214 001004
8116 056216 012737 012776' 001164'
8117 056224 000421
8118 056226 122712 000013
8119 056232 001004

```

PRINT DEVICE TYPE

INPUTS	P1 - ADDRESS OF A BYTE THAT IS NODE TYPE
EXPLICIT OUTPUTS	NONE
IMPLICIT OUTPUTS	THE NODE TYPE WILL BE PRINTED IN PSEUDO-ENGLISH
SUBORDINATE ROUTINES	NONE
CALLING SEQUENCE	CALL PRTTYP P1

GETTYP:

P\$POP	R2	: get address node type
CMPB	#IDTUNA,(R2)	: DELUA/DEUNA?
BNE	50\$: branch if not
MOV	#UNA.TYPADR	: save una description
BR	100\$: leave
CMPB	#IDTQNA,(R2)	: QNA?
BNE	60\$: branch if not
MOV	#QNA.TYPADR	: save qna description
BR	100\$: leave
CMPB	#IDTLUA,(R2)	: LUA?
BNE	70\$: branch if not
MOV	#LUA.TYPADR	: save LUA description
BR	100\$: leave
CMPB	#IDTCNA,(R2)	: CNA?
BNE	80\$: branch if not
MOV	#CNA.TYPADR	: save CNA description
BR	100\$: leave
CMPB	#IDTCSA,(R2)	: CSA?
BNE	90\$: branch if not

```
8120 056234 012737 013006' 001164'      MOV    #SCA,TYPADR      : save CSA description
8121 056242 000412                      BR     100$              : leave
8122 056244 122712 000021      90$: CMPB   #IDTSRV,(R2)    : DECserver?
8123 056250 001004                      BNE    95$              : branch if not
8124 056252 012737 013016' 001164'      MOV    #SRV,TYPADR     : save DECserver description
8125 056260 000403                      BR     100$              : leave
8126 056262 012737 013032' 001164'      MOV    #UNKNWN,TYPADR  : save 'unknown' description
8127 056270                      95$: 100$: RETURN          : save CSA description
8128
8129
8130
8131      ;--+
8132      ; Name - EXELIS           Execute the Listen Command
8133
8134      ; Functional Description
8135      ; This routine implements the LISTEN command of the NIE.
8136      ; The purpose of the LISTEN command is to be able to monitor
8137      ; the activity of nodes on a network.
8138      ; Listening on the network consists of receiving
8139      ; all frames that pass a user specified filter. The filter
8140      ; may be on the frame's destination address, source address,
8141      ; protocol type, or any combination of the three.
8142      ; A log will be kept containing information on frames
8143      ; that pass the filter(s) including: destination address,
8144      ; source address, protocol type, packet length, and number
8145      ; of receipts. If a frame's characteristics match the first
8146      ; four then the number of receipts counter is incremented.
8147      ; A maximum of 30 entries will be stored in the log.
8148      ; A list of source addresses of frames that pass the
8149      ; filters will also be kept along with a count of the number
8150      ; of times that source address has been heard from
8151      ; The routine will print information on frames that pass
8152      ; filters every one millisecond or if there are no frames
8153      ; outstanding in the receive ring.
8154      ; The only way to stop listening is to type a control-C.
8155
8156      ; Inputs - none
8157
8158      ; Outputs - Implicit
8159      ; LISLOG - log containing frame characteristics
8160      ; LISNUM - the number of times the LISTEN command has been
8161      ; entered since the log has been cleared
8162      ; LISSEC - total number of seconds of listening
8163      ; LISMIN - total number of minutes of listening
8164      ; LISFSC - seconds to fill log
8165      ; LISFMN - minutes to fill log
8166      ; ADRLIS - source address list
8167
8168      ; Calling Procedure: JSR PC,EXELIS
8169
8170      ; Side Effects -
8171      ; 1.) control will pass to the DRS upon control-C
8172
8173      ; Subordinate Routines -
8174      ; CMPTWO - buffer comparison
8175      ; RECEIVE - receive frames
8176      ; PRLENT - print a listen event
```

PRTTYP print the device type

```

8177          : Register Usage -
8178          :   R1 - scratch
8179          :   R2 - pointer to buffer containing frame header
8180          :   R3 - pointer to received frame
8181          :   R4 - pointer to listen log/address list
8182          :
8183          :--+
8184 056272      EXELIS:::
8185
8186 056272      CALL DEVSTART
8187 056300 012702 002566' MOV #$WDM0,R2
8188 056304 012762 100000 000002 MOV #100000,2(R2)
8189 056312      CALL FUNCT #WDMODE
8190 056324      P$POP R2
8191 056326 001404 BEQ 5$
8192 056330      ERRDF 39,EMSG23,ERR1
8193
8194 056340 105737 001234' 5$: TSTB LISNUM
8195 056344 001007      BNE 10$
8196 056346 005037 001242' CLR LISMIN
8197 056352 005037 001244' CLR LISSEC
8198 056356 012737 000001 002052' MOV #1,TIMERS
8199
8200 056364 013737 001242' 002040' 10$: MOV LISMIN,TIMMIN
8201 056372 013737 001244' 002042' MOV LISSEC,TIMSEC
8202 056400      PRINTF #LISHD1
8203 056420      PRINTF #NEWLII
8204 056440 105237 001234' INCB LISNUM
8205
8206 056444      20$: BREAK
8207 056446      CALL RECEIVE
8208 056454      P$POP R2
8209 056456 001772 BEQ 20$
8210
8211 056460      25$: BREAK
8212 056462 013737 002040' 001242' MOV TIMMIN,LISMIN
8213 056470 013737 002042' 001244' MOV TIMSEC,LISSEC
8214 056476 013703 002100' MOV RRGNXT,R3
8215 056502      CALL GETRNX,#RRGNXT
8216 056514 016337 000006 001240' MOV 6(R3),LBYTEC
8217 056522 042737 170000 001240' BIC #170000,LBYTEC
8218 056530 016302 000010      MOV 10(R3),R2
8219
8220          :--+
8221          : Test to see if the received frame passes the user specified filters
8222          :--+
8223
8224 056534 105737 001254' TSTB DESFLG
8225 056540 001412 BEQ 40$
8226 056542      CALL CMPTWO R2,#DESFIL,#3
8227 056562      P$POP R1
8228 056564 001036 BNE 55$
8229
8230 056566 062702 000006 40$: ADD #SOURCC,R2
8231 056572 105737 001253' TSTB SOUFLG
8232 056576 001412 BEQ 50$
8233 056600      CALL CMPTWO R2,#SOUFIL,#3

```

8234 056620
8235 056622 001017 P\$POP R1 ; get equals indicator
8236 BNE 55\$; not equal, don't proceed
8237 056624 062702 000006 50\$: ADD #6,R2 ; point R2 to protocol type
8238 056630 105737 001255' TSTB PROFLG ; see if p.t. filter has been specified
8239 056634 001420 BEQ 60\$; no p.t. filter
8240 056636 CALL CMPTWO R2,#PROFIL,#1 ; check against filter
8241 056656 P\$POP R1 ; get equals indicator
8242 056660 001406 BEQ 60\$; passed filter
8243
8244 ;---+
8245 ; The received frame did not pass all filters, so release it and
8246 ; continue listening
8247 ;---+
8248 056662 000137 056444' 55\$: CALL RELBUF R3 ; release the receive buffer
8249 056672 000137 056444' JMP 20\$; and keep on listening
8250
8251
8252 056676 005237 001236' 60\$: INC LPACNM ; increment number of frames that passed filter
8253
8254 ;---+
8255 ; Now we've got a frame that has made it through the specified filters.
8256 ; R3 points to the buffer that contains the frame. Log information in
8257 ; listen log and address list.
8258 ;
8259 ;
8260 ; If all four fields - destination, source, protocol type, and character
8261 ; count - match an entry in the listen log, update the count for that
8262 ; entry. If not and there is room in the log, make a new entry.
8263 056702 012704 100000 MOV #LISLOG,R4 ; point R4 to listen log
8264 056706 016302 000010 MOV 10(R3),R2 ; point R2 to receive buffer
8265
8266 ;---+
8267 ; NOTE: the listen log has been set up such that individual entries have
8268 ; fields that are in the same relative locations as those in the received
8269 ; frame.
8270 ;---+
8271
8272 056712 020437 001232' 70\$: CMP R4,LISNXT ; have we checked all entries?
8273 056716 001434 BEQ 85\$; yes, try to add a new entry
8274 056720 CALL CMPEXT #0RRING,R2,#0LLLOG,R4,#7 ; see if dest., source, and p.t. match
8275 056746 P\$POP R1 ; get equals indicator
8276 056750 001014 BNE 80\$; not equal, check next entry
8277 056752 CALL REMAP #0LLLOG ; allow access to listen log
8278 056764 026437 000016 001240' CMP LBCOU(R4),LBYTEC ; see if byte counts match
8279 056772 001003 BNE 80\$; not equal, check next entry
8280 056774 005264 000020 INC LISCOU(R4) ; update count for this entry
8281 057000 000454 BR 100\$; go check address list
8282
8283 057002 062704 000022 80\$: ADD #LISENT,R4 ; point R4 to next entry in listen log
8284 057006 000741 BR 70\$; and keep checking
8285
8286 057010 105737 001252' 85\$: TSTB LISFUL ; has the log been filled?
8287 057014 001046 BNE 100\$; yes, go check address list
8288
8289 ;---+
8290 ; To make a new entry, just move dest, source, p.t., and char count into

```

8291 :      listen log and set count to one.
8292 ;--+
8293
8294 057016          CALL    MOVEXT #0RRING,R2,#0LLOG,R4,#7 ; move dest., source.. and p.t. into log
8295 057044          CALL    REMAP #0LLOG ; allow access to listen log
8296 057056 013764 001240' 000016    MOV    LBYTEC,LBCOU(R4) ; move byte count into log
8297 057064 012764 000001 000020    MOV    #1,LISCOU(R4) ; set count for this entry to one
8298
8299 057072 062737 000022 001232'    ADD    #LISENT,LISNXT ; update next entry pointer
8300 057100 023727 001232' 101034    CMP    LISNXT,#LISEND ; Is the log full?
8301 057106 001011                BNE    100$ ; No.
8302 057110 112737 177777 001252'    MOVB   #-1,LISFUL ; Raise log full flag
8303 057116 013737 002040' 001246'    MOV    TIMMIN,LOGFMN ; record the time it took to
8304 057124 013737 002042' 001250'    MOV    TIMSEC,LOGFSC ; fill the log
8305
8306 057132 012704 101034        100$: MOV    #ADRLIS,R4 ; point R4 to address list
8307 057136 062702 000006        ADD    #SOURCC,R2 ; point R2 to source address
8308
8309 057142 020437 001256'        110$: CMP    R4,ADRNXT ; have we checked all entries?
8310 057146 001430                BEQ    125$ ; YES, try to add entry to addr. list
8311
8312 057150          CALL    CMPEXT #0RRING,R2,#0LLOG,R4,#3 ; see if we have an address match
8313 057176          P$POP  R1 ; get equals indicator
8314 057200 001010                BNE    120$ ; if not equal, check next entry
8315 057202          CALL    REMAP #0LLOG ; allow access to listen log
8316 057214 005264 000006                INC    ADRCOU(R4) ; they were equal, so update count for this entry
8317 057220 000434                BR     140$ ; and go on
8318
8319 057222 062704 000010        120$: ADD    #ADRENT,R4 ; point R4 to next entry
8320 057226 000745                BR     110$ ; and keep checking
8321
8322 057230 020427 101414        125$: CMP    R4,#ADREND ; Have we filled the address list
8323 057234 001426                BEQ    140$ ; YES, can't add, but continue
8324
8325 ;--+
8326 ;      Add an entry to the address list by moving in the source address of the
8327 ;      received frame and setting the count to one.
8328 ;--+
8329
8330 057236          CALL    MOVEXT #0RRING,R2,#0LLOG,R4,#3 ; store source address
8331 057264          CALL    REMAP #0LLOG ; allow access to listen log
8332 057276 012764 000001 000006    MOV    #1,6(R4) ; set count for this addr. to one
8333 057304 062737 000010 001256'    ADD    #ADRENT,ADRNXT ; update next spot pointer
8334
8335 ;--+
8336 ;      With all that has gone on since we first received a good frame, there is
8337 ;      a good chance that we've received more. So, to keep up, do another
8338 ;      receive. If nothing's there, then print out the information from the
8339 ;      last frame processed.
8340 ;--+
8341
8342 057312          140$: CALL    RECEIVE ; See if anything's arrived
8343 057320          P$POP  R2 ; R2 is nonzero if we received something
8344 057322 001406                BEQ    150$ ; nothing there go print
8345
8346 057324 005737 002052'        145$: TST    TIMERS ; has time expired?
8347 057330 001012                BNE    160$ ; NO, don't try to print

```


8405
8406
8407 : Action routine to clear the listen data
8408
8409 060030 012737 100000 001232' ACTCLI: MOV #LISLOG,LISNXT : clear listen log
8410 060036 012737 101034 001256' MOV #ADRLIS,ADRNXT : clear address list
8411 060044 005037 001242' CLR LISMIN : reset elapsed time timer
8412 060050 005037 001244' CLR LISSEC :
8413 060054 005037 001246' CLR LOGFMN : reset log filled timer
8414 060060 005037 001250' CLR LOGFSC :
8415 060064 005037 001236' CLR LPACNM : clear number of frames that passed filter
8416 060070 005037 001234' CLR LISNUM : clear number of listen commands
8417 060074 105037 001252' CLRB LISFUL : clear listen log filled flag
8418 060100 105037 001253' CLRB SOUFLG : clear source filter presence
8419 060104 105037 001254' CLRB DESFLG : clear dest. filter presence
8420 060110 105037 001255' CLRB PROFLG : clear p.t. filter presence
8421
8422 060114 105037 001300' CLRB P\$NNUF : clear not enough flag
8423 060120 000207 RTS PC
8424
8425 :--
8426 : Name - PRLENT
8427 : Functional Description:
8428 : This routine prints the destination, source, protocol type, and
8429 : message length of a frame. The information to be printed may
8430 : be from the listen log or from an actual received frame.
8431 : Inputs - P1 - A pointer to an entry in the listen log or to a message
8432 : buffer.
8433 : P2 - The length of the entry or message
8434 :
8435 : Outputs - none
8436 :
8437 : Calling procedure - CALL PRLENT P1,P2
8438 :
8439 : Side effects - Information about the frame/listen log entry is printed at
8440 : the user's terminal.
8441 :
8442 : Subordinate Routines -
8443 : BINHEX - convert binary to an ASCII HEX string
8444 : Register Usage -
8445 : R2 - pointer to buffer that contains dest., source, and protocol
8446 : type
8447 : R3 - contains the length of the message
8448 :
8449 :
8450 :
8451 060122 PRLENT:
8452 060122 P\$POP R2,R3 : R2 points to an entry in the listen log
8453 060126 CALL BINHEX R2,#6,#STRBUF : convert dest addr. to HEX
8454 060146 PRINTF #DADDR,#STRBUF :
8455 060172 062702 000006 ADD #SOURCC,R2 : point R2 to source addr.
8456 060176 CALL BINHEX R2,#6,#STRBUF : convert it to HEX
8457 060216 PRINTF #SADDR,#STRBUF :
8458 060242 062702 000006 ADD #6,R2 : point R2 to protocol type
8459 060246 CALL BINHEX R2,#2,#STRBUF : convert it to HEX
8460 060266 PRINTF #PTYPE,#STRBUF :
8461 060312 PRINTF #CHARAC,R3 : print message length

```
8462 060334          RETURN           : return to the dubious caller!
8463
8464
8465      :--+
8466      : Name - MEMMAP
8467
8468      : Functional Description
8469      : All the CPUs that this diagnostic runs on have at
8470      : least an 18-bit bus providing for at least 128kW of
8471      : physical memory. Of this memory, only 32kW are strictly
8472      : allocated for the diagnostic. But, there is another 32kW
8473      : block that is available to the diagnostic by requesting
8474      : its use from the DRS. The management of the memory is
8475      : supposed to be done by the DRS. With the nature of this
8476      : diagnostic, speed being of the essence, it has become
8477      : necessary for me to skirt the DRS and handle the management
8478      : of this extended memory.
8479      : This routine will check with the DRS first to make
8480      : sure that the extended memory exists. It then will format
8481      : the extended memory in the following manner.
8482
8483      :-----+
8484      :> FUTURE USE          > 377776
8485      :>                      > 360000
8486      :-----+
8487      :> LISTEN LOG AND ADDRESS LIST FOR > 357776
8488      :> LISTEN COMMAND          > 340000
8489      :-----+
8490      :>                      > 337776
8491      :>
8492      :> SUMMARY TABLE          .
8493      :>                      > 300000
8494      :-----+
8495      :> DEFAULT ADDRESS TABLE > 277776
8496      :> NODE TABLE             > 260000
8497      :-----+
8498      :> TRANSMIT RING AND TRANSMIT BUFFERS > 257776
8499      :>                      > 240000
8500      :-----+
8501      :>                      > 237776
8502      :>
8503      :> RECEIVE RING AND RECEIVE BUFFERS .
8504      :>
8505      :>
8506      :-----+          . 200000
8507
8508      : To access this memory, KPAR4 and KPARS will be remapped
8509      : to point to two contiguous 4kW pages of extended memory.
8510
8511      : NOTE: The extended memory cannot be used by code that
8512      : resides at virtual addresses greater than or equal to
8513      : 100000(0). This is because these addresses would select
8514      : KPAR4 or KPARS which are pointing to extended memory.
8515      : (which, for obvious reasons, would completely screw everything
8516      : up).
8517
8518      : Inputs - none
```

```
8519          ; Outputs - none
8520
8521
8522
8523
8524
8525          ; Calling Procedure: CALL MEMMAP
8526
8527          ; Side Effects -
8528          1.) If the call to the DRS returns successfully, then
8529          extended memory will be formatted as above
8530
8531          ; Subordinate routines -
8532          REMAP - used to remap memory so that the transmit ring may be
8533          accessed
8534          RETMEM - used to return the mapping of memory to its original
8535          state
8536
8537
8538          ; Register Usage -
8539
8540
8541
8542 060336
8543 060336          MMU      OFF           ; let diagnostic control MMU
8544
8545
8546
8547          ;---+
8548          This diagram shows the structure of the transmit and receive rings
8549          note RING_BASE+10 is defined by this program. It is the virtual address
8550          of the buffer associated with the particular entry. In the DELUA/DEUNA
8551          documentation it is reserved for the port driver.
8552
8553          +-----+
8554          > Segment length    >
8555          >                         ) RING_BASE+0
8556
8557          +-----+
8558          > Segment physical   >
8559          > address             ) RING_BASE+2
8560
8561          +-----+
8562          > Status              >
8563          >                         ) RING_BASE+4
8564
8565          +-----+
8566          > Status & TDR/MLEN  >
8567          >                         ) RING_BASE+6
8568
8569          +-----+
8570          > Segment virtual     >
8571          > address             ) RING_BASE+10
8572
8573
8574
8575 060344          ;---+
8576          Now build the receive ring. There will be eight entries in
8577          the ring. The receive buffers follow directly after the receive
8578          ring or 120(0) away from the start of this segment of memory.
8579
8580          ;---+
8581          CALL    REMAP    #ORRING       ; enable access to portion of memory
```

```

8576 ; that has receive ring and buffers
8577
8578
8579 060356 012701 100120      MOV    #RBUFV1,R1      ; R1 has virt. addr. of first buffer
8580 060362 012702 100000      MOV    #RRING,R2      ; R2 has base address of receive ring
8581 060366 012703 000120      MOV    #R11501,R3      ; R3 points to the first receive buffer
8582 060372 012704 000010      MOV    #NO.NRR,R4      ; R4 has count of receive ring entries
8583
8584 060376 012722 002756      20$:   MOV    #RPKLEN,(R2)+ ; Set up length of segment (1518(D))
8585 060402 010322             MOV    R3,(R2)+      ; store address <15:01> of SEGB
8586 060404 012722 000001      MOV    #R11716,(R2)+ ; store address <17:16> of SEGB
8587 060410 005722             TST    (R2)+        ; leave room for buffer length
8588 060412 010122             MOV    R1,(R2)+      ; store virtual addr. of SEGB
8589 060414 062701 002756      ADD    #RPKLEN,R1      ; point R1 to next receive buffer
8590 060420 062703 002756      ADD    #RPKLEN,R3      ; point R3 to next receive buffer
8591 060424 005304             DEC    R4           ; decrement loop control
8592 060426 001363             BNE    20$          ; keep going if more to do
8593
8594
8595 :--+
8596 : Now build transmit ring and buffers. There will be two entries
8597 in the transmit ring. The transmit buffers follow the transmit
8598 ring directly or start at address 20(0)
8599 :--+
8600
8601 060430
8602
8603
8604 060442 012701 100050      MOV    #XBUFV1,R1      ; R1 has virt addr. of first buffer
8605 060446 012702 100000      MOV    #XRING,R2      ; R2 has base address of transmit ring
8606 060452 012703 040050      MOV    #X11501,R3      ; R3 points to the first transmit buffer
8607 060456 012704 000004      MOV    #NO.NTR,R4      ; R4 has count of transmit ring entries
8608
8609 060462 012722 002756      30$:   MOV    #RPKLEN,(R2)+ ; setup segment length
8610 060466 010322             MOV    R3,(R2)+      ; store address <15:01> of SEGB
8611 060470 012722 000001      MOV    #X11716,(R2)+ ; store address <17:16> of SEGB
8612 060474 005722             TST    (R2)+        ; leave room for buffer length
8613 060476 010122             MOV    R1,(R2)+      ; store virt. addr. of SEGB
8614 060500 062701 002756      ADD    #RPKLEN,R1      ; point R1 to next transmit buffer
8615 060504 062703 002756      ADD    #RPKLEN,R3      ; point R3 to next transmit buffer
8616 060510 005304             DEC    R4           ; decrement loop control
8617 060512 001363             BNE    30$          ; non-zero means more to do
8618
8619 :--+
8620 : The node table needs to be cleared.
8621 :--+
8622 060514
8623 060526 012702 100000      CALL   REMAP #NONTAB    ; allow access to node table
8624 060532 005022             MOV    #NODTBL,R2      ; let R2 point to the node table
8625 060534 020227 110000      40$:   CLR    (R2)+        ; DO clear the node location WHILE
8626 060540 001374             CMP    R2,#NODEND     ; there are more locations to clear
                                         BNE    40$          ; ENDDO
8627
8628
8629 :--+
8630 : The summary table must be cleared also
8631 060542
8632 060554 012702 100000      CALL   REMAP #OSTTAB    ; allow access to summary table
                                         MOV    #STATBL,R2      ; let R2 point to the summary table

```

8633 060560 005022 50\$: CLR (R2)+
8634 060562 020227 126000 CMP R2,**#STAEND**
8635 050566 001374 BNE 50\$; clear a word of summary table
8636 ; Are there more locations to clear?
8637 060570 CALL RETMEM ; YES, keep going
8638 ; restore mapping of upper memory
8639 060576 RETURN ; GOODBYE!
8640
8641 ;---+
8642 ; Name - REMAP
8643
8644 ; Functional Description
8645 ; This routine is called to remap the upper portion of our
8646 ; virtual address space to a new portion of physical memory.
8647 ; The portion being remapped is that which is pointed to by
8648 ; KPAR4 and KPAR5.
8649 ; The new value for KPAR4 is passed to the routine
8650 ; as a parameter. KPAR5 will be this parameter plus 200(0).
8651 ; The memory management unit will be enabled, also.
8652
8653 ; Inputs -
8654 ; P1 - new value for KPAR4
8655
8656 ; Outputs - none
8657
8658 ; Calling Procedure: CALL REMAP P1
8659
8660 ; Side Effects -
8661 ; 1.) KPAR4 and KPAR5 have been remapped to a new portion of
8662 ; physical memory
8663
8664 ; 2.) the CPU's memory management unit has been enabled
8665
8666 ; Subordinate Routines - none
8667
8668 ; Register Usage -
8669 ; R1 - holds new value for KPARs
8670
8671 ;---+
8672 060600 REMAP:::
8673
8674 ;---+
8675 ; Create new values for the new KPAR4 and KPAR5, then remap those
8676 ; registers.
8677 ;---+
8678
8679 060600 P\$POP R1 ; get new value for KPAR4
8680 060602 012737 000000 177572 MOV #MMUDIS,**#MMCSRO** ; disable memory management
8681 060610 010137 172350 MOV R1,**#KPAR4** ; remap KPAR4
8682
8683 060614 062701 000200 ADD #200,R1 ; create new value for KPAR5
8684 060620 010137 172352 MOV R1,**#KPAR5** ; remap KPAR5
8685
8686 060624 012737 000001 177572 MOV #MMUENA,**#MMCSRO** ; enable memory management unit
8687
8688 060632 RETURN ; that's all folks!
8689

```
8690      ;--+
8691      ; Name - RETMEM
8692      ;
8693      ; Functional Description
8694          This routine is called to restore the mapping of memory to
8695          its original state. The original values of KPAR4 and KPAR5
8696          are restored and the memory management unit is disabled.
8697      ;
8698      ; Inputs - Implicit
8699          NKPAR4 - the original value for KPAR4 (1000(0))
8700          NKPAR5 - the original value for KPAR5 (1200(0))
8701      ;
8702      ; Outputs - none
8703      ;
8704      ; Calling Procedure: CALL RETMEM
8705      ;
8706      ; Side Effects -
8707          1.) KPAR4 and KPAR5 are restored to their original values
8708      ;
8709      ; Subordinate Routines - none
8710      ;
8711      ; Register Usage - none
8712      ;
8713      ;--+
8714 060634      RETMEM:::
8715 060634 012737 000000 177572      MOV     #MMUDIS, @#MMCSR0      ; disable MMU
8716 060642 012737 001000 172350      MOV     #NKPAR4, @#KPAR4      ; restore KPAR4
8717 060650 012737 001200 172352      MOV     #NKPAR5, @#KPAR5      ; restore KPAR5
8718      ;
8719 060656      RETURN                ; LATER!!
8720      ;
8721      ; new Routine
8722      ;--+
8723      ; Name - PARVIR           SET UP PAR AND VIRTUAL ADDRESSES
8724      ;
8725      ; Functional Description
8726          This routine is used to modify KPAR4 and KPAR5 so that two
8727          portions of extended memory can be compared or data can be
8728          moved from one portion of extended memory to another.
8729      ;
8730          There are four inputs to the routine: two pairs, consisting
8731          of a base address of a data structure in extended memory
8732          and a virtual address within the data structure. Modifications
8733          may be necessary to the base and virtual addresses because
8734          some data structures are two pages big.
8735      ;
8736          The following pseudo-code illustrates the derivation of new base
8737          and virtual addresses:
8738      ;
8739          KPAR4 <- first base address
8740      ;
8741          TEST BIT 13 of first virtual address
8742      ;
8743          If SET THEN
8744              (* want to access the second page of a data structure.
8745                  Do this by adding 200(0) to KPAR4 *)
8746              KPAR4 <- KPAR4 + 200(0)
```

```
8747 :  
8748 : (* need to clear bit 13 of virtual address so it will  
8749 : map through KPAR4 *)  
8750 : CLEAR BIT 13 of first virtual address  
8751 :  
8752 : ENDIF  
8753 :  
8754 : (* ELSE no change on first pair *)  
8755 :  
8756 : KPAR5 <- second base address  
8757 :  
8758 : TEST BIT 13 of second virtual address  
8759 :  
8760 : IF SET THEN  
8761 : (* want to access the second page of a data structure.  
8762 : Do this by adding 200(0) to KPAR5 *)  
8763 : KPAR5 <- KPAR5 + 200(0)  
8764 :  
8765 : ELSE  
8766 : (* KPAR5 was correct, but need to set bit 13 of virtual  
8767 : address to map through KPAR5 *)  
8768 : SET BIT 13 of second virtual address  
8769 :  
8770 : ENDIF  
8771 :  
8772 : After the base and virtual addresses are derived, KPAR4 and  
8773 : KPAR5 are written and MMU is enabled.  
8774 :  
8775 : Inputs - Implicit - NOTE: because of speed considerations registers  
8776 : one through four must be set up before routine  
8777 : is called  
8778 : R1 - first base value  
8779 : R2 - first virtual address  
8780 : R3 - second base value  
8781 : R4 - second virtual address  
8782 :  
8783 : Outputs - none  
8784 :  
8785 : Calling Procedure: SET UP R1 - R4  
8786 : JSR PC,PARVIR  
8787 :  
8788 : Side Effects -  
8789 : 1.) KPAR4 and KPAR5 are remapped  
8790 : 2.) the memory management unit is enabled  
8791 : 3.) R1 - R4 may be modified  
8792 :  
8793 : Subordinate Routines - none  
8794 :  
8795 : Register Usage - as above  
8796 :  
8797 :---+  
8798 PARVIRT:: 060660  
8799 060660 012737 000000 177572 MOV #MMUDIS,0#MMCSRO ; disable memory management  
8800 :---+  
8801 : Test bit 13 of the source virtual address. If it is set, clear  
8802 : it and point KPAR4 to next page in memory  
8803 :
```

```

8804
8805 060666 032702 020000      ;---+
8806 050672 001404      BIT    #BIT13,R2      : Test bit 13 of source virtual addr.
8807 060674 042702 020000      BEQ    10$       : branch if clear
8808 060700 062701 000200      BIC    #BIT13,R2      : clear bit 13 to map through KPAR4
8809                                         ADD    #200,R1      : point KPAR4 to next page in memory
8810
8811                                         ;---+
8812                                         ; Test bit 13 of the destination virtual address. If it was set then
8813                                         ; point KPAR5 to next page in memory. If it was clear, then set it
8814                                         ; to map through KPAR5 as is.
8815
8816 060704 032704 020000      10$:   BIT    #BIT13,R4      : Test bit 13 of dest. virtual address
8817 060710 001403      BEQ    20$       : ... bit was clear
8818 060712 062703 000200      ADD    #200,R3      : point KPAR5 to next page in memory
8819 060716 000402      BR     30$       : ... and continue
8820
8821 060720 052704 020000      20$:   BIS    #BIT13,R4      : set bit 13 to map through KPARS
8822
8823 060724 010137 172350      30$:   MOV    R1,#KPAR4      : remap KPAR4 ...
8824 060730 010337 172352      MOV    R3,#KPARS      : ... and KPARS
8825
8826 060734 012737 000001 177572      MOV    #MMUENA,#MMCSRO      : enable memory management unit
8827
8828 060742 000207      RTS    PC
8829
8830
8831                                         ;---+
8832                                         ; Name - CMPEXT      COMPARE TWO PORTIONS OF EXTENDED MEMORY
8833
8834                                         ; Functional Description
8835                                         ; This routine is called to compare two portions of extended
8836                                         ; memory. It calls PARVIR to remap the two portions of
8837                                         ; memory, then does a word by word comparison of the length
8838                                         ; specified in the call to the routine by calling CMPTWO.
8839                                         ; It then calls RETMEM to remap memory to its original state.
8840
8841                                         ; Inputs -
8842                                         ; P1 - base address of string one
8843                                         ; P2 - virtual address of string one
8844                                         ; P3 - base address of string two
8845                                         ; P4 - virtual address of string two
8846                                         ; P5 - number of words to compare
8847
8848                                         ; Outputs -
8849                                         ; P6 - Comparison indicator -- 0 = compared/-1 = no compare
8850
8851                                         ; Calling Procedure: CALL CMPEXT P1, P2, P3, P4, P5
8852                                         ; P$POP P6
8853
8854                                         ; Side Effects - none
8855
8856                                         ; Subordinate Routines
8857                                         ; PARVIR - adjust the base and virtual addresses
8858                                         ; CMPTWO - compare the two strings
8859                                         ; RETMEM - remap memory to its original state
8860

```

```
8861 : Register Usage -
8862 : R1 - base address of string one (also return status)
8863 : R2 - virtual address of string one
8864 : R3 - base address of string two (also compare number)
8865 : R4 - virtual address of string two
8866 :
8867 :
8868 060744 CMPEXT:::
8869 060744 P$POP R1,R2,R3,R4 ; Set up registers for call to PARVIR
8870 JSR PC,PARVIR ; adjust base and virtual addresses
8871 060754 004737 060660'
8872 P$POP R3
8873 060760 CALL CMPTWO R2,R4,R3 ; R3 gets number of bytes to compare
8874 060762 ; do the compare
8875 060776 P$POP R1 ; R1 gets compare indicator
8876 061000 CALL RETMEM ; remap memory to its original state
8877
8878 061006 RETURN R1 ; chow!!
8879
8880 :
8881 ; Name - MOVEXT MOVE DATA IN EXTENDED MEMORY
8882 :
8883 ; Functional Description
8884 : This routine is used to move data between two portions
8885 : of extended memory. It calls PARVIR to adjust the base and
8886 : virtual addresses it will be referencing. Then does a word
8887 : by word transfer between the source and destination.
8888 : Finally it calls RETMEM to remap memory to its original state.
8889 :
8890 ; Inputs -
8891 : P1 - source base address
8892 : P2 - source virtual address
8893 : P3 - destination base address
8894 : P4 - destination virtual address
8895 : P5 - number of words to transfer between source and destination
8896 :
8897 ; Outputs - none
8898 :
8899 ; Side Effects -
8900 : 1.) the data transfer
8901 :
8902 ; Subordinate Routines
8903 : PARVIR - adjust base and virtual addresses
8904 : RETMEM - remap memory to its original state.
8905 :
8906 ; Register Usage -
8907 : R1 - source base address (and byte count of transfer)
8908 : R2 - source virtual address
8909 : R3 - destination base address
8910 : R4 - destination virtual address
8911 :
8912 :
8913 061012 MOVEXT:::
8914 061012 P$POP R1,R2,R3,R4 ; Setup R1 - R4 for call to PARVIR
8915 JSR PC,PARVIR ; adjust base and virtual addresses
8916 061022 004737 060660'
8917
```

```
8918 061026          P$POP R1           ; get byte count of transfer
8919
8920 061030 012224    10$: MOV  (R2)+,(R4)+   ; transfer a single word
8921 061032 005301    DEC  R1           ; decrement loop control
8922 061034 001375    BNE  10$          ; non-zero means more to do
8923
8924 061036          CALL  RETMEM        ; restore memory mapping
8925 061044          RETURN          ; thatsa all!!
8926
8928
8929          ; INSERT LOCAL STORAGE THAT IS USED ONLY
8930          ; DURING THIS TEST.
8931
8932
8933
8934          ; INSERT MESSAGES THAT ARE USED ONLY
8935          ; DURING THIS TEST.
8936
8938
8939          .EVEN
8940
8941 061046          ENDTST          ; BEGIN THE REMAINING TESTS ON NEW PAGES.
8942
8944
8945
8946
```

8949
8950 .SBTTL HARDWARE PARAMETER CODING SECTION
8951
8952 :++
8953 : THE HARDWARE PARAMETER CODING SECTION CONTAINS MACROS
8954 : THAT ARE USED BY THE SUPERVISOR TO BUILD P-TABLES. THE
8955 : MACROS ARE NOT EXECUTED AS MACHINE INSTRUCTIONS BUT ARE
8956 : INTERPRETED BY THE SUPERVISOR AS DATA STRUCTURES. THE
8957 : MACROS ALLOW THE SUPERVISOR TO ESTABLISH COMMUNICATIONS
8958 : WITH THE OPERATOR.
8959 :--
8960
8961 061050 BGNHRD
8962
8963 :*****
8964 : INSERT HARDWARE PARAMETER INTERPRETIVE CODE HERE. THIS CODE
8965 : IS USED BY THE SUPERVISOR TO INTERROGATE THE OPERATOR FOR
8966 : DEVICE INFORMATION TO PUT IN THE P-TABLE. THIS CODE IS USED
8967 : IN CONJUNCTION WITH THE DEFAULT P-TABLE TEMPLATE. THE MACROS
8968 : USED IN THIS SECTION ARE "GPRMD", "GPRMA" AND "GPRML".
8969 :*****
8970
8971
8972
8973 061052 GPRMA ASKCSR,0,0,160000,177776,YES : get csr address
8974 061062 GPRMA ASKVEC,2,0,0,776,YES : get vector address
8975 061072 GPRMD ASKPRI,4,0,340,0,7,YES : get priority level
8976
8977 061104 ENDHRD
8978
8979
8980 :*****
8981 : INSERT MESSAGES THAT ARE USED ONLY
8982 : DURING THE HARDWARE PARAMETER CODING SECTION.
8983 :*****
8984
8985
8986 061104 127 110 101 ASKCSR: .ASCIZ /WHAT IS THE PCSRO ADDRESS?/
061107 124 040 111
061112 123 040 124
061115 110 105 040
061120 120 103 123
061123 122 117 040
061126 101 104 104
061131 122 105 123
061134 123 077 000
8987 061137 127 110 101 ASKVEC: .ASCIZ /WHAT IS THE VECTOR ADDRESS?/
061142 124 040 111
061145 123 040 124
061150 110 105 040
061153 126 105 103
061156 124 117 122
061161 040 101 104
061164 104 122 105
061167 123 123 077
061172 000
8988 061173 127 110 101 ASKPRI: .ASCIZ /WHAT IS THE PRIORITY LEVEL?/
061176 124 040 111
061201 123 040 124
061204 110 105 040
061207 120 122 111

061212	117	122	111
061215	124	131	040
061220	114	105	126
061223	105	114	077
061226	000		

8989
8990

.EVEN

```
8992           .SBTTL SOFTWARE PARAMETER CODING SECTION
8993
8994
8995 :++ THE SOFTWARE PARAMETER CODING SECTION CONTAINS MACROS
8996 : THAT ARE USED BY THE SUPERVISOR TO BUILD P-TABLES. THE
8997 : MACROS ARE NOT EXECUTED AS MACHINE INSTRUCTIONS BUT ARE
8998 : INTERPRETED BY THE SUPERVISOR AS DATA STRUCTURES. THE
8999 : MACROS ALLOW THE SUPERVISOR TO ESTABLISH COMMUNICATIONS
9000 : WITH THE OPERATOR.
9001 :--
9002
9003 061230
9004
9005
9006 :-----+
9007 :      INSERT SOFTWARE PARAMETER INTERPRETIVE CODING HERE. THIS CODE
9008 :      IS USED BY THE SUPERVISOR TO INTERROGATE THE OPERATOR FOR
9009 :      SOFTWARE INFORMATION WHICH WILL BE PLACED IN THE SOFTWARE
9010 :      TABLE. THIS SECTION IS OPTIONAL.
9011 :-----+
9012
9013
9014 .EVEN
9015
9016 061232
9017
9018
9019
9020 :-----+
9021 :      INSERT MESSAGES THAT ARE USED ONLY
9022 :      DURING THE SOFTWARE PARAMETER CODING SECTION.
9023 :-----+
9024
9025
9026 061232
9027 061232
9028
9029
9030 :-----+
9031 :      THIS IS A PATCH AREA THAT SHOULD BE INCLUDED IN ALL DIAGNOSTICS.
9032 :      ADJUST THE SIZE TO FIT YOUR OWN PREFERENCES.
9033 :-----+
9034
9035
9036 061252
9037 061256
         LASTAD
L$LAST::
```

```
9038
9039
9041      ; HARDCODED P-TABLES MAY BE PLACED HERE BY USING THE SETUP MACROS.
9042      ; THIS SECTION IS OPTIONAL AND SHOULD BE REMOVED IF IT IS NOT BEING
9043      ; USED. CHANGE THE POINTER MACRO ARGUMENT TO REFLECT THE REMOVAL.
9044
9045
9046      ; THE P-TABLES ARE DELIMITED BY THE "BGNSETUP" AND "ENDSETUP" MACROS.
9047      ; THE "BGNSETUP" MACRO HAS ONE ARGUMENT WHICH IS THE NUMBER OF
9048      ; P-TABLE ENTRIES. EACH ENTRY IS DELIMITED BY THE "BGNPTAB" AND
9049      ; "ENDPTAB" MACROS. NEITHER OF THESE MACROS REQUIRE AN ARGUMENT.
9050
9052
9053      ; BGNSETUP      1
9054      ; BGNPTAB
9055      ; .WORD 0
9056      ; ENDPTAB
9057      ; ENDSETUP
9058
9059      000001          .END
```

Symbol table

AADDR	013723R	ADRENT= 000010 G	BNCCNT	002064R	CLRMSG	014177R	CPYLMR	014256R
ACTALP	044250R	ADRLIS= 101034 G	BNCLOG	000053	CLRQIK	000047	CRC	004000 G
ACTBLD	040630R	ADRNXT 001256R	BNCPKT	002060R	CLRSTA	000017 G	CRNALL	000032
ACTBLG	041714R	ALEMPT 013615R	BOE	000400 G	CLUPPR	000033	CRUN	000004
ACTCLI	060030R	ALHDR 013657R	BOOT	000005 G	CMDBUF	000732R	CSAVE	000006
ACTCMP	040224R	ALLNOD 017102R	BOUNCE	000052	CMDTY1	017370R	CSAVR4	000014
ACTCMS	047526R	ALPHA = 000000 G	BRDADR	002142R	CMDTY2	017375R	CSHCTR	000002 G
ACTCML	051304R	ANCHOR 027200R	BUFL	100000 G	CMDTY3	017405R	CSHMSG	000034
ACTCNT	047632R	AREA 002056RG	BUflen	003126RG	CMDTY4	017413R	CSIZE	000026
ACTCPY	044574R	ARGTY1 017462R	BUILD	000003	CMDTY5	017420R	CSLIST	000060
ACTCQK	040214R	ARGTY2 017470R	CADRER	012477R	CMDTY6	017424R	CTARGET	000000 G
ACTCSU	051460R	ARGTY3 017501R	CALPHA	000016	CMDTY7	017434R	CTYPE	000025
ACTCTT	044320R	ARGTY4 017512R	CASIST	000001 G	CMDTY8	017441R	CUNSAV	000041
ACTDES	053150R	ARGTY5 017523R	CBOADR	001166R	CMDTY9	017447R	CUNSVF	000045
ACTDFT	051544R	ARGTY6 017527R	CB0BUF	001042R	CMPBUF	003130RG	CZEROS	000020
ACTDIR	045750R	ARGTY7 017536R	CCCITT	000023	CMPERM	025774R	C\$AU	000052
ACTEXT	044134R	ASKCSR 061104R	CCITT	000005 G	CMPER1	026042R	C\$AUTO	000061
ACTFB8	042224RG	ASKPRI 061173R	CCLIST	000061	CMPER2	026115R	C\$BRK	000022
ACTFCT	051410R	ASKVEC 061137R	CCLMSG	000035	CMPERS	026142R	C\$BSEG	000004
ACTHLP	040240R	ASSEMB= 000010	CCLNAD	000004 G	CMPEXT	060744RG	C\$BSUB	000002
ACTIBB	042066RG	BA = 000000 G	CCLNAL	000010 G	CMPSTR	031610R	C\$CEFG	000045
ACTIDT	043254R	BCOUNT 003016RG	CCLSUM	000042	CMPTWO	054112R	C\$CLK	000062
ACTLIS	053206RG	BINDEC 034122RG	CCNTR	000036	CNA	012776R	C\$CLEA	000012
ACTMSG	044030R	BINHEX 031752RG	CCPYS	000027	CNDADR	000030	C\$CLOS	000035
ACTNAD	044652R	BIT0 = 000001 G	CDEFLT	000044	CNDLOG	000037	C\$CLP1	000006
ACTNAL	045034R	BIT00 = 000001 G	CDIR	000043	CNODEL	000031	C\$CVEC	000036
ACTNOD	040312R	BIT01 = 000002 G	CEXADR	000013	CNODE	000015	C\$DCLN	000044
ACTNUF	040174R	BIT02 = 000004 G	CEXIT	000020 G	CNTRO0	017546R	C\$DODU	000051
ACTNUL	040202R	BIT03 = 000010 G	CEXPRO	000056	CNTRO1	017626R	C\$DRPT	000024
ACTONE	044260R	BIT04 = 000020 G	CFLAG	002024R	CNTRO2	017675R	C\$DU	000053
ACTOPR	044330R	BIT05 = 000040 G	CFUNCT	000040	CNTRO3	017730R	C\$EDIT	000003
ACTPAT	047352R	BIT06 = 000100 G	CHARAC	013445R	CNTRO4	017775R	C\$ERDF	000055
ACTPRO	C 3222R	BIT07 = 000200 G	CLIACT	040012R	CNTRO5	020052R	C\$ERHR	000056
ACTRNA	045302R	BIT08 = 000400 G	CLIALP	000006	CNTRO6	020121R	C\$ERRO	000060
ACTRNL	046414R	BIT09 = 001000 G	CLIBIF	000003	CNTRO7	020160R	C\$ERSF	000054
ACTRUN	045146R	BIT1 = 000002 G	CLIBR	000002	CNTRO8	020230R	C\$ERSO	000057
ACTSAV	051612R	BIT10 = 002000 G	CLIBRX	011732R	CNTRO9	020302R	C\$ESCA	000010
ACTS88	042042RG	BIT11 = 004000 G	CLIDEC	000011	CNTRO10	020352R	C\$ESEG	000005
ACTSLI	057372R	BIT12 = 010000 G	CLIERRM	011623R	CNTRO11	020410R	C\$ESUB	000003
ACTSMS	047434R	BIT13 = 020000 G	CLIERR	000000	CNTRO12	020457R	C\$ETST	000001
ACTSND	051006R	BIT14 = 040000 G	CLIEXI	000001	CNTRO13	020524R	C\$EXIT	000032
ACTSOU	053112R	BIT15 = 100000 G	CLIMBG	011705R	CNTRO14	020571R	C\$GETB	000026
ACTSQK	040204R	BIT2 = 000004 G	CLIMUF	011654R	CNTRO15	020624R	C\$GETW	000027
ACTSR4	044242R	BIT3 = 000010 G	CLINUM	000005	CNTRO16	020666R	C\$GMAN	000043
ACTSUM	042674R	BIT4 = 000020 G	CLI OCT	000010	CNTRO17	020734R	C\$GPHR	000042
ACTSZE	044516R	BIT5 = 000040 G	CLISPA	000004	CNTRO18	021006R	C\$GPL0	000030
ACTTYP	044510R	BIT6 = 000100 G	CLISTR	000012	CNTRO19	021052R	C\$GPRI	000040
ACTUSF	052270R	BIT7 = 000200 G	CLITRE	003430R	CNTRO20	021123R	C\$INIT	000011
ACTXAD	044144R	BIT8 = 000400 G	CLI\$PM	011614R	CNTRO21	021162R	C\$INLP	000020
ACTZRO	044270R	BIT9 = 001000 G	CLKBR	002030R	COMMAND	030330RG	C\$MANI	000050
ACTOAL	044310R	BLD8UF 033120RG	CLKCSR	002026R	COMPAR	017302R	C\$MEM	000031
ACTIAL	044300R	BLDD00N 012244R	CLKEN	002036R	CONES	000017	C\$MSG	000023
ADR	- 000020 G	BLDFAS 032304RG	CLKHZ	002034R	COPRSL	000024	C\$OPEN	000034
ADRBUF	001070R	BLDLD 032040RG	CLKINT	027040RG	COUNT	003032RG	C\$PNTB	000014
ADRCOU	- 000006 G	BLDMSG 012151R	CLKSET	027014RG	CPATRN	000005	C\$PNTF	000017
ADREL	014621R	BLDREQ 032670RG	CLKVEC	002032R	CPOER	012553R	C\$PNTS	000016
ADREN	101414 G	BNCBUF 002062R	CLRCNT	000013 G	CPYCNT	003122RG	C\$PNTX	000015

C\$QIO = 000377	EA = 000001 G	ENP = 000400 G	F\$PWR = 000017	HELP2 = 006033R
C\$RDBU= 000007	EDPACK 031414RG	ENTRND 053616R	F\$RPT = 000012	HELP20 = 010020R
C\$REFG= 000047	EF.CON= 000036 G	ERRDLK 005730RG	F\$SEG = 000003	HELP21 = 010076R
C\$RESE= 000033	EF.NEW= 000035 G	ERRFLG 003020RG	F\$SOFT= 000005	HELP22 = 010161R
C\$REVI= 000003	EF.PWR= 000034 G	ERRMSG 005726RG	F\$SRV = 000010	HELP23 = 010262R
C\$RFLA= 000021	EF.RES= 000037 G	ERRNBR 005724RG	F\$SUB = 000002	HELP24 = 010362R
C\$RPT = 000025	EF.STA= 000040 G	ERROR 027316RG	F\$SM = 000014	HELP25 = 010473R
C\$SEFG= 000046	EMPSLT 013241R	ERRS = 040000 G	F\$TEST= 000001	HELP26 = 010601R
C\$SPRI= 000041	EMSG0 001616RG	ERRTYP 005722RG	GETCL 037524R	HELP27 = 010673R
C\$SVEC= 000037	EMSG01 021215R	ERR1 026624RG	GETCOM 033100R	HELP28 = 011001R
C\$TPRI= 000013	EMSG02 021254R	ERR2 026654RG	GETFNT= 000002 G	HELP29 = 011105R
C.COLL= 000074 G	EMSG03 021304R	ERR3 026742RG	GETIDA 056020R	HELP3 = 006126R
C.MREC= 000010 G	EMSG04 021346R	EVL = 000004 G	GETPCB= 000001 G	HELP30 = 011207R
C.MXMT= 000040 G	EMSG05 021400R	EXEBLD 040644R	GETRNX 033056RG	HELP31 = 011326R
C.PREC= 000004 G	EMSG06 021443R	EXEBNC 042354R	GETTYP 056134R	HELP32 = 011376R
C.PXMD= 000054 G	EMSG07 021503R	EXEHP 040250RG	GETXNX 033070RG	HELP33 = 011505R
C.PXMT= 000034 G	EMSG08 021556R	EXELIS 056272RG	G\$CNT0= 000200	HELP4 = 006177R
C.PXM2= 000050 G	EMSG09 021616R	EXIT = 000011	G\$DELM= 000372	HELP5 = 006250R
C.PXM3= 000044 G	EMSG1 001617RG	E\$END = 002100	G\$DISP= 000003	HELP6 = 006350R
C.RDAT= 000020 G	EMSG10 021646R	E\$LOAD= 000035	G\$EXCP= 000400	HELP7 = 006463R
C.RERB= 000014 G	EMSG14 021706R	FAADDR1= 000022 G	G\$HILI= 000002	HELP8 = 006574R
C.RERR= 000016 G	EMSG15 021761R	FAADDR2= 000032 G	G\$LOLI= 000001	HELP9 = 006664R
C.RLEX= 000032 G	EMSG16 022014R	FAADDR3= 000042 G	G\$NO = 000000	HEXBIN 031632RG
C.RLIN= 000030 G	EMSG18 022067R	FAADDR4= 000052 G	G\$OFFS= 000400	HEXC 031730R
C.RMDB= 000024 G	EMSG19 022146R	FAFCT1= 000020 G	G\$OFSI= 000376	HLPEND 001412R
C.SECS= 000002 G	EMSG2 001620RG	FAFCT2= 000030 G	G\$PRMA= 000001	HLPTAB 001310R
C.XABB= 000066 G	EMSG20 022204R	FAFCT3= 000040 G	G\$PRMD= 000002	HN = 031606R
C.XABT= 000070 G	EMSG22 022236R	FAFCT4= 000050 G	G\$PRML= 000000	HOE = 100000 G
C.XDAT= 000060 G	EMSG23 022265R	FASIST 003366RG	G\$RADA= 000140	HXERR 031574R
C.XMDB= 000064 G	EMSG24 022332R	FASKIP= 000016 G	G\$RADB= 000000	HXEXIT 031600R
COALT = 000022	EMSG25 022405R	FATFLG 003002RG	G\$RADD= 000040	HXFORM 031504RG
C1ALT = 000021	EMSG26 022474R	FATI = 000400 G	G\$RADL= 000120	IBE = 010000 G
DADDR 013424R	EMSG3 001621RG	FDATA1= 000032 G	G\$RADO= 000020	ICAB = 040000 G
DATCMP 033260RG	EMSG30 022530R	FDATA2= 000042 G	G\$XFER= 000004	IDENT = 000010
DECNET 002054RG	EMSG31 022575R	FILLIN 000526R	G\$YES = 000010	IDTCNA= 000003 G
DECSTR 034324RG	EMSG33 022636R	FINDSL 053714R	HDMSG1 015710R	IDTCSA= 000013 G
DEF = 002000 G	EMSG34 022654R	FORLOG 052160R	HDMSG2 015761R	IOTLUA= 000011 G
DEFADR 012700R	EMSG35 022724R	FRAM = 020000 G	HDMSG3 016034R	IDTQNA= 000005 G
DEFEND= 120000 G	EMSG36 022761R	FREMEM 002136RG	HDMSG4 016070R	IDTSRV= 000021 G
DEFNOD= 010000 G	EMSG37 023006R	FRESIZ 002134RG	HDMSG5 016145R	IDTUNA= 000001 G
DEFTBL= 110000 G	EMSG38 023052R	FULAST 017140R	HDMSG6 016216R	IDU = 000040 G
DEPADR 002234RG	EMSG4 001622RG	FULSLT 054024R	HDMSG7 016256R	IER = 020000 G
DESADR= 000055	EMSG41 023116R	FUNCT 030352RG	HDMSG8 016317R	ILADMS 012316R
DESFIL 001104RG	EMSG42 023162R	FUNTAB 002160RG	HDMSG9 016362R	ILADM1 012402R
DESFLG 001254R	EMSG43 023225R	F\$AU = 000015	HEADER= 000016 G	ILLADR 001206R
DESTIN= 000000 G	EMSG44 023274R	F\$AUTO= 000020	HELP = 000001	INIBNC= 000051
DEVICE 000524R	EMSG45 023340R	F\$BGN = 000040	HELP1 005732R	INICLN 037276R
DEVSTA 027454R	EMSG46 023375R	F\$CLEA= 000007	HELP10 006753R	INIEXI 037300R
DEVSTO 027656R	EMSG47 023442R	F\$DU = 000016	HELP11 007044R	INIT = 035662R
DFPTBL 000204RG	EMSG48 023512R	F\$END = 000041	HELP12 007142R	INIT1 = 035702R
DIAGMC= 000000	EMSG49 023537R	F\$HARD= 000004	HELP13 007247R	INTE = 000100 G
DIRCOM 045772R	EMSG5 001722RG	F\$HW = 000013	HELP14 007346R	INTR = 000200 G
DIRECT 017124R	EMSG50 023641R	F\$INIT= 000006	HELP15 007440R	ISR = 000100 G
DMPMEM= 000020 G	EMSG51 023716R	F\$JMP = 000050	HELP16 007453R	IXE = 004000 G
DMT = 004000 G	EMSG52 023773R	F\$MOD = 000000	HELP17 007542R	I\$AU = 000041
DN1PLG 003012RG	EMSG53 024040R	F\$MSG = 000011	HELP18 007645R	I\$AUTO= 000041
DTBHDR 013152R	EMSG54 024076R	F\$PROT= 000021	HELP19 007715R	I\$CLN = 000041

Symbol table

I\$DU - 000041	LOGFSC 001250R	L\$SPC 000056RG	MSG3C 001440R	NOD133 004516R
I\$HRD - 000041	LOGNAM 012706R	L\$SPCP 000020RG	MSG4 015643R	NOD134 004520R
I\$INIT - 000041	LOGNM 045126RG	L\$PTP 000024RG	MSG4C 001442R	NOD135 004522R
I\$MOD - 000041	LOGVAL 001162R	L\$STA 000030RG	MSG5C 001444R	NOD136 004526R
I\$MSG - 000041	LOPDIR 003260RG	L\$SW 000214RG	MSG6C 001446R	NOD137 004532R
I\$PROT - 000040	LOT - 000010 G	L\$TEST 000114RG	NCHN - 020000 G	NOD14 003540R
I\$PTAB - 000041	LPACNM 001236R	L\$TML 000014RG	NCMPAR - 000050	NOD140 004536R
I\$PWR - 000041	LST 031750R	L\$UNIT 000012RG	NETADR 012726R	NOD141 004542R
I\$RPT - 000041	LTMSSG 013736R	L10000 000212R	NEW 037250R	NOD142 004546R
I\$SEG - 000041	LUA 012766R	L10001 000214R	NEWLI1 013416R	NOD143 004552R
I\$SETU - 000041	LUPAIR 017113R	L10002 026652R	NEWLI2 013421R	NOD144 004556R
I\$SFT - 000041	L\$ACP 000110RG	L10003 026740R	NIHLT - 000006 G	NOD145 004562R
I\$SRV - 000041	L\$APT 000036RG	L10004 027012R	NIRCNT 003006RG	NOD146 004566R
I\$SUB - 000041	L\$AU 037516RG	L10005 027160R	NIUNI - 000007 G	NOD147 004572R
I\$TST - 000041	L\$AUT 000070RG	L10006 030326R	NKPAR4 - 001000 G	NOD15 003554R
J\$JMP - 000167	L\$AUTO 037306RG	L10007 035652R	NKPAR5 - 001200 G	NOD150 004574R
KEYWD1 001064R	L\$CCP 000106RG	L10011 037304R	NOCMPR 014441R	NOD151 004600R
KEYWD2 001066R	L\$CLEA 037310RG	L10012 037306R	NOD 014162R	NOD152 004604R
KPAR4 - 172350 G	L\$CO 000032RG	L10013 037506R	NODADR 012673R	NOD153 004622R
KPARS - 172352 G	L\$DEPO 000011RG	L10014 037514R	NODE - 000002	NOD154 004626R
KPAR6 - 172354 G	L\$DESC 000136RG	L10015 037522R	NODEND - 110000 G	NOD155 004632R
LBCOU - 000016 G	L\$DESP 000076RG	L10016 061046R	NODTBL - 100000 G	NOD156 004636R
LBYTEC 001240R	L\$DEVP 000060RG	L10017 061104R	NODTY 001200R	NOD157 004642R
LCAR - 004000 G	L\$DISP 000200RG	L10020 061232R	NODTYP 012720R	NOD16 003560R
LCLKEN - 0C0100 G	L\$DLY 000116RG	MEMMAP 060336RG	NODO 003430R	NOD160 004646R
LCOL - 010000 G	L\$DTP 000040RG	MESPAT 017005R	NOD1 003434R	NOD161 004652R
LCOUNT 013454R	L\$DTYP 000034RG	MESPA1 017056R	NOD10 003510R	NOD162 004656R
LDADR1 - 000022 G	L\$DU 037510RG	MMCSRO - 177572 G	NOD100 004270R	NOD163 004662R
LDADR2 - 000032 G	L\$DUT 000072RG	MMUDIS - 000000 G	NOD101 004274R	NOD164 004666R
LDATA - 000022 G	L\$DVTY 000122RG	MMUENA - 000001 G	NOD1C2 004300R	NOD165 004672R
LDFCT1 - 000020 G	L\$EF 000052RG	MORE - 010000 G	NOD103 004302R	NOD166 004714R
LDFCT2 - 000030 G	L\$ENVI 000044RG	MOVEXT 061012RG	NOD104 004306R	NOD167 004720R
LDMEM - 000021 G	L\$ERRT 005722RG	MSGAD 001450RG	NOD105 004322R	NOD17 003572R
LDRESP 011757R	L\$ETP 000102RG	MSGCNT 001432RG	NOD106 004326R	NOD170 004724R
LDSKIP - 000016 G	L\$EXP1 000046RG	MSGPRM 015213R	NOD107 004332R	NOD171 004730R
LEMSG 013563R	L\$EXP4 000064RG	MSGTAB 001414R	NOD11 003514R	NOD172 004734R
LENGTH 017273R	L\$EXP5 000066RG	MSGTY0 017322R	NOD110 004336R	NOD173 004740R
LFMSG 013464R	L\$HARD 061052RG	MSGTY1 017330R	NOD111 004342R	NOD174 004744R
LGERMS 026210R	L\$HIME 000120RG	MSGTY2 017335R	NOD112 004354R	NOD175 004750R
LINHLP 011752R	L\$HPCP 000016RG	MSGTY3 017343R	NOD113 004360R	NOD176 004754R
LISBUF 001214R	L\$HPTP 000022RG	MSGTY4 017350R	NOD114 004364R	NOD177 004760R
LISCOU - 000020 G	L\$HW 000204RG	MSGTY5 017355R	NOD115 004370R	NOD2 003440R
LISEND - 101034 G	L\$ICP 000104RG	MSGTY6 017363R	NOD116 004374R	NOD20 003576R
LISENT - 000022 G	L\$INIT 035662RG	MSGOC 001432R	NOD117 004400R	NOD200 004764R
LISFUL 001252R	L\$LADP 000026RG	MSG00 001466RG	NOD12 003520R	NOD201 005004R
LISHD1 013265R	L\$LAST 061256RG	MSG01 001616RG	NOD120 004404R	NOD202 005010R
LISHD2 013371R	L\$LOAD 000100RG	MSG02 001617RG	NOD121 004410R	NOD203 005014R
LISLOG - 100000 G	L\$LUN 000074RG	MSG03 001620RG	NOD122 004414R	NOD204 005020R
LISMIN 001242R	L\$MREV 000050RG	MSG04 001621RG	NOD123 004420R	NOD205 005024R
LISNUM 001234R	L\$NAME 000000RG	MSG05 001622RG	NOD124 004424R	NOD206 005030R
LISNXT 001232R	L\$PRI0 000042RG	MSG1 015263R	NOD125 004430R	NOD207 005044R
LISSEC 001244R	L\$PROT 035654RG	MSG1C 001434R	NOD126 004446R	NOD21 003602R
LISTEN - 000057	L\$PRT 000112RG	MSG11 015376R	NOD127 004452R	NOD210 005050R
LOC DST 031300R	L\$REPP 000062RG	MSG12 015511R	NOD13 003534R	NOD211 005064R
LOE - 040000 G	L\$REV 000010RG	MSG2 015551R	NOD130 004470R	NOD212 005070R
LOGDEL 014707R	L\$RPT 035642RG	MSG2C 001436R	NOD131 004474R	NOD213 005104R
LOGFMN 001246R	L\$SOFT 061232RG	MSG3 015602R	NOD132 004512R	NOD214 005110R

Symbol table

NOD215 005124R	NOD3 003444R	NOD73 004216R	N148\$ 004646R	N26\$ 003704R
NOD216 005130R	NOD30 003660R	NOD74 004236R	N149\$ 004656R	N28\$ 003730R
NOD217 005144R	NOD300 005622R	NOD75 004242R	N1491\$ 004652R	N29\$ 003752R
NOD22 003614R	NOD301 005626R	NOD76 004260R	N150\$ 004662R	N30\$ 003774R
NOD220 005150R	NOD302 005630R	NOD77 004264R	N151\$ 004672R	N300\$ 005636R
NOD221 005164R	NOD303 005634R	NORESP 017233R	N152\$ 004730R	N31\$ 004012R
NOD222 005170R	NOD304 005636R	NOTNUF = 000012	N153\$ 004740R	N310\$ 005642R
NOD223 005204R	NOD305 005642R	NO.NRR = 000010 G	N154\$ 004750R	N315\$ 005646R
NOD224 005210R	NOD306 005646R	NO.NTR = 000004 G	N1541\$ 004744R	N32\$ 004036R
NOD225 005224R	NOD307 005652R	NTBHDR 013042R	N155\$ 004754R	N320\$ 005652R
NOD226 005230R	NOD31 003662R	NTBLOV 014775R	N156\$ 004764R	N330\$ 005656R
NOD227 005234R	NOD310 005656R	NTEXTI 054144R	N157\$ 005014R	N331\$ 005666R
NOD23 003620R	NOD311 005662R	NULL = 000000	N16\$ 003514R	N332\$ 005672R
NOD230 005240R	NOD312 005666R	NULSTR 012625R	N160\$ 005020R	N335\$ 005676R
NOD231 005254R	NOD313 005672R	NXTDEL 053064R	N161\$ 005024R	N340\$ 005706R
NOD232 005260R	NOD314 005676R	NXTNDL 053032R	N162\$ 005070R	N350\$ 005712R
NOD233 005264R	NOD315 005702R	N10\$ 003434R	N163\$ 005110R	N50\$ 004060R
NOD234 005270R	NOD316 005706R	N100\$ 004150R	N164\$ 005130R	N70\$ 004064R
NOD235 005274R	NOD317 005712R	N101\$ 004154R	N165\$ 005150R	N72\$ 004070R
NOD236 005300R	NOD32 003664R	N102\$ 004174R	N166\$ 005170R	N74\$ 004100R
NOD237 005316R	NOD320 005716R	N104\$ 004216R	N167\$ 005210R	N76\$ 004120R
NOD24 003636R	NOD33 003700R	N106\$ 004242R	N168\$ 005234R	N78\$ 004124R
NOD240 005322R	NOD34 003704R	N108\$ 004264R	N17\$ 003540R	N80\$ 004126R
NOD241 005326R	NOD35 003724R	N11\$ 003444R	N170\$ 005240R	N81\$ 004132R
NOD242 005332R	NOD36 003730R	N110\$ 004270R	N1701\$ 005260R	N82\$ 004136R
NOD243 005336R	NOD37 003746R	N112\$ 004274R	N1702\$ 005270R	N90\$ 004142R
NOD244 005342R	NOD4 003450R	N1122\$ 004332R	N175\$ 005300R	N95\$ 004146R
NOD245 005362R	NOD40 003752R	N1123\$ 004370R	N1751\$ 005322R	OFLO - 010000 G
NOD246 005366R	NOD41 003770R	N1124\$ 004336R	N1752\$ 005332R	OK 016602R
NOD247 005402R	NOD42 003774R	N12\$ 003450R	N176\$ 005342R	OKFU 016742R
NOD25 003640R	NOD43 004010R	N120\$ 004302R	N177\$ 005366R	OKRE 016625R
NOD250 005406R	NOD44 004012R	N121\$ 004306R	N1771\$ 005406R	OKTR 016673R
NOD251 005412R	NOD45 004032R	N122\$ 004326R	N1772\$ 005416R	OLLOG = 003400 G
NOD252 005416R	NOD46 004036R	N123\$ 004360R	N1773\$ 005422R	ONE = 004000 G
NOD253 005422R	NOD47 004054R	N124\$ 004400R	N178\$ 005426R	ONEALT = 000003 G
NOD254 005426R	NOD5 003464R	N126\$ 004404R	N18\$ 003560R	ONES = 000001 G
NOD255 005432R	NOD50 004060R	N127\$ 004410R	N180\$ 005432R	ONTAB = 002600 G
NOD256 005436R	NOD51 004062R	N128\$ 004414R	N181\$ 005436R	OPNERR 011560R
NOD257 005456R	NOD52 004064R	N129\$ 004424R	N182\$ 005462R	OPRSEL = 000006 G
NOD26 003652R	NOD53 004070R	N13\$ 003470R	N183\$ 005500R	OPSLBF 001722R
NOD260 005462R	NOD54 004074R	N130\$ 004430R	N184\$ 005522R	ORRING = 002000 G
NOD261 005474R	NOD55 004100R	N132\$ 004452R	N185\$ 005540R	OSTAB = 003000 G
NOD262 005500R	NOD56 004114R	N134\$ 004474R	N186\$ 005544R	OTRING = 002400 G
NOD263 005516R	NOD57 004120R	N135\$ 004520R	N1861\$ 005550R	OUTBLK 052136R
NOD264 005522R	NOD6 003470R	N136\$ 004516R	N1862\$ 005560R	OWN = 100000 G
NOD265 005540R	NOD60 004124R	N14\$ 003474R	N1863\$ 005600R	O\$APTS = 000000
NOD266 005544R	NOD61 004126R	N140\$ 004522R	N1864\$ 005610R	O\$AU = 000000
NOD267 005550R	NOD62 004132R	N141\$ 004526R	N190\$ 005614R	O\$BGNR = 000001
NOD27 003656R	NOD63 004136R	N1412\$ 004542R	N20\$ 003576R	O\$BGNs = 000000
NOD270 005554R	NOD64 004142R	N142\$ 004552R	N200\$ 005616R	O\$DU = 000000
NOD271 005560R	NOD65 004146R	N1421\$ 004556R	N201\$ 005622R	O\$ERRT = 000000
NOD272 005574R	NOD66 004150R	N143\$ 004572R	N210\$ 005630R	O\$GNSW = 000000
NOD273 005600R	NOD67 004154R	N1431\$ 004562R	N22\$ 003620R	O\$POIN = 000001
NOD274 005604R	NOD7 003474R	N145\$ 004574R	N23\$ 003640R	O\$SETU = 000000
NOD275 005610R	NOD70 004170R	N146\$ 004600R	N231\$ 003656R	PART 034340RG
NOD276 005614R	NOD71 004174R	N1461\$ 004604R	N24\$ 003660R	PARVIR 060660RG
NOD277 005616R	NOD72 004212R	N147\$ 004636R	N25\$ 003664R	PASABT 016426R

Symbol table

PATCH	003132RG	P\$AEPR	001302R	RRGSRT	002070RG	STRT	= 000004 G	TSTMS2	016471R
PATTRN	017213R	P\$BLD	001275R	RRING	- 100000 G	SUMM	014167R	TSTMS3	016517R
PCBBO	002150RG	P\$BONC	001306R	RSET	- 000040 G	SUMMRY	- 000007	TSTMS4	016532R
PCBB2	002152RG	P\$BUFA	001260R	RSTT	- 000015 G	SUMMS1	026306R	TXI	- 010000 G
PCBB4	002154RG	P\$CNT	001266R	RTRY	- 002000 G	SUMMS2	026426R	TYPADR	001164R
PCBB6	002156RG	P\$CPYS	001174R	RTRYER	012067R	SUMMS3	026553R	T\$ARGC	- 000002
PCCALL	003124RG	P\$EXIT	034466R	RUN	- 000003 G	SUMMS5	026602R	T\$CODE	- 002032
PCEFLG	003004RG	P\$GDBD	001301R	RUNALL	045312R	SUMMS6	026616R	T\$ERRN	- 000047
PCEI	- 040000 G	P\$HEX	001277R	RUNCOM	047064R	SVCGL	- 000000	T\$EXCP	- 000000
PCLKCT	- 001600 G	P\$HLP	001276R	RUNDIR	045760R	SVCINS	- 177777	T\$FLAG	- 000040
PCLKEN	- 000111 G	P\$LIST	001274R	RUNLUP	046424R	SVCSUB	- 177777	T\$GMAN	- 000000
PCMMSG	025734RG	P\$MERR	001304R	RUNPAT	047362R	SVCTAG	- 177777	T\$HILI	- 000007
PCSRO	002106RG	P\$NCMP	001303R	RXI	- 020000 G	SVCTST	- 177777	T\$LAST	- 000001
PCSR0C	002116RG	P\$NNUF	001300R	R11501	- 000120 G	S\$LSYM	010000	T\$LOLI	- 000000
PCSR1	002110RG	P\$NUM	001270R	R11716	- 000001 G	S.BYTE	002776RG	T\$LSYM	- 010000
PCSR1C	002120RG	P\$PASS	001176R	SADDR	013431R	S.COMP	002774RG	T\$LTNO	- 000001
PCSR2	002112RG	P\$RADX	001272R	SAVED	015172R	S.LEN	002772RG	T\$NEST	- 177777
PCSR2C	002122RG	P\$SIZE	001172R	SCA	013006R	S.NREC	002770RG	T\$NSO	- 000005
PCSR3	002114RG	P\$TEXT	001305R	SELMMSG	053464R	S.REC	002766RG	T\$PTNU	- 000000
PCSR3C	002124RG	P\$TREE	001262R	SERI	- 100000 G	S.XFER	003000RG	T\$SAVL	- 177777
PCTO	- 000200 G	P\$TRV	034342RG	SETQIK	- 000046	TABCLR	015066R	T\$SEGL	- 177777
PDMD	- 000010 G	P\$TR5	034352R	SFPTBL	000214RG	TABEMT	014113R	T\$SUBN	- 000000
PFNOP	- 000000 G	P\$TYPE	001170R	SICCOU	- 000016 G	TABFUL	014041R	T\$TAGL	- 177777
PHYADR	002244RG	QNA	012756R	SIFFID	- 000024 G	TASIST	003302RG	T\$TAGN	- 010021
PNOP	- 000003 G	RASIST	003334RG	SIMSG1	024134R	TEMP	003110RG	T\$TEMP	- 000005
PNT	- 001000 G	RBFCNT	003014RG	SIMSG2	024206R	TEMPBL	003040RG	T\$TEST	- 000001
POSOS	025221R	RBUFV1	- 100120 G	SIMSG3	024261R	TEMP1	003112RG	T\$TSTM	- 177777
POSOSO	025250R	RCBI	- 002000 G	SIMSG4	024334R	TEMP2	003114RG	T\$TSTS	- 000001
POSOS1	025330R	RCVBUF	003030RG	SIMSG5	024407R	TEMP3	003116RG	T\$\$AU	- 010015
POSEID	055456RG	RCVERR	003026RG	SIMSG6	024462R	TENPWR	034254R	T\$\$AUT	- 010012
POSEXI	056016R	RDCNTS	- 000012 G	SIMSG7	024535R	TIMERS	002052R	T\$\$CLE	- 010013
POSLOC	025660R	RDDEFA	- 000002 G	SIMSG8	024605R	TIMER1	002046R	T\$\$DU	- 010014
POSNAM	025610R	RDLIN	053400R	SIMSG9	024657R	TIMER2	002050R	T\$\$HAR	- 010017
POSRVN	025470R	RD MODE	- 000014 G	SIRCPT	- 000022 G	TIMMIN	002040R	T\$\$HW	- 010000
POSSN	025410R	RDMULA	- 000006 G	SIZLMT	014342R	TIMOUT	003022RG	T\$\$INI	- 010011
POSSTR	025730R	RDPHYA	- 000004 G	SLOT	001202RG	TIMSEC	002042R	T\$\$MSG	- 010004
POSSVN	025540R	RDRNGS	- 000010 G	SLOT1	001204RG	TIMTCK	002044R	T\$\$PRO	- 010010
PREG14	027162RG	RDSTA	- 000016 G	SMSG10	024732R	TKPAR6	- 002400 G	T\$\$RPT	- 010007
PRI	- 002000 G	RDSYS	- 000022 G	SMSG11	025005R	TMRF	- 000012 G	T\$\$SOF	- 010020
PRIMLD	- 000001 G	READY	- 000002 G	SMSG12	025060R	TMRO	- 000011 G	T\$\$SRV	- 010006
PRI00	- 000000 G	RECAST	017174R	SMSG13	025132R	TRAST	017154R	T\$\$SW	- 010001
PRI01	- 000040 G	RECERR	012014R	SOUADR	- 000054	TRVACT	034470R	T\$\$TES	- 010016
PRI02	- 000100 G	RECEVE	031002RG	SOUFIL	001076RG	TRVADR	035366R	T1	037524RG
PRI03	- 000140 G	RELBUF	031220RG	SOUFLG	001253R	TRVALP	035224R	UAM	- 000200 G
PRI04	- 000200 G	REMAP	060600RG	SOURCC	- 000006 G	TRVBIF	034574R	UBTO	- 040000 G
PRI05	- 000240 G	REQID	003252RG	SOURCE	031412R	TRVBR	034564R	UCB10	002372RG
PRI06	- 000300 G	RESET	- 000000 G	SPACES	013256R	TRVBC	034510R	UCB11	002416RG
PRI07	- 000340 G	RESTOR	015201R	SRV	013016R	TRVDEC	034670R	UCB12	002442RG
PRLENT	060122R	RESTRT	037214R	STACK5	000214R	TRVERR	034526R	UCB13	002442RG
PRNTID	054364R	RETMEM	060634RG	STAEND	- 126000 G	TRVEXI	034546R	UCB20	002626RG
PROFIL	001112RG	RETRY	017247R	START	035762R	TRVNMA	034710R	UCB21	002626RG
PROFLG	001255R	RETRY5	003024RG	STATBL	- 100000 G	TRVNOB	034520R	UCB22	002670R
PROTOT	- 000014 G	RMTC	- 000010 G	STATUS	002600RG	TRVNUM	034702R	UCB23	002670R
PROT00	003034RG	RPKLEN	002756 G	STOP	- 000017 G	TRVOCT	034702R	UCB6	002272RG
PROT02	003036RG	RRGCUR	002074RG	STP	- 001000 G	TRVSPA	034616R	UCB7	002332RG
PTYPE	013437R	RRGLST	002104RG	STRBUF	001116R	TRVSTR	035270R	UDBB	002756RG
P\$ACT	001264R	RRGNXT	002100RG	STRBU1	001140R	TSTMS1	016451R	UNA	012746R

Symbol table

UNACSR	002126RG	WDMODE= 000015 G	XRGCUR	002072RG	X11716= 000001 G	\$RDMC	002262RG
UNAINI	027706RG	WDMULA= 000007 G	XRGYST	002102RG	ZEROS = 000002 G	\$RDMD	002556RG
UNAPISR	030130RG	WDPHYA= 000005 G	XRGNXT	002076RG	ZROALT= 000004 G	\$RDPH	002242RG
UNAPRI	002132RG	WDRNGS= 000011 G	XRGSRT	002066RG	\$CLRC 002546RG	\$RDRN	002362RG
UNAVEC	002130RG	WDSYS = 000023 G	XRING = 100000 G		\$CLRS 002606RG	\$RDST	002576RG
UNBOND	014533R	WRITES 033570RG	XSTRIN	053322R	\$DMEM 002616RG	\$RDSY	002650RG
UNIHLT-	000005 G	XBUFV1= 100050 G	X\$ = 000321		\$LMEM 002640RG	\$WDMC	002322RG
UNIT	002140RG	XFER 003120RG	X\$ALWA= 000000		\$PATCH 061232RG	\$WDMO	002566RG
UNKNMN	013032R	XFLAG 003010RG	X\$FALS= 000040		\$PNOP 002230RG	\$WDPH	002252RG
UNSMMSG	015133R	XMIT 030414RG	X\$OFFS= 000400		\$RDCN 002432RG	\$WDRN	002406RG
USCI	= 000400 G	XPKLEN= 002756 G	X\$TRUE= 000020		\$RDDE 002232RG	\$WTSY	002660RG
WAIT	027234RG	XPWR = 100000 G	X11501= 040050 G				

. ABS. 000000 000 (RW,I,GBL,ABS,OVR)
 061256 001 (RW,I,LCL,REL,CON)

Errors detected: 0

*** Assembler statistics

Work file reads: 344
 Work file writes: 336
 Size of work file: 30278 Words (119 Pages)
 Size of core pool: 19402 Words (74 Pages)
 Operating system: RSX-11M/PLUS (Under VAX/VMS)

Elapsed time: 00:12:59.40
 ZUACC,ZUACC/CR/-SP-SUPR11/ML.ZUACC