PRODUCT CODE: AC-T227C MC

PRODUCT NAME: CZUACCO DEUNA NI EXERCISER

PRODUCT DATE: JULY 3, 1985

MAINTAINER: JAMES CRITSER

AUTHOR: GARY MCCOY

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 XXDP+ 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

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 ↑C
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 PCSR0 ADDRESS? (0)  174510 ? ... type PCSR0 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]ooppair/[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
[BO]unce /<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/[TY]pe=a/[S]ize=n/[C]opies=m - allows the user to
                             modify the default message type, size
                             and copy count

[M]essage /[TE]xt =%<hex data string> - input user defined hex
                                        data
[M]essage /[TE]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 URS.

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.

Table of contents

```
                        .SBTTL  PROGRAM HEADER
                        ;       .ENABL  ABS,AMA
                        ;       .= 2000
                                .ENABL  AMA


                        .SBTTL  Program Macros


                                ;I$STACK macro
                                ;-------------

                                ;+++
                                ;The I$STACK macro facilitates initializing the R6 (hardware) stack
                                ;and the R5 (parameter) stack.  R5 is set to the stack low limit
                                ;(STAKLO) and the parameter stack grows upward.  R6 is set to the
                                ;stack high limit (STAKHI) and the hardware stack grows downward.
                                ;If there is a stack over-run, it will be detected by the PREG14
                                ;routine.
                                ;---

                        ;++
                        ; THE PROGRAM HEADER IS THE INTERFACE BETWEEN
                        ; THE DIAGNOSTIC PROGRAM AND THE SUPERVISOR.
                        ;--

000000                          POINTER BGNRPT


000000                          HEADER  CZUAC,C,0,0,1,PRIO7


                        ;
                        ; NAMES OF DEVICES SUPPORTED BY PROGRAM
                        ;
000122                          DEVTYP  <DEUNA,DELUA>

                        ; TEST DESCRIPTION
                        ;
000136                          DESCRIPT        <CZUAC DEUNA,DELUA NI EXERCISER>
                                .EVEN


                        ;
                        ; FORMAT STATEMENTS USED IN PRINT CALLS
                        ;
```

```
                                .SBTTL  DISPATCH TABLE

                                ;**
                                ; THE DISPATCH TABLE CONTAINS THE STARTING ADDRESS OF EACH TEST.
                                ; IT IS USED BY THE SUPERVISOR TO DISPATCH TO EACH TEST.
                                ;--

000176                                  DISPATCH 1

```

```
                                .SBTTL  DEFAULT HARDWARE P-TABLE

                                ;++
                                ; THE DEFAULT HARDWARE P-TABLE CONTAINS DEFAULT VALUES OF
                                ; THE TEST-DEVICE PARAMETERS.  THE STRUCTURE OF THIS TABLE
                                ; IS IDENTICAL TO THE STRUCTURE OF THE HARDWARE P-TABLES,
                                ; AND IS USED AS A "TEMPLATE" FOR BUILDING THE P-TABLES.
                                ;--

000202                                  BGNHW   DFPTBL

000204  174510                         .WORD   174510                  ; CSR
000206  000120                         .WORD   120                     ; VECTOR
000210  000240                         .WORD   PRIO5                   ; PRIORITY


000212                                  ENDHW
```

```
                                .SBTTL  SOFTWARE P-TABLE

                                ;++
                                ; THE SOFTWARE TABLE CONTAINS VARIOUS DATA USED BY THE
                                ; PROGRAM AS OPERATIONAL PARAMETERS.  THESE PARAMETERS ARE
                                ; SET UP AT ASSEMBLY TIME AND MAY BE VARIED BY THE OPERATOR
                                ; AT RUN TIME.
                                ;--

000212                                  BGNSW   SFPTBL


000214                                  ENDSW

                                .SBTTL  GLOBAL EQUATES SECTION




                                ;++
                                ; THE GLOBAL EQUATES SECTION CONTAINS PROGRAM EQUATES THAT
                                ; ARE USED IN MORE THAN ONE TEST.;--


000214                                  EQUALS
                                ;
                                ; BIT DIFINITIONS
                                ;
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" ARGUEMENT 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                          CLIOCT= 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                          CZEROS=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                          CCPYS=27
604     000030                          CNDADR=30
605     000031                          CNODAL=31
```

```
000032                          CRNALL=32
000033                          CLUPPR=33
000034                          CSHMSG=34
000035                          CCLMSG=35
000036                          CCNTR=36
000037                          CNDLOG=37
000040                          CFUNCT=40
000041                          CUNSAV=41
000042                          CCLSUM=42
000043                          CDIR=43
000044                          CDEFLT=44
000045                          CUNSVF=45
000046                          SETQIK=46
000047                          CLRQIK=47
000050                          NCMPAR=50
000051                          INIBNC=51
000052                          BOUNCE=52
000053                          BNCLOG=53
000054                          SOUADR=54
000055                          DESADR=55
000056                          CEXPRO=56
000057                          LISTEN=57
000060                          CSLIST=60
000061                          CCLIST=61

000000                          ALPHA==0                        ;MESSAGE TYPE VALUES
000001                          ONES==1
000002                          ZEROS==2
000003                          ONEALT==3
000004                          ZROALT==4
000005                          CCITT==5
000006                          OPRSEL==6

                        ;
                        ;       GLOBAL EQUATES FOR THE DEUNA/DELUA DRIVER
                        ;
                        ;Port Control and Status Register 0

100000          SERI    ==      BIT15           ; STATUS ERROR INTERRUPT
040000          PCEI    ==      BIT14           ; PORT COMMAND ERROR INTERRUPT
020000          RXI     ==      BIT13           ; RECEIVE RING INTERRUPT
010000          TXI     ==      BIT12           ; TRANSMIT RING INTERRUPT
004000          DNI     ==      BIT11           ; DONE INTERRUPT
002000          RCBI    ==      BIT10           ; RECEIVE BUFFER UNAVAILABLE
000400          USCI    ==      BIT08           ; UNSOLICITED STATE CHANGE INTERRUPT
000400          FATI    ==      BIT08           ; FATAL ERROR INTERERUPT
000200          INTR    ==      BIT07           ; INTERRUPT SUMMARY <15:08>
000100          INTE    ==      BIT06           ; INTERRUPT ENABLE
000040          RSET    ==      BIT05           ; DEUNA/DELUA RESET

                        ; PORT COMMANDS in bit 3 to bit 0
                        ; -------------------------------

000001                  GETPCB == bit00                 ; Get Address of Port Control Block
000002                  GETFNT == bit01                 ; Get Command in Port Control Block
000003                  PNOP == bit00!bit01             ; No operation performed
```

```
                        STRT  == bit02                  ;  Enable XMIT and RCVR
                        BCOT == bit02!bit00             ;  Boot , -> Prim load state,
                                                        ;   initate downline load

                        PDMD == bit03                   ; polling demand/wake up bit
                        TMRO == bit03!bit00             ; sanity timer enable ( =1 its on)
                        TMRF == bit03!bit01             ; Sanity Timer Off
                        RSTT == bit03!bit02!bit00       ; reset sanity timer
                        STOP == bit03!bit02!bit01!bit00 ; Suspend DEUNA/DELUA operation


                    ;Port Control and Status Register 1


                        XPWR == bit15                   ; transceiver power ok
                        ICAB == bit14                   ; port to link cable ok

                        ; self test error code in bit 13 to bit 08
                        PCTO  == bit07                  ; port command timeout

                        RMTC == bit03                   ; remote console reserved (=1)

                        ; port state in bit 2 to bit 0

                        RESET == 0                      ;  000 reset state
                        PRIMLD== bit00                  ;  001 primary load state
                        READY== bit01                   ;  010 ready state
                        RUN ==  bit01!bit00             ;  011 running state

                        UNIHLT == bit02!bit00           ;  101 unibus halted state
                        NIHLT  == bit02!bit01           ;  110 ni halted state
                        NIUNI  == bit02!bit01!bit00     ;  111 ni and unibus halted state


                    ;Port Control and Status Register 2

                        ; lower 16 address bits of the port control block base
                        ; address pointer  in bit 15 to bit 0

                    ;Port Control and Status Register 3

                        ; upper 2 address bits of the port control block base
                        ; address pointer  in bit 1 to bit 0

                    ;Port Functions

                        ; function codes are as follows

                           PFNOP == 0                   ; no operation performed
                           RDDEFA == bit01              ; read default physical address

                           RDPHYA == bit02              ; read physical address
                           WDPHYA == bit02!bit00        ; write physical address

                           RDMULA == bit02!bit01        ; read list of multicast addresses
```

```
720         000007                          WDMULA == bit02!bit01!bit00  ; write list of multicast addresses
721
722         000010                          RDRNGS  == bit03             ; read both the rcvr and xmit rings
723         000011                          WDRNGS  == bit03!bit00       ; write both the rcvr and xmit rings
724
725         000012                          RDCNTS  == bit03!bit01       ; read counters
726         000013                          CLRCNTS == bit03!bit01!bit00 ; read and clear counters
727
728         000014                          RDMODE == bit03!bit02        ; read internal link mode register
729         000015                          WDMODE == bit03!bit02!bit00  ; write internal link mode register
730
731         000016                          RDSTA  == bit03!bit02!bit01  ; read port status
732         000017                          CLRSTA == bit03!bit02!bit01!bit00
733                                                                      ; read and clear port status
734
735         000020                          DMPMEM == bit04              ; dump internal memory
736         000021                          LDMEM  == bit04!bit00        ; load internal memory
737
738         000022                          RDSYS  == bit04!bit01        ; read system id parameters
739         000023                          WDSYS  == bit04!bit01!bit00  ; write system id parameters
740
741                                 ;
742                                 ;       Ethernet frame  offsets
743                                 ;
744
745
746         000016                  header  ==      14.             ; offset (size) to end of  header in bytes
747
748         000000                  destin  ==      0               ; destination address
749         000006                  source  ==      6               ; source address
750         000014                  protoT  ==      12.             ; protocol type field
751
752                                 ;               ------------------------
753                                 ;               !  destination address  !
754                                 ;               ------------ -----------
755                                 ;               !       (6 bytes)       !
756                                 ;               ------------------------
757                                 ;               !                       !
758                                 ;               ------------------------
759                                 ;       +6      !    source  address    !
760                                 ;               ------------------------
761                                 ;               !       (6 bytes)       !
762                                 ;               ------------------------
763                                 ;               !                       !
764                                 ;               ------------------------
765                                 ;       +12.    !    protocall type     !
766                                 ;               ------------------------
767                                 ;       +14.    !         data          !
768                                 ;               ------------------------
769                                 ;               !       more data       !
770                                 ;
771
772                                 ;+
773                                 ;       Xmit ring descriptor definitions
774                                 ;-
775
776                                 ; TDRB+0
```

```
                                ;
                                ;       nothing needed
                                ;
                                ; TDRB+2
                                ;
                                ;       nothing needed
                                ;
                                ;
                                ; TDRB+4
                                ;
                                ;
000400                                  enp     ==      bit08           ; end of frame flag
001000                                  stp     ==      bit09           ; stop of frame flag
002000                                  def     ==      bit10           ; defferring frame flag
004000                                  one     ==      bit11           ; xmit successful after one retry
010000                                  more    ==      bit12           ; xmit successful after more than
                                                                        ;       one retry
040000                                  errs    ==      bit14           ; ERROR SUMMARY BIT
100000                                  own     ==      bit15           ; ownership bit (=1 DEUNA/DELUA, =0 host)
                                ;
                                ; TDRB+6
                                ;
002000                                  rtry    ==      bit10           ; retry error bit
004000                                  lcar    ==      bit11           ; lost carrier error bit
010000                                  lcol    ==      bit12           ; late collision error bit
                                ;
040000                                  ubto    ==      bit14           ; unibus timeout error bit
100000                                  bufl    ==      bit15           ; buffer length error bit
                                ;
                                ;+
                                ;       Rcvr ring descriptor defintions
                                ;-
                                ;
                                ; RDRB+0
                                ;
                                ;       nothing needed
                                ;
                                ; RDRB+2
                                ;
                                ;       nothing needed
                                ;
                                ;
                                ; RDRB+4
                                ;
                                ;
                                        ; --> indicates same as for transmit ring descriptor base
                                ;
004000                                  crc     ==      bit11           ; crc error in received frame
010000                                  oflo    ==      bit12           ; message overflow
020000                                  fram    ==      bit13           ; framing error
                                ;
                                        ;errs   ==      bit14           ; ERROR SUMMARY BIT
                                        ;own    ==      bit15           ; ownership bit (=1 DEUNA/DELUA, =0 host)
                                ;
                                ; RDRB+6
                                ;
020000                                  nchn    ==      bit13           ; set to indicate DEUNA/DELUA in no
```

```
                                                                ; buffer chain on rcvr mode

                        ;ubto   ==      bit14                   ; unibus timeout error bit
                        ;bufl   ==      bit15                   ; buffer length error bit

002756                  xpklen  == 1518.                        ; transmit frame length
002756                  rpklen  == 1518.                        ; recieve frame length
000004                  no.ntr  == 4                            ; number of entries in xmit rings
000010                  no.nrr  == 8.                           ; number of entries in receive rings
000016                  LBCOU   == 16                           ; offset to byte count for this frame type
000020                  LISCOU  == 20                           ; offset to count for listen log entry
000022                  LISENT  == 22                           ; length of one entry in listen log
000006                  ADRCOU  == 6                            ; offset to count for address list entry
000010                  ADRENT  == 10                           ; length of one entry in address list


                        ;
                        ; System ID reply message offsets
                        ;
000022                          sircpt == 22
000024                          siffid == 24
000016                          siccou == 16
                        ;
                        ; Device type defs
                        ;
000001                          IDTUNA == 1             ; DEUNA
000003                          IDTCNA == 3             ; DECNA
000005                          IDTQNA == 5             ; DEQNA
000011                          IDTLUA == 11            ; DELUA
000013                          IDTCSA == 13            ; DECSA - PLUTO
000021                          IDTSRV == 21            ; DSRVA - POSEIDON
                        ;
                        ; Loop Direct Offsets
                        ;
000016                          ldskip == 16            ; offset to skip count
000020                          ldfct1 == 20            ; offset to forward function code
000022                          ldadr1 == 22            ; offset to forward address
000030                          ldfct2 == 30            ; offset to reply function code
000032                          ldadr2 == 32            ; offset to reply address
000022                          ldata  == 22            ; number of bytes of data buffer occupied by
                                                        ; loop header

                        ;
                        ; Full Assist Offsets
                        ;
000016                          faskip == 16            ; offset to skip count
000020                          fafct1 == 20            ; offset to first forward function code
000022                          faadr1 == 22            ; offset to first forward address
000030                          fafct2 == 30            ; offset to second forward function code
000032                          faadr2 == 32            ; offset to second forward address
000040                          fafct3 == 40            ; offset to third forward function code
000042                          faadr3 == 42            ; offset to third forward address
000050                          fafct4 == 50            ; offset to reply function code
000052                          faadr4 == 52            ; offset to reply address
000032                          fdata1 == 32            ; length of loopback header
000042                          fdata2 == 42            ; length of loopback header for full assist
                        ;
```

```
                                ; Counter Offsets
                                ;
000002                                  c.seca == 2
000004                                  c.prec == 4
000010                                  c.mrec == 10
000014                                  c.rerb == 14
000016                                  c.rerr == 16
000020                                  c.rdat == 20
000024                                  c.rmdb == 24
000030                                  c.rlin == 30
000032                                  c.rlex == 32
000034                                  c.pxmt == 34
000040                                  c.mxmt == 40
000044                                  c.pxm3 == 44
000050                                  c.pxm2 == 50
000054                                  c.pxmd == 54
000060                                  c.xdat == 60
000064                                  c.xmdb == 64
000066                                  c.xabb == 66
000070                                  c.xabt == 70
000074                                  c.coll == 74

                                ;--*
                                ;       The following equates are for use with the memory management hardware
                                ;       and its associated routines
                                ;--*
172350                                  KPAR4   ==      172350          ; address of KPAR4
172352                                  KPAR5   ==      172352          ; address of KPAR5
172354                                  KPAR6   ==      172354          ; address of KPAR6

001000                                  NKPAR4  ==      001000          ; original value for KPAR4
001200                                  NKPAR5  ==      001200          ; original value for KPAR5
002400                                  TKPAR6  ==      002400          ; value for KPAR6 to do write rings
                                                                        ; function only

177572                                  MMCSR0  ==      177572          ; address of MMU CSR0
000001                                  MMUENA  ==      000001          ; mask to enable MMU
000000                                  MMUDIS  ==      000000          ; mask to disable MMU

                                ;--*
                                ;       The following values will be used as new values for KPAR4 and KPAR5
                                ;       registers, which, will then point to the page that contains the
                                ;       indicated structures
                                ;--*
002000                                  ORRING  ==        2000          ; offset to receive ring
002400                                  OTRING  ==        2400          ; offset to transmit ring
002600                                  ONTAB   ==        2600          ; offset to node table
003000                                  OSTAB   ==        3000          ; offset to summary table
003400                                  OLLOG   ==        3400          ; offset to listen log

000000                                  BA      ==           0          ; base address for call to BUFREQ
000001                                  EA      ==           1          ; extended bits(18:16) for call to BUFREQ

                                ;--*
                                ;       The following equates are virtual addresses of data structures that
                                ;       are mapped into extended memory.  Since KPAR4 and KPAR5 are the only
                                ;       two page address registers that are being used to remap to extended
```

```
                        ;       memory, the virtual addresses of the data structures will be in the
                        ;       range 100000(0) - 137776(0).
                        ;--*
        100000                  NODTBL  ==      100000          ; address of node table
        110000                  NODEND  ==      110000          ; address of end of node table
        110000                  DEFTBL  ==      110000          ; address of default address table
        120000                  DEFEND  ==      120000          ; address of end of default table
        010000                  DEFNOD  ==      010000          ; distance between node and default addr.
        100000                  STATBL  ==      100000          ; address of summary table
        126000                  STAEND  ==      126000          ; address of end of summary table
        100000                  LISLOG  ==      100000          ; address of listen log
        101034                  LISEND  ==      101034          ; address of end of listen log
        101034                  ADRLIS  ==      101034          ; address of listen address list
        101414                  ADREND  ==      101414          ; address of end of listen address list
        100000                  RRING   ==      100000          ; address of receive ring
        100000                  XRING   ==      100000          ; address of transmit ring

                        ;--*
                        ;       The next equates are the actual 18-bit physical addresses of the
                        ;       the first transmit and receive buffers, respectively
                        ;--*
        040050                  X11501  ==      040050          ; address bits <17:01> ...
        000001                  X11716  ==      000001          ; ... of first transmit buffer
        000120                  R11501  ==      000120          ; address bits <17:01> ...
        000001                  R11716  ==      000001          ; ... of first receive buffer

                        ;--*
                        ;       And now the virtual addresses of the first transmit and receive
                        ;       buffers, respectively.
                        ;--*

        100050                  XBUFV1  ==      100050          ; virtual addr. of first transmit buffer
        100120                  RBUFV1  ==      100120          ; virtual addr. of first receive buffer


                        .SBTTL  GLOBAL DATA SECTION

                        ;**
                        ; THE GLOBAL DATA SECTION CONTAINS DATA THAT ARE USED
                        ; IN MORE THAN ONE TEST.
                        ;--
                        ;COMMAND LINE BUFFER, DATA LOCATIONS AND MESSAGES FOR ACTION ROUTINES

000214                  STACKS: .BLKW   100.            ; PARAMETER STACK -- USED TO PASS PROCEDURE ARGS
000524  000000          DEVICE: .WORD   0               ;DEFAULT TO DEUNA
000526                  FILLIN: .BLKB   132.            ;BUFFER FOR SINGLE LINE READ FROM FILE
000732                  CMDBUF: .BLKB   72.             ;BUFFER FOR OPERATOR COMMANDS
001042                  CBOBUF: .BLKB   17.             ;BUFFER TO HOLD INPUT ASCII ADDRESS/PROTOCOL TYPE STRING
                                .EVEN
001064  000000          KEYWD1: .WORD   0               ;
001066  000000          KEYWD2: .WORD   0
001070  000000          ADRBUF: .WORD   0               ;BUFFER FOR NODE ADDRESS
001072  000000                  .WORD   0
001074  000000                  .WORD   0
001076                  SOUFIL::
001076  000000                  .WORD   0               ;BUFFER FOR SOURCE FILTER FOR LISTEN COMMAND
001100  000000                  .WORD   0
```

```
1005 001102  000000                    .WORD   0
1006 001104                    DESFIL::
1007 001104  000000                    .WORD   0               ;BUFFER FOR DESTINATION FILTER FOR LISTEN COMMAND
1008 001106  000000                    .WORD   0
1009 001110  000000                    .WORD   0
1010 001112                    PROFIL::
1011 001112  000000                    .WORD   0               ;BUFFER FOR PROTOCOL FILTER FOR LISTEN COMMAND
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$MERR: .BYTE   0               ;RETURN -1 IF ERROR IN OPERATOR SELECTED
1063                                                            ;MESSAGE INPUT OCCURED, 0 FOR GOOD INPUT
1064 001305     000             P$TEXT: .BYTE   0               ; indicates text, not address to TRVADR routine
1065 001306     000             P$BONC: .BYTE   0               ; indicate we are processing bounce command
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
1.12
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
1122 001450                         MSGAD::
1123 001450  001466'                        .WORD   MSG00
1124 001452  001616'                        .WORD   MSG01
1125 001454  001617'                        .WORD   MSG02
1126 001456  001620'                        .WORD   MSG03
1127 001460  001621'                        .WORD   MSG04
1128 001462  001622'                        .WORD   MSG05
1129 001464  001722'                        .WORD   OPSLBF
1130
1131 001466    040     041     042  MSG00:: .ascii  \ !"#$%&'()*+,-/0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ\
     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  \[]t-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                          EMSG5::
 1150 001722                          OPSLBF: .blkb   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)
 1169                                          .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 recieve ring
 1191 002072  100000                  xrgcur::.word   XRING           ; current entry in transmit ring
 1192 002074  100000                  rrgcur::.word   RRING           ; current entry in recieve ring
 1193 002076  100000                  xrgnxt::.word   XRING           ; next entry in transmit ring
 1194 002100  100000                  rrgnxt::.word   RRING           ; next entry in recieve 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
```

```
                                ;
                                ;INFORMATION ABOUT THE CURRENT UNIT AS OBTAINED FROM THE HARDWARE P-TABLE
                                ;
                                ;*****************************************************************************************
                                                                ;PCSRs of current slot
002106  000000  PCSR0:: .WORD           ;  address of PCSR0     (port command field
002110  000000  PCSR1:: .WORD           ;               1       (state & self test fields
002112  000000  PCSR2:: .WORD           ;               2       (pcb address lo 15 bits
002114  000000  PCSR3:: .WORD           ;               3       (pcb address hi 2 bits

002116  000000  PCSR0C::.WORD   0       ;PCSR0 CONTENTS
002120  000000  PCSR1C::.WORD   0       ;PCSR1 CONTENTS
002122  000000  PCSR2C::.WORD   0       ;PCSR2 CONTENTS
002124  000000  PCSR3C::.WORD   0       ;PCSR3 CONTENTS


002126  000000  UNACSR::.WORD   0       ;CSR
002130  000000  UNAVEC::.WORD   0       ;VECTOR
002132  000000  UNAPRI::.WORD   0       ;PRIORITY

002134  000000  FRESIZ::.WORD   0       ;POINTER TO WORD CONTAINING SIZE OF FREE MEMORY
002136  000000  FREMEM::.WORD   0       ;POINTER TO FREE MEMORY SPACE

002140  000000  UNIT::  .WORD   0       ;CURRENT UNIT NUMBER BEING TESTED

                                ;
                                ; broadcast address - FF-FF-FF-FF-FF-FF
                                ;
002142  177777          brdadr:         .word   -1
002144  177777                          .word   -1
002146  177777                          .word   -1

                                ;       Port control block function structures

                                                                ;port control block
002150  000000  PCBB0:: .word   0       ;       port function
002152  000000  PCBB2:: .word   0       ;       port function dependent parameters
002154  000000  PCBB4:: .word   0       ;       port function dependent parameters
002156  000000  PCBB6:: .word   0       ;       port function dependent parameters

                                ;               function table

002160  002230' FUNTAB::          .word   $PNOP           ; no op
002162  000000                   .word   0               ; fill in the hole
002164  002232'                  .word   $RDDE           ; read default physical address
002166  000000                   .word   0               ; fill in another hole
002170  002242'                  .word   $RDPH           ; read physical address
002172  002252'                  .word   $WDPH           ; write physical address
002174  002262'                  .word   $RDMC           ; read multicast address list
002176  002322'                  .word   $WDMC           ; write multicast address list
002200  002362'                  .word   $RDRN           ; read descriptor rings
002202  002406'                  .word   $WDRN           ; write descriptor rings
002204  002432'                  .word   $RDCN           ; read counters
002206  002546'                  .word   $CLRC           ; read and clear counters
002210  002556'                  .word   $RDMO           ; read mode
002212  002566'                  .word   $WDMO           ; write mode
```

```
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
1268 002230  000000                 $pnop::         .word 0         ; no-op
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                                 .word   0       ; pcbb+4
1278 002240  000000                                 .word   0       ; pcbb+6
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                                 .word   0       ; pcbb+4
1288 002250  000000                                 .word   0       ; pcbb+6
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                                 .word   0       ; pcbb+2
1296 002256  000000                                 .word   0       ; pcbb+4
1297 002260  000000                                 .word   0       ; pcbb+6
1298
1299                                ;+
1300                                ;       RDMULA == bit02!bit01   ; read multicast address list
1301                                ;-
1302
1303                                .even
1304 002262  000006                 $RDMC::         .word   6       ; function code
1305 002264  002272'                                .word   ucb6    ; ucbb address
1306 002266  000000                                 .word   0       ; pcbb+4
1307 002270  000000                                 .word   0       ; pcbb+6
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                                   .even
 1316 002322  000007                    $WDMC::         .word   7       ; function code
 1317 002324  002332'                                   .word   ucb7    ; ucbb address
 1318 002326  000400                                    .word   400     ; length of list = 1
 1319 002330  000000                                    .word   0       ; pcbb+6
 1320
 1321 002332  000253                    ucb7::          .word   253     ; multicast address for loopback
 1322 002334  001000                                    .word   1000
 1323 002336  000000                                     word   0
 1324 002340                                            blkw    9.      ; room for three more addresses
 1325
 1326                                   ;+
 1327                                   ;          RDRNGS == bit03              ; read both the rcvr and xmit rings
 1328                                   ;
 1329
 1330                                   .even
 1331 002362  000010                    $RDRN::         .WORD   10      ; FUNCTION CODE
 1332 002364  002372'                                   .word UCB10     ; ucbb address
 1333 002366  000000                                    .word   0       ; null
 1334 002370  000000                                    .word   0       ; null
 1335
 1336                                   .even
 1337
 1338 002372  140000                    ucb10::         .word XRING+40000       ; ucbb
 1339 002374  002000                                    .word 2000      ; ucbb+2
 1340 002376  000000                                    .word   0       ; ucbb+4
 1341 002400  100000                                    .word RRING     ; ucbb+6
 1342 002402  002000                                    .word 2000      ; ucbb+10
 1343 002404  000000                                    .word   0       ; ucbb+12
 1344
 1345
 1346                                   ;+
 1347                                   ;          WDRNGS  == bit03!bit00       ; write both the rcvr and xmit rings
 1348                                   ;-
 1349
 1350                                   .even
 1351
 1352 002406  000011                    $WDRN::         .WORD   11      ; FUNCTION CODE
 1353 002410  002416'                                   .word  UCB11    ; ucbb address
 1354 002412  000000                                    .word   0       ; null
 1355 002414  000000                                    .word   0       ; null
 1356
 1357                                   .even
 1358
 1359 002416                            ucb11::
 1360 002416  040000                                    .word   40000   ; transmit ring base address
 1361 002420     001                                    .byte   1       ; hi bits of transmit ring base address
 1362 002421     005                                    .byte   5       ; five words per ring entry (1 for port driver)
 1363 002422  000004                                    .word   NO.NTR  ; four transmit descriptors in the ring
 1364
 1365 002424  000000                                    .word   0       ; receive ring base address
 1366 002426     001                                    .byte   1       ; hi bits of receive ring base address
 1367 002427     005                                    .byte   5       ; five words per ring entry (1 for port driver)
 1368 002430  000010                                    .word   NO.NRR  ; eight receive descriptors in the ring
 1369
 1370
```

```
1371
1372                                    ;+
1373                                    ;       RDCNTS  == bit03!bit01      ; read counters
1374                                    ;
1375
1376                                    .even
1377 002432  000012                            $RDCN::         .WORD   12      ; FUNCTION
1378 002434  002442'                                           .word   UCB12   ; ucbb address
1379
1380                                                                            ; DEFAULT COUNT OF COUNTER LIST
1381                                                                                    ;       40 (octal)
1382 002436  000000                                            .word   0       ; null
1383
1384 002440  000110                                            .word   110     ;  CTRLEN
1385                                                                            ;
1386
1387                                    .even
1388
1389 002442                                    ucb13::
1390 002442  000000                            ucb12::         .word   0       ; ucbb
1391 002444  000000                                            .word   0       ; ucbb+2
1392 002446  000000                                            .word   0       ; ucbb+4
1393 002450  000000                                            .word   0       ; ucbb+6
1394 002452  000000                                            .word   0       ; ucbb+10
1395 002454  000000                                            .word   0       ; ucbb+12
1396 002456  000000                                            .word   0       ; ucbb+14
1397 002460  000000                                            .word   0       ; ucbb+16
1398 002462  000000                                            .word   0       ; ucbb+20
1399 002464  000000                                            .word   0       ; ucbb+22
1400 002466  000000                                            .word   0       ; ucbb+24
1401 002470  000000                                            .word   0       ; ucbb+26
1402 002472  000000                                            .word   0       ; ucbb+30
1403 002474  000000                                            .word   0       ; ucbb+32
1404 002476  000000                                            .word   0       ; ucbb+34
1405 002500  000000                                            .word   0       ; ucbb+36
1406 002502  000000                                            .word   0       ; ucbb+40
1407 002504  000000                                            .word   0       ; ucbb+42
1408 002506  000000                                            .word   0       ; ucbb+44
1409 002510  000000                                            .word   0       ; ucbb+46
1410 002512  000000                                            .word   0       ; ucbb+50
1411 002514  000000                                            .word   0       ; ucbb+52
1412 002516  000000                                            .word   0       ; ucbb+54
1413 002520  000000                                            .word   0       ; ucbb+56
1414 002522  000000                                            .word   0       ; ucbb+60
1415 002524  000000                                            .word   0       ; ucbb+62
1416 002526  000000                                            .word   0       ; ucbb+64
1417 002530  000000                                            .word   0       ; ucbb+66
1418 002532  000000                                            .word   0       ; ucbb+70
1419 002534  000000                                            .word   0       ; ucbb+72
1420 002536  000000                                            .word   0       ; ucbb+74
1421 002540  000000                                            .word   0       ; ucbb+76
1422 002542  000000                                            .word   0       ; ucbb+100
1423 002544  000000                                            .word   0       ; ucbb+102
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                                                                            ; DEFAULT COUNT OF COUNTER LIST
1434 002552  000000                                             .word   0       ; null
1435 002554  000040                                             .word   40      ;  (# OF WORDS IN LIST * UPPER BYTE)
1436                                                                            ; MAX NUMBER VALUE = 32 (decimal) =
1437                                                                            ;       40 (octal)
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                                                                            ; bits to read the una internal
1451                                                                            ;       mode register
1452 002562  000000                                             .word   0       ; null
1453 002564  000000                                             .word   0       ; null
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                                                                            ; bits to write the una internal
1463                                                                            ;       mode register
1464 002572  000000                                             .word   0       ; null
1465 002574  000000                                             .word   0       ; null
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                                                                            ;       maximum supported by UNA
1477                                                                            ; upper byte = # of multicast adrs
1478                                                                            ;       currently supported by UNA
1479 002604  000000                                             .word   0       ; word = maximum # of words in
1480                                                                            ;       ucb for counters
1481                                                                            ;       as currently perceived
1482                                                                            ;       by the UNA
1483
1484                                    ;+
```

```
                                ;          CLRSTA  == bit03!bit02!bit01!bit0
                                ;-                                      ; read and clear write port status
                                .even
002606  000017                  $clrs::         .word   17              ; function code
002610  000000                                  .word   0               ; a list of ERRORS and STATUS
002612  000000                                  .word   0               ; lower byte = # of multicast adrs
                                                                        ;       maximum supported by UNA
                                                                        ; upper byte = # of multicast adrs
                                                                        ;       currently supported by DEUNA/DELUA
002614  000000                                  .word   0               ; word = maximum # of words in
                                                                        ;       ucb for counters
                                                                        ;       as currently perceived
                                                                        ;       by the DEUNA/DELUA
                                ;+
                                ;          DMPMEM == bit04              ; dump internal memory
                                ;-
                                .even
002616  000020                  $dmem::         .word   20              ; function code
002620  002626'                                 .word   ucb20           ; ucbb address
002622  000000                                  .word   0               ; MBZ
002624  000000                                  .word   0               ; MBZ

002626                          ucb20::
002626  000000                  ucb21::         .word   0               ; function length (no of words to xfer)
002630  000000                                  .word   0               ; hdbb - host memory data block address
002632  000000                                  .word   0               ; internal DEUNA address ...
002634  021040                                  .word   21040           ; ... changed if DELUA
002636  000000                                  .word   0               ; extra word for IDBB<23:0> -- if DELUA

                                ;+
                                ;          LDMEM  == bit04!bit00        ; load DEUNA/DELUA internal memory
                                ;-
                                .even
002640  000021                  $lmem::         .word   21              ; function code
002642  002626'                                 .word   ucb21           ; ucbb address
002644  000000                                  .word   0
002646  000000                                  .word   0

                                ;+
                                ;          RDSYS  == bit04!bit01        ; read system id
                                ;-
                                .even
002650  000022                  $rdsy::         .word   22              ; function code
002652  002670'                                 .word   ucb22           ; ucbb address
002654  000000                                  .word   0
002656  000033                                  .word   27.             ; length of id message
                                ;+
                                ;          WTSYS  == bit04!bit01!bit00  ; write system id
                                ;-

                                .even
002660  000023                  $wtsy::         .word   23              ; function code
```

```
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:
1547 002670  000000                  ucb23:           .word   0       ;udbb+0
1548 002672  000000                                   .word   0       ;udbb+2
1549 002674  000000                                   .word   0       ;udbb+4
1550 002676  000000                                   .word   0       ;udbb+6
1551 002700  000000                                   .word   0       ;udbb+10
1552 002702  000000                                   .word   0       ;udbb+12
1553 002704  000000                                   .word   0       ;udbb+14
1554 002706  000000                                   .word   0       ;udbb+16
1555 002710  000000                                   .word   0       ;udbb+20
1556 002712  000000                                   .word   0       ;udbb+22
1557 002714  000000                                   .word   0       ;udbb+24
1558 002716  000000                                   .word   0       ;udbb+26
1559 002720  000000                                   .word   0       ;udbb+30
1560 002722  000000                                   .word   0       ;udbb+32
1561 002724  000000                                   .word   0       ;udbb+34
1562 002726  000000                                   .word   0       ;udbb+36
1563 002730  000000                                   .word   0       ;udbb+40
1564 002732  000000                                   .word   0       ;udbb+42
1565 002734  000000                                   .word   0       ;udbb+44
1566 002736  000000                                   .word   0       ;udbb+46
1567 002740  000000                                   .word   0       ;udbb+50
1568 002742  000000                                   .word   0       ;udbb+52
1569 002744  000000                                   .word   0       ;udbb+54
1570 002746  000000                                   .word   0       ;udbb+56
1571 002750  000000                                   .word   0       ;udbb+60
1572 002752  000000                                   .word   0       ;udbb+62
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              s.rec:: .word   0                    ; messages received
1585 002770  000000              s.nrec::.word   0                    ; messages not received
1586 002772  000000              s.len:: .word   0                    ; length errors
1587 002774  000000              s.comp::.word   0                    ; compare errors
1588 002776  000000              s.byte::.word   0                    ; bytes compared
1589 003000  000000              s.xfer::.word   0                    ; bytes transfered
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              nircnt::.word   0                    ; DEUNA/DELUA recieve 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                    ; recieve buffers lost counter
1601 003016  000000                bcount::.word   0                    ; unexplained interrupts counter
1602 003020  000000                errflg::.word   0                    ; error flag
1603 003022  000000                timout::.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               ; protocall type for loopback messages
1609 003036  001140                prot02::.word   001140               ; protocall 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  000000                xfer::  .word   0                    ; stores 'bytes transfered'
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                  ; reciept 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
```

```
                                        ;
                                        ; Recieve assist message
                                        ;

003334                                  RASIST::
003334  000000                                  .word   0                       ; skip count
003336  000002                                  .word   2                       ; function = forward data
003340  000000  000000  000000                  .word   0,0,0                   ; transmit assist address
003346  000002                                  .word   2                       ; function = forward data
003350  000000  000000  000000                  .word   0,0,0                   ; local node address
003356  000001                                  .word   1                       ; function = reply
003360  000000  000000  000000                  .word   0,0,0                   ; local node address

                                        ;
                                        ; Full assist message
                                        ;

003366                                  FASIST::
003366  000000                                  .word   0                       ; skip count
003370  000002                                  .word   2                       ; function = forward data
003372  000000  000000  000000                  .word   0,0,0                   ; target node address
003400  000002                                  .word   2                       ; function = forward data
003402  000000  000000  000000                  .word   0,0,0                   ; assist node address
003410  000002                                  .word   2                       ; function = forward data
003412  000000  000000  000000                  .word   0,0,0                   ; local node address
003420  000001                                  .word   1                       ; function = reply
003422  000000  000000  000000                  .word   0,0,0                   ; local node address


                                        .SBTTL  COMMAND LINE ACTION TREE

                                        ;SAMPLE CLI TREE NODE  (ALWAYS AT LEAST 1 WORD)

                                        ; ----------------------
                                        ; ! ACTION  ! CHAR CODE !
                                        ; ----------------------
                                        ; !  MISS DISPLACEMENT  !       ONLY IF "MISS" ARGUMENT DEFINED
                                        ; ----------------------
                                        ; !  NEXT MODE DISPLMNT !       ONLY IF "ASCII" ARGUMENT DEFINED
                                        ; ----------------------
                                        ; !  ASCIZ MATCH STRING !       ONLY IF "ASCII" ARGUMENT DEFINED
                                        ; !       (.EVEN)      !
                                        ; ----------------------
                                        .NLIST  ME
003430                                  CLITRE:

                                        ;FIRST KEYWORD
003430                                          CLI     CLISPA,0,N10$                   ;SKIP ANY LEADING SPACES
003434                                  N10$:   CLI     <'?>,HELP,N12$                  ;IS THE FIRST NON-SP CHAR. A "?"
003440                                          CLI     CLISPA,0,N11$                   ; skip spaces
003444                                  N11$:   CLI     CLISPA,0,N50$                   ; error if non-space characters left
003450                                  N12$:   CLI     CLISTR,HELP,N14$,<'HELP'>       ;ELSE IS FIRST WORD A "HELP"
003464                                          CLI     CLISPA,0,N13$                   ; skip spaces after executing
003470                                  N13$:   CLI     CLISPA,0,N50$                   ; error if nonspace chars left
003474                                  N14$:   CLI     CLISTR,NOTNUF,N16$,<'NODE'>     ;ELSE IS FIRST WORD A "NODE"
003510                                          CLI     CLIBR,0,N80$                    ;  IF YES, BR N80$
```

COMMAND LINE ACTION TREE

```
 1713 003514                    N16$:   CLI     <'B>,NOTNUF,N18$                ; is char a b?
 1714 003520                            CLI     CLISTR,BUILD,N17$,<'UILD'>      ;ELSE IS FIRST WORD A "BUILD"
 1715 003534                            CLI     CLIBR,0,N70$                    ;  IF YES, SEE BR N70$
 1716 003540                    N17$:   CLI     CLISTR,0,N50$,<'OUNCE'>         ; IS IT BOUNCE COMMAND?
 1717 003554                            CLI     CLIBR,0,N300$                   ; branch if it is
 1718 003560                    N18$:   CLI     CLISTR,NOTNUF,N20$,<'RUN'>      ;ELSE IS FIRST WORD A "RUN"
 1719 003572                            CLI     CLIBR,0,N180$                   ;  IF YES, BR N180$
 1720 003576                    N20$:   CLI     <'S>,NOTNUF,N25$                ;ELSE IS FIRST CHAR. A "S"
 1721 003602                            CLI     CLISTR,0,N22$,<'HOW'>           ;  IF YES IS REST OF WORD "HOW"
 1722 003614                            CLI     CLIBR,0,N100$                   ;    IF YES,BR N100$
 1723 003620                    N22$:   CLI     CLISTR,SUMMRY,N23$,<'UMMARY'>   ;  ELSE IS REST OF WORD "UMMARY"
 1724 003636                            CLI     CLIEXI,0                        ;    IF YES, DO "SUMM" AND EXIT
 1725 003640                    N23$:   CLI     CLISTR,0,N24$,<'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,0                        ;  ELSE "ILL COMMAND"
 1729 003662                            CLI     CLIEXI,0                        ;  EXIT
 1730 003664                    N25$:   CLI     CLISTR,NOTNUF,N26$,<'CLEAR'>    ;ELSE IS FIRST WORD A "CLEAR"
 1731 003700                            CLI     CLIBR,0,N120$                   ;  IF YES, BR N120$
 1732 003704                    N26$:   CLI     CLISTR,NOTNUF,N28$,<'IDENTIFY'> ;ELSE IS FIRST WORD "IDENTIFY"
 1733 003724                            CLI     CLIBR,0,N140$                   ;  IF YES, GET ADDRS, BR N140$
 1734 003730                    N28$:   CLI     CLISTR,NOTNUF,N29$,<'MESSAGE'>  ;ELSE IS FIRST WORD  MESSAGE"
 1735 003746                            CLI     CLIBR,0,N160$                   ;  IF YES, BR N160$
 1736 003752                    N29$:   CLI     CLISTR,0,N30$,<'UNSAVE'>        ;ELSE IS FIRST WORD "UNSAVE"
 1737 003770                            CLI     CLIBR,0,N210$                   ;  IF YES, BR TO N210$
 1738 003774                    N30$:   CLI     CLISTR,EXIT,N31$,<'EXIT'>       ;ELSE IS FIRST WORD "EXIT"
 1739 004010                            CLI     CLIEXI,0                        ;  IF YES EXIT
 1740 004012                    N31$:   CLI     CLISTR,NOTNUF,N32$,<'FUNCTION'> ;ELSE IS FIRST WORD "FUNCTION"
 1741 004032                            CLI     CLIBR,0,N200$                   ;  IF YES, BR N200$
 1742 004036                    N32$:   CLI     CLISTR,LISTEN,N50$,<'LISTEN'>   ;ELSE IS FIRST WORD "LISTEN"
 1743 004054                            CLI     CLIBR,0,N145$                   ;  IF YES, BR N145$
 1744 004060                    N50$:   CLI     CLIERR,0                        ;OTHERWISE "ILL CMD",
 1745 004062                            CLI     CLIEXI,0                        ;  EXIT
 1746
 1747                           ;SECOND KEYWORD FOR BUILD COMMAND
 1748
 1749 004064                    N70$:   CLI     CLISPA,0,N72$                   ; SKIP LEADING SPACES
 1750 004070                    N72$:   CLI     <'/>,NULL,N50$                  ; ERR IF ILLEGAL QUALIFIER
 1751 004074                            CLI     CLISPA,0,N74$                   ; skip spaces
 1752 004100                    N74$:   CLI     CLISTR,SETQIK,N50$,<'QUICK'>    ; SET QUCK BUILD FLAG IF QUICK
 1753 004114                            CLI     CLISPA,0,N76$                   ; skip spaces
 1754 004120                    N76$:   CLI     CLISPA,0,N50$                   ; error if more to command
 1755 004124                    N78$:   CLI     CLIEXI,0                        ; EXIT
 1756
 1757                           ;SECOND KEYWORD (ADR/TYPE) FOR NODE COMMAND
 1758
 1759 004126                    N80$:   CLI     CLISPA,0,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$:   CLI     CLIBIF,0,N50$                   ;TAKE ERROR BRANCH IF ERROR EXISTS
 1764 004146                    N95$:   CLI     CLIEXI,0                        ;EXIT
 1765
 1766                           ;SECOND KEYWORD FOR SHOW COMMAND
 1767
 1768 004150                    N100$:  CLI     CLISPA,0,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 ADDRESS FROM NODE TABLE
 1834 004646                    N148$:  CLI     CLIBR,SOUADR,N145$              ;    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,SOUADR,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
 1855                           ;REMAINING COMMAND LINE FOR MESSAGE COMMAND
 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$:  CLI     CLISTR,CZEROS,N164$,<'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$:  CLI     CLISTR,CCCITT,N167$,<'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"
```

```
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"
[redacted]                           CLI     CLISPA,0,N1863$                 ; skip spaces
                           N1863$:   CLI     <'=>,0,N50$                     ;PARSE THROUGH "="
1916 005604                          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$:    CLI     CLIOCT,CFUNCT,N50$              ; 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$:    CLI     CLIOCT,0,N50$                   ; 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  \%N%ACOMMAND SUMMARY FOR THE NETWORK INTERCONNECT EXERCISER (NIE)\
 1972 006033   HELP2:  .ASCIZ  \%N%A(it is only necessary to type the letters in brackets)\
 1973 006126   HELP3:  .ASCIZ  \%N2%A[H]elp or ?        - types this help text.\
 1974 006177   HELP4:  .ASCIZ  \%N2%A[E]xit            - return to the supervisor.\
 1975 006250   HELP5:  .ASCIZ  \%N2%A[SH]ow [N]odes     - prints information in current node table.\
 1976 006350   HELP6:  .ASCIZ  \%N2%A[SH]ow [M]essage  - prints the selected message type, size and copies.\
 1977 006463   HELP7:  .ASCIZ  \%N2%A[SH]ow [C]ounters - prints the low level counters of the HOST NODE.\
 1978 006574   HELP8:  .ASCIZ  \%N2%A[R]un [L]ooppair/Pass=nn - runs the looppair test.\
 1979 006664   HELP9:  .ASCIZ  \%N2%A[R]un [A]ll/Pass=nn - runs the node-to-node test.\
 1980 006753   HELP10: .ASCIZ  \%N2%A[R]un [D]irect/Pass=nn - runs the loop direct test.\
 1981 007044   HELP11: .ASCIZ  \%N2%A[R]un [P]attern/Pass=nn - runs the message pattern test.\
 1982 007142   HELP12: .ASCIZ  \%N2%A[M]essage/[T]ype=a/[S]ize=n/[C]opies=m - allows the operator to\
 1983 007247   HELP13: .ASCIZ  \%N%Amodify the default message type, size and copy parameters.\
 1984 007346   HELP14: .ASCIZ  \%N2%A[N]ode adr - enters a physical address into the node\
 1985 007440   HELP15: .ASCIZ  \%N%Atable.\
 1986 007453   HELP16: .ASCIZ  \%N2%A[SU]mmary - prints a summary of the test results.\
 1987 007542   HELP17: .ASCIZ  \%N2%A[B]uild - builds a table of remote node physical addresses by\
 1988 007645   HELP18: .ASCIZ  \%N%Alistening to ID messages on the NI.\
 1989 007715   HELP19: .ASCIZ  \%N2%A[C]lear [N]ode/adr - removes the node specified by either adr\
 1990 010020   HELP20: .ASCIZ  \%N%Aor node logical name from the node table.\
 1991 010076   HELP21: .ASCIZ  \%N2%A[C]lear [N]ode/[A]l - clears the node table.\
 1992 010161   HELP22: .ASCIZ  \%N2%A[C]lear [M]essage - sets all message parameters to default.\
 1993 010262   HELP23: .ASCIZ  \%N2%A[C]lear [S]ummary - clears the table of summary test data.\
 1994 010362   HELP24: .ASCIZ  \%N2%A[I]dentify adr - uses the request ID function to identify NI nodes.\
 1995 010473   HELP25: .ASCIZ  \%N2%A[Sa]ve filename - Saves the contents of the node table to a file\
 1996 010601   HELP26: .ASCIZ  \%N2%A[U]nsave filename - restores node table from a file.\
 1997 010673   HELP27: .ASCIZ  \%N2%A[L]isten [S]ource/adr/[D]estination/adr/[P]rotocol/protocol type\
 1998 011001   HELP28: .ASCIZ  \%N2%A          - listens for frames that pass the specified filters.\
 1999 011105   HELP29: .ASCIZ  \%N%S8%ANotes: 1) adr is the physical address of a node on the NI.\
 2000 011207   HELP30: .ASCIZ  \%N%S8%A        2) Pass count is a decimal number between 1 and 65534. A default\
 2001 011326   HELP31: .ASCIZ  \%N%S8%A           value of 1 is assumed.\
 2002 011376   HELP32: .ASCIZ  \%N%S8%A           Specifying -1 causes the test to be run indefinately.\
 2003 011505   HELP33: .ASCIZ  \%N%S8%A        3) filename is an xxdp file.\
 2004                  .EVEN
 2005
 2006 011560   OPNERR: .ASCIZ  /%N%A?Unable to Open "%T%A"?/
 2007 011614   CLI$PM: .ASCIZ  <12><15>/NIE>/                  ;NIE PROMPT
 2008 011623   CLIERM: .ASCIZ  /%N%A?ILL CMD-BAD SYNTAX?/
 2009 011654   CLINUF: .ASCIZ  /%N%A?INCOMPLETE COMMAND?/
 2010 011705   CLINBG: .ASCIZ  /%N%A?NUMBER TOO BIG?/
 2011 011732   CLIBRX: .ASCIZ  /%N%A?BAD RADIX?/
 2012 011752   LINHLP: .ASCIZ  /%T%N/
 2013 011757   LDRESP: .ASCIZ  /%N%ANODE %T%A HAS RESPONDED./
 2014 012014   RECERR: .ASCIZ  /%N%AFRAME RECEIVED WITH DEUNA,DELUA ERROR./
 2015 012067   RTRYER: .ASCIZ  /%N%ATRANSMISSION ABORTED -- EXCESSIVE COLLISIONS./
 2016 012151   BLDMSG: .ASCIZ  /%N%D2%A Node addresses added, elapsed time: %D2%A minutes./
 2017 012244   BLDDON: .ASCIZ  /%N%A Build completed after %D2%A minutes./
 2018 012316   ILADMS: .ASCII  /%N%ACannot use Broadcast address (FF-FF-FF-FF-FF-FF)/
 2019 012402   ILADM1: .ASCIZ  /%N%Afor loop testing. Address was not added to node table.%N/
```

GLOBAL TEXT SECTION

```
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%O4/
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\
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  /%N2%AListen log is empty!/
2050 013615  ALEMPT: .ASCIZ  /%N2%AAddress list is empty, also./
2051 013657  ALHDR:  .ASCIZ  /%N2%A SOURCE ADDRESS        COUNT%N/
2052 013723  AADDR:  .ASCIZ  /%N%T%S4%D6/
2053 013736  LTMSG:  .ASCIZ  /%N2%ATotal elapsed listen time: %Z2%A:%Z2%A.  Listen commands: %D2/
2054 014041  TABFUL: .ASCIZ  /%N%AThe %T%A table is filled to capacity!/
2055 014113  TABEMT: .ASCIZ  /%N%AThe %T%A table is currently empty!/
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%O4%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  UNSMSG: .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%Ahowever, if no new nodes are added to the table for a 10 minute period/
2073 015511  MSG12:  .ASCIZ  /%N%Athe 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  /%N2%A ROM MICROCODE VERSION (DECIMAL): %D3/
2080 016034  HDMSG3:  .ASCIZ  /%N2%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%A %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/                 ;COMMAND TYPES
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/                ;ARGUMENT TYPES
2133 017470  ARGTY2:  .ASCIZ  /MESSAGES/
```

```
2134 017501  ARGTY3: .ASCIZ  /COUNTERS/
2135 017512  ARGTY4: .ASCIZ  /LOOPPAIR/
2136 017523  ARGTY5: .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  /%N%S5%ACONTENTS OF NODE %T%A INTERNAL COUNTERS:/
2146 017626  cntr01: .asciz  /%N2%ASECONDS SINCE LAST ZEROED:%S15%Z5/
2147 017675  cntr02: .asciz  /%N%AFRAMES RECEIVED:%S20%T/
2148 017730  cntr03: .asciz  /%N%AMULTICAST FRAMES RECEIVED:%S10%T/
2149 017775  cntr04: .asciz  /%N%AFRAMES REC'D WITH ERROR - BITMAP:%S10%B3/
2150 020052  cntr05: .asciz  /%N%AFRAMES RECEIVED WITH ERROR:%S14%Z5/
2151 020121  cntr06: .asciz  /%N%ADATA BYTES RECEIVED:%S16%T/
2152 020160  cntr07: .asciz  /%N%AMULTICAST DATA BYTES RECEIVED:%S6%T/
2153 020230  cntr08: .asciz  /%N%ARECEIVED FRAMES LOST-INTERNAL:%S11%Z5/
2154 020302  cntr09: .asciz  /%N%ARECEIVED FRAMES LOST -LOCAL:%S13%Z5/
2155 020352  cntr10: .asciz  /%N%AFRAMES TRANSMITTED:%S17%T/
2156 020410  cntr11: .asciz  /%N%AMULTICAST FRAMES TRANSMITTED:%S7%T/
2157 020457  cntr12: .asciz  /%N%AFRAMES TRANSMITTED 3+ TRYS:%S9%T/
2158 020524  cntr13: .asciz  /%N%AFRAMES TRANSMITTED 2 TRYS:%S10%T/
2159 020571  cntr14: .asciz  /%N%AFRAMES DEFFERED:%S20%T/
2160 020624  cntr15: .asciz  /%N%ADATA BYTES TRANSMITTED:%S13%T/
2161 020666  cntr16: .asciz  /%N%AMULTICAST BYTES TRANSMITTED:%S8%T/
2162 020734  cntr17: .asciz  /%N%ATRANSMIT FRAMES ABORTED-BITMAP:%S9%B6/
2163 021006  cntr18: .asciz  /%N%ATRANSMIT FRAMES ABORTED:%S17%Z5/
2164 021052  cntr19: .asciz  /%N%AXMIT COLLISION CHECK FAILURE:%S12%Z5/
2165 021123  cntr20: .asciz  /%N%APORT DRIVER ERRORS:%S22%Z5/
2166 021162  cntr21: .asciz  /%N%ABABBLE 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 023162  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  /%N%ATOO MUCH DATA FOR BOUNCE/
2206 023375  EMSG46: .ASCIZ  /%N%ANO 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  /%N%ACURRENT HARDWARE ADDRESS:          %T/
2221 024206  simsg2: .asciz  /%N%AReceipt number:                   %O6/
2222 024261  simsg3: .asciz  /%N%AMaintenance version:              %Z2/
2223 024334  simsg4: .asciz  /%N%AECO:                             %Z2/
2224 024407  simsg5: .asciz  /%N%AUser ECO:                        %Z2/
2225 024462  simsg6: .asciz  /%N%AFunction:                        %O2/
2226 024535  simsg7: .asciz  /%N%ADevice:                          /
2227 024605  simsg8: .asciz  /%N%AConsole User Address:            %T/
2228 024657  simsg9: .asciz  /%N%AReservation Timer:               %O6/
2229 024732  smsg10: .asciz  /%N%AConsole Command Size:           %O6/
2230 025005  smsg11: .asciz  /%N%AConsole Response Size:          %O6/
2231 025060  smsg12: .asciz  /%N%ADEFAULT HARDWARE ADDRESS:       %T/
2232 025132  smsg13: .asciz  /%N%%ASystem Time:                    %O6%O6%O6%O6%O6/
2233
2234
2235         ;--+
2236         ;       Poseidon Specific fields of a system ID message
2237         ;--+
2238 025221  posds:  .asciz  /%N2%ADiagnostic Status/
2239 025250  posds0: .asciz  /%N%A          WORD 0:                 %O6%A(0)/
2240 025330  posds1: .asciz  /%N%A          WORD 1:                 %O6%A(0)/
2241 025410  possn:  .asciz  /%N%AServer Number:                    %O6%A(0)/
2242 025470  posrvn: .asciz  /%N%ARom Version Number:              /
2243 025540  possvn: .asciz  /%N%ASoftware Version Number:         /
2244 025610  posnam: .asciz  /%N%AServer Name:                     /
2245 025660  posloc: .asciz  /%N%AServer Location:                 /
2246 025730  posstr: .asciz  /%T/
2247
```

```
2248            .even
2249 025734    PCMSG:: .asciz  /%N%APC OF CALLING ROUTINE = %O6/
2250            .even
2251 025774    cmperh: .asciz  /%N%ACOMPARE ERRORS IN LOOP MESSAGE%N2/
2252 026042    cmper1: .asciz  /%N%AWord number:%D4%A(D) Expected=%O6%A(O)/
2253 026115    cmper2: .asciz  /%A Recieved=%O6%A(O)/
2254 026142    cmper3: .asciz  /%N%ATotal mismatches in message = %D4/
2255 026210    lgerme: .asciz  /%N%ALength Error -- Bytes Expected: %O6%A Bytes Received: %O6/
2256 026306    summe1: .asciz  /%N%A      NODE        RECEIVES RECEIVES NOT LENGTH COMPARE  BYTES         BYTES/
2257 026426    summe2: .asciz  /%N%A    ADDRESS       COMPLETE   COMPLETE   ERRORS  ERRORS COMPARED    TRANSFERRED%N/
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
```

```
                                .SBTTL  GLOBAL ERROR REPORT SECTION

                                ;++
                                ; THE GLOBAL ERROR REPORT SECTION CONTAINS MESSAGE PRINTING AREAS
                                ; USED BY MORE THAN TEST TO OUTPUT ADDITIONAL ERROR INFORMATION.  PRINTB
                                ; (BASIC) AND PRINTX (EXTENDED) CALLS ARE USED TO CALL PRINT SERVICES.
                                ;--


026624                          BGNMSG  ERR1
026624                                  PRINTX  @PCMSG,PCCALL
026650                                  DOCLN
026652                          ENDMSG

026654                          BGNMSG  ERR2
026654  010146                          MOV     R1,-(SP)
026656  013701  001170'                 MOV     P$TYPE,R1
026662  006301                          ASL     R1
026664  062701  001414'                 ADD     @MSGTAB,R1
026670                                  PRINTX  @EMSG35,@STRBUF
026714                                  PRINTX  @EMSG36,(R1)
026736  012601                          MOV     (SP)+,R1
026740                          ENDMSG

026742                          BGNMSG  ERR3
026742                                  PRINTX  @EMSG37,@STRBUF
026766                                  PRINTX  @EMSG38,@STRBU1
027012                          ENDMSG

                                ;*****************************************************************************
                                ;       THESE MESSAGE AREAS ARE USED TO OUTPUT SUPPLEMENTARY INFORMATION
                                ;       AFTER AN ERROR CALL.  THEY ARE INVOKED BY APPENDING THE NAME
                                ;       OF THE AREA TO AN ERROR CALL:  ERRXXX 1,ERRORMESSAGE,AREANAME.
                                ;       THE CORRESPONDING MESSAGE AREA IS SET UP IN THIS SECTION:
                                ;               BGNMSG  AREANAME
                                ;               [CODE]
                                ;               ENDMSG
                                ;
                                ;       THE AREAS IN THIS SECTION ARE FOR MESSAGES USED IN MORE THAN ONE
                                ;       TEST.  USE THE PRINTB (PRINT BASIC) AND PRINTX (PRINT EXTENDED)
                                ;       MACROS.
                                ;*****************************************************************************
```

```
                        .SBTTL  GLOBAL SUBROUTINES SECTION

                        ;++
                        ; THE GLOBAL SUBROUTINES SECTION CONTAINS THE SUBROUTINES
                        ; THAT ARE USED IN MORE THAN ONE TEST.
                        ;--

                        ;++
                        ; FUNCTIONAL DESCRIPTION:
                        ;    SUBROUTINE TO....

                        ;*****************************************************************************
                        ;       COMPLETE THE "SUBROUTINE TO...." STATEMENT WITH A FUNCTIONAL
                        ;       DESCRIPTION OF THIS SUBROUTINE.
                        ;*****************************************************************************

                        ; INPUTS:

                        ;*****************************************************************************
                        ;       LIST THE INPUT DATA THAT ARE EXPLICITLY PASSED TO THIS SUBROUTINE.
                        ;*****************************************************************************

                        ; IMPLICIT INPUTS:

                        ;*****************************************************************************
                        ;       LIST THE INPUT DATA THAT ARE IMPLICITLY USED BY THIS SUBROUTINE;
                        ;       FOR EXAMPLE, DATA READ FROM COMMON AREAS.
                        ;*****************************************************************************

                        ; OUTPUTS:

                        ;*****************************************************************************
                        ;       LIST THE OUTPUT DATA THAT ARE EXPLICITLY GIVEN BY THIS SUBROUTINE
                        ;*****************************************************************************

                        ; IMPLICIT OUTPUTS:

                        ;*****************************************************************************
                        ;       LIST THE OUTPUT DATA THAT ARE IMPLICITLY GIVEN BY THIS SUBROUTINE;
                        ;       FOR EXAMPLE, DATA STORED IN COMMON AREAS.
                        ;*****************************************************************************

                        ; SUBORDINATE ROUTINES USED:

                        ;*****************************************************************************
                        ;       LIST THE SUBROUTINES CALLED BY THIS SUBROUTINE.
                        ;*****************************************************************************

                        ; FUNCTIONAL SIDE EFFECTS:

                        ;*****************************************************************************
                        ;       DESCRIBE ANY EFFECTS THIS SUBROUTINE MAY HAVE UPON OTHER
                        ;       MODULES OF THE DIAGNOSTIC PROGRAM.  AN EXAMPLE OF THIS IS
                        ;       THE SUBROUTINE INHIBITS ALL INTERRUPTS WITH PRIORITY 7.
                        ;*****************************************************************************

                        ; 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                           ;*****************************************************************************
 2396                           ;     INSERT THE CODE FOR THIS SUBROUTINE.   THE NAME OF THE SUBROUTINE SHOULD
 2397                           ;     BE DEFINED WITH A DOUBLE-COLON (::);  THIS WILL MAKE THE SUBROUTINE GLOBAL.
 2398                           ;*****************************************************************************
 2400
 2402                           ;*****************************************************************************
 2403                           ;     BEGIN EACH SUBROUTINE AT THE TOP OF A NEW PAGE.
 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)+
 2445 027032  012122                   mov     (R1)+,(R2)+             ; Load clock's intr. vector into "CLKVEC"
```

```
2446 027034  012122                    mov     (R1)+,(R2)+             ; Load clock's freq. into "CLKHZ'
2447 027036  000207                    rts     PC
2448
```

```
                                .sbttl  CLKINT  Clock Interrupt Servide Routine

                                ;- +
                                ; Functional Description:
                                ;               This is the clock interrupt service routine which takes care
                                ;               of keeping the "time-since-start" and counting down any of the
                                ;               "event" timers.  The timers are used to time completion of
                                ;               device requests.  The "time-since-start" is used to be logged
                                ;               with each entry into the event log.
                                ;
                                ; Inputs - Implicit -
                                ;               TIMTCK - The current no. of ticks left to be counted until
                                ;                        a second has been counted off
                                ;               CLKHZ -  The no. of ticks in a second, determined by the
                                ;                        sys. line freq.
                                ;               TIMMIN & TIMSEC - Current value of "time-since-start" in
                                ;                        minutes and seconds
                                ;               TIMER 1,2 and S - Current values of "event timers"
                                ;
                                ; Outputs - Implicit -
                                ;               New value of event timer "1" & "2" decremented by 1 tick
                                ;               if it was non-zero
                                ;               New value of event timer "S" decremented by 1 second if it
                                ;               was non-zero
                                ;
                                ; Calling procedure : This routine is entered upon clock interrupt
                                ;
                                ; Side effects -
                                ;               The clock is disabled upon entry and reenabled when leaving
                                ;
                                ; Subordinate Routines - none
                                ;
                                ; Register Usage - none
                                ;
                                ;--+

027040                                  BGNSRV  CLKINT

027040  005077  152762                  clr     @CLKCSR                 ; disable the clock from interrupting
027044  005337  002044'                 dec     TIMTCK                  ; decrement the no. of ticks/sec
027050  001015                         bne     1$                      ; go check timers
027052  013737  002034'  002044'        mov     CLKHZ,TIMTCK            ; reset the no. of ticks/sec.
027060  005237  002042'                 inc     TIMSEC                  ; inc. no of secs-since-start
027064  022737  000074  002042'        cmp     #60.,TIMSEC             ; see if we've counted 60 sec.s yet
027072  001004                         bne     1$                      ; if not, go check timers
027074  005237  002040'                 inc     TIMMIN                  ;  else, inc. minutes-since-start
027100  005037  002042'                 clr     TIMSEC                  ;  and restart second counter

027104  005737  002046'          1$:    tst     TIMER1                  ; see if TIMER1 timing anything
027110  001402                         beq     2$                      ;  if=0, no, check next timer
027112  005337  002046'                 dec     TIMER1                  ;  else decrement the timer value (by 1 tick)
027116  005737  002050'          2$:    tst     TIMER2                  ; see if TIMER2 timing anything
027122  001402                         beq     3$                      ;  if=0, no, check next timer
027124  005337  002050'                 dec     TIMER2                  ;  else decrement timer value (by 1 tick)
027130  005737  002052'          3$:    tst     TIMERS                  ; see if TIMERS timing anything
027134  001406                         beq     4$                      ;  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,@CLKCSR           ; 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                         PREG14::
 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     1$              ;
 2561 027210                                 ERRSF   1,EMSG54,ERR1   ; print stack overflow error ... and depart!
 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                                    ; Functional Description:
2579                                    ;               This routine is called to wait for the Done Interrupt bit (DNI)
2580                                    ;               of PCSR0 to be set signifying the completion of a port command.
2581                                    ;               If the DEUNA/DELUA reports some sort of error, ERRFLG will
2582                                    ;               have been raised in the interrupt service routine.   In this
2583                                    ;               case the error reporting routine will be called.
2584                                    ;
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 occured
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$
2614 027272  005714                    2$:     tst     (r4)                    ;has timer expired?
2615 027274  001365                            bne     1$                      ; br if no to wait for interrupt
2616 027276  000403                            br      5$                      ;br to 5$
2617 027300                            3$:     call    ERROR                   ;call error routine
2618 027306  012702  177777            5$:     mov     #-1,r2                  ;indicate failure
2619 027312                            6$:     return  r2                      ;return with success/failure indication
2620
2621                                    .sbttl  ERROR   Handle UNA interrupt errors
2622
2623                                    ;--+
2624                                    ; Functional Description:
2625                                    ;               This subroutine checks the error flags set by
2626                                    ;               UNAISR the interrupt service routine and prints
2627                                    ;               out the approriate error messsages.
2628                                    ;
2629                                    ; Inputs - implicit -
2630                                    ;               error flags should be set by UNAISR routines.
```

```
                                ; Outputs - implicit -
                                ;               error messages are printed out to the operator console.
                                ;
                                ; calling sequence:
                                ;               call ERROR
                                ;
                                ; Side effects -
                                ;               1.) error flags that were set in UNAISR are cleared here.
                                ;               2.) errors will be reported at the user's terminal
                                ;               3.) the diagnostic will be exited
                                ;
                                ; Subordinate routines -
                                ;               ERR1 - extended error report
                                ;
                                ;--+

027316  005337  003020'         ERROR:: dec     errflg                  ;decriment error counter to show
                                                                        ;that it has been handled
027322  005737  003004'                 tst     pceflg                  ;see if port command error
027326  001016                         bne     5$                      ; if yes, branch
027330  005737  003002'                 tst     fatflg                  ;see if UNA fatal error
027334  001022                         bne     10$                     ; if yes, branch
027336  005737  003016'                 tst     bcount                  ;see if unexplained interrupt
027342  001026                         bne     15$                     ; if yes, branch
027344  005737  003014'                 tst     rbfcnt                  ; receive buffers unavailable?
027350  001032                         bne     18$                     ; branch if yes
027352                                 errdf   2,emsg04,err1           ;else unknown error
027362  000433                         br      20$                     ; exit

027364  005337  003004'         5$:     dec     pceflg                  ; indicate that it was handled
027370                                 errdf   3,emsg01,err1           ;port command error
027400  000424                         br      20$                     ; exit

027402  005337  003002'         10$:    dec     fatflg                  ; keep up on book keeping
027406                                 errdf   4,emsg02,err1           ;UNA fatal error
027416  000415                         br      20$                     ; exit

027420  005337  003016'         15$:    dec     bcount                  ; book keeping
027424                                 errdf   5,emsg03,err1           ;unexplained interrupt
027434  000406                         br      20$                     ; exit

027436  005337  003014'         18$:    dec     rbfcnt
027442                                 errdf   6,emsg43,err1           ; report it

027452                          20$:    return                          ;return

                                ;--+
                                ; Name - DEVSTART                       Start the DELUA/DEUNA
                                ;
                                ; Functional Description:
                                ;               This routine is called to start up the DELUA/DEUNA.
                                ;               The transmit and receive rings will be reset with their
                                ;               associated pointers reset to the beginnings of their
                                ;               respective rings.  This is done because when given a
                                ;               start port command, the DELUA or DEUNA will reset its
                                ;               pointers to the host rings.
                                ;                       After resetting the rings, a START port command
```

```
                                        ;               will be issued, causing the DELUA/DEUNA to transition to
                                        ;               the running state.
                                        ;
                                        ; Inputs - none
                                        ;
                                        ; Outputs - none
                                        ;
                                        ; Calling Procedure: CALL DEVSTART
                                        ;
                                        ; Side Effects -
                                        ;               1.) transmit and receive rings are reset, and
                                        ;               2.) the DELUA/DEUNA is in the running state
                                        ;
                                        ; Subordinate Routines - none
                                        ;
                                        ; Register Usage -
                                        ;               R1 - pointer to transmit and receive rings
                                        ;               R2 - scratch
                                        ;
                                        ;--+
027454                                  DEVSTART:
                                        ;--+
                                        ;       Reset transmit and receive ring pointers
                                        ;--+
027454  013737  002066' 002072'         mov     XRGSRT,XRGCUR                   ; point them ...
027462  013737  002066' 002076'         mov     XRGSRT,XRGNXT                   ; ... all to the ...
027470  013737  002070' 002074'         mov     RRGSRT,RRGCUR                   ; ... beginning of their ...
027476  013737  002070' 002100'         mov     RRGSRT,RRGNXT                   ; ... associated rings.

                                        ;--+
                                        ;       Clear the ownership bit of all entries in the transmit ring.  This
                                        ;       will make us the owner of all entries.
                                        ;--+
027504                                  CALL    REMAP   #OTRING                 ; enable access to transmit ring
027516  012702  000004                  mov     #NO.NTR,R2                      ; R2 is loop control
027522  013701  002072'         10$:    mov     XRGCUR,R1                       ; point R1 to transmit ring
027526  042761  100000  000004         bic     #own,4(R1)                      ; we own all entries
027534                                  CALL    GETXNX  #XRGCUR                 ; point to next entry
027546  005302                          dec     R2                              ; do for all ring entries
027550  001364                          bne     10$                             ;

                                        ;--+
                                        ;       Give ownership of all receive ring entries to the DELUA/DEUNA by
                                        ;       setting each entry's OWN bit.
                                        ;--+
027552                                  CALL    REMAP   #ORRING                 ; enable access to receive ring
027564  012702  000010                  mov     #NO.NRR,R2                      ; R2 is loop control
027570  013701  002074'         20$:    mov     RRGCUR,R1                       ; point R1 to receive ring
027574  052761  100000  000004         bis     #own,4(R1)                      ; DELUA/DEUNA owns all entries
027602                                  CALL    GETRNX  #RRGCUR                 ; point to next entry
027614  005302                          dec     R2                              ; do for all ring entries
027616  001364                          bne     20$                             ;

                                        ;--+
                                        ;       Now put the device in the running state by issueing a START port
                                        ;       command.
                                        ;--+
```

```
 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$:    CALL    RETMEM                          ; restore memory mapping
 2751 027654                            RETURN                                  ; leave ...
 2752
 2753
 2754                           ;--+
 2755                           ; Name - STOP                                   Stop the DELUA/DEUNA
 2756                           ;
 2757                           ; Functional Description:
 2758                           ;               This routine is called to stop the DELUA/DEUNA and
 2759                           ;               leave it in the ready state.
 2760                           ;
 2761                           ; Inputs - none
 2762                           ;
 2763                           ; Outputs - none
 2764                           ;
 2765                           ; Calling Procedure: CALL DEVSTOP
 2766                           ;
 2767                           ; Side Effects -
 2768                           ;               1.) The DELUA/DEUNA will be left in the ready state
 2769                           ;
 2770                           ; Subordinate Routines - none
 2771                           ;
 2772                           ; Register Usage -
 2773                           ;               R1 - return status of STOP port command
 2774                           ;
 2775                           ;--+
 2776 027656                    DEVSTOP:
 2777 027656                            CALL    COMAND  #STOP                   ; Issue the STOP port command
 2778 027670                            P$POP   R1                              ; get return status
 2779 027672  001404                    BEQ     10$                             ; leave if okay
 2780 027674                            ERRDF   8,EMSG47,ERR1                   ; 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                           ; Functional Description:
 2791                           ;               The purpose of this routine is to initialize and startup
 2792                           ;               the DELUA/DEUNA.  The initilization of the DELUA/DEUNA is
 2793                           ;               as follows:
 2794                           ;
 2795                           ;               1.) Issue a GET PCBB port command to tell the device where
 2796                           ;                   the port control block is located in host memory.
 2797                           ;
 2798                           ;               2.) Issue a write ring descriptor port command to tell the
 2799                           ;                   device where the receive and transmit rings are located
 2800                           ;                   in host memory.
 2801                           ;
```

```
                                ;               The device is then started by issueing a START port command.
                                ;               Then the devices physical address is read and stored.
                                ;
                                ; Inputs - none
                                ;
                                ; Outputs - none
                                ;
                                ; Calling Procedure: CALL UNAINI
                                ;
                                ; Side effects -
                                ;               PHYADR - contains the device's default physical address
                                ;
                                ; Subordinate Routines -
                                ;               COMAND - subroutine to issue a port command
                                ;               FUNCT - subroutine to issue an ancillary port command
                                ;               REMAP - used to modify KPAR4 and KPAR5 so that receive/transmit
                                ;                       rings can be accessed
                                ;
                                ; Register Usage -
                                ;               R1, R2 - scratch
                                ;               R3 - contains address of PCSR0
                                ;               R4 - pointer to memory location to hold devices's physical
                                ;                    address
                                ;
                                ;--*
027706                          UNAINI::
                                ;--*
                                ;       Reset the DELUA/DEUNA then enable device interrupts
                                ;--*
027706  013703  002106'                 mov     PCSR0, R3                       ; move address of PCSR0 to R3
027712  042713  000100                  bic     #inte,(R3)                      ; disable interrupts
027716  012713  000040                  mov     #reset, (R3)                    ; hardware reset una
                                
027722  005002                          clr     r2                              ; loop counter init
027724  011301                  7$:     mov     (R3), r1                        ; read PCSR0
027726  032701  004000                  bit     #DNI, r1                        ; wait for command to finish
027732  001006                          bne     9$                              ; back til DNI =1
027734  005302                          dec     r2                              ; count down delay
027736  001372                          bne     7$                              ; back until timeout
027740                                  errdf   9,EMSG15,ERR1                   ; print " DNI Did not set from"
                                                                                ;       " a RESET"
027750  012713  004000          9$:     mov     #dni, (R3)                      ; write one to clear DNI
027754  052713  000100                  bis     #inte, (R3)                     ; enable interrupts
                                
                                ;--*
                                ;       Tell the device where the port comand block is located in
                                ;       host memory
                                ;--*
027760  012763  002150' 000004          mov     #PCBB0,4(r3)                    ; lower 16 bits of adrs
027766  005063  000006                  clr     6(r3)                           ; upper 2
                                
027772                                  call    comand  #getpcb                 ; load address
030004                                  P$POP   r2                              ; get success/failure report
030006  001404                          beq     10$                             ; continue if OK
030010                                  errdf   10,emsg05,err1                  ; else report error
                                
030020                          10$:
```

```
                                        ;--+
                                        ;       Write the rings ...
                                        ;
                                        ;--+
030020                                          call    funct @wdrngs                   ; write descriptor rings
030032                                          P$POP   R2                              ; check for error
030034  001407                                  beq     20$                             ; if OK, continue
030036                                          errdf   11,emsg16,err1                  ; else report error
                                        
030046                                          call    devstart                        ; start up the DELUA/DEUNA
                                        
                                        ;--+
                                        ;       Read the device's physical address and save it in the variable
                                        ;       PHYADR.
                                        ;--+
                                        
030054                                  20$:    call    funct @rdphys                   ; read una physical address
030066                                          P$POP   r2                              ; check for error
030070  001404                                  beq     25$                             ; if OK, continue
030072                                          errdf   12,emsg06,err1                  ; else  report error
                                        
030102  012701  002152'                  25$:    mov     #PCBB2, R1                      ; store physical address
030106  012704  002244'                          mov     #PHYADR, R4
030112  012124                                  mov     (R1)+,(R4)+                     ; move first two bytes
030114  012124                                  mov     (R1)+,(R4)+                     ;  and second two
030116  011114                                  mov     (R1),(R4)                       ;  and done
                                        
030120                                          CALL    RETMEM                          ; restore memory mapping
030126                                          RETURN
                                        
                                        
                                        .sbttl  unaisr  una interrupt service routine
                                        ;--+
                                        ; Functional Description:
                                        ;               This is the interrupt service routine for the DELUA/DEUNA.
                                        ;               Each time this routine is entered, the following takes place:
                                        ;
                                        ;               1.) All CSRs are saved for debug
                                        ;
                                        ;               2.) All write-one-to-clear bits are cleared
                                        ;
                                        ;               3.) flags corresponding to all bits, except port command
                                        ;                   field, of PCSR0 are set if the corresponding bits in PCSR0
                                        ;                   are set.
                                        ;
                                        ;               4.) and, If an error has occurred, then ERRFLG is set
                                        ;
                                        ; Inputs - none
                                        ;
                                        ; Outputs - Implicit -
                                        ;               flags are set corresponding to the set bits in PCSR0
                                        ;
                                        ; Calling Procedure: the routine is an interrupt routine, so it is vectored
                                        ;               to on device interrupt
                                        ;
                                        ; Side effects - none
                                        ;
```

```
                                ; Subordinate Routines - none
                                ;
                                ; Register Usage -
                                ;               R1 - address of PCSR0
                                ;               R3 - contents of PCSR0
                                ;
                                ;--*

030130                          BGNSRV  UNAISR

030130  010146                          mov     r1,-(sp)                ;save r1
030132  010246                          mov     r2,-(sp)                ;...
030134  010346                          mov     r3,-(sp)                ;...

030136  005003                          clr     r3                      ;setup write 1 to clr mask
030140  013701  002106'                 mov     pcsr0,r1                ;get pcsr0 address

030144  011103                          mov     (r1),r3                 ;and its contents

030146  012137  002116'                 mov     (R1)+,PCSR0C            ;save pcsr's for debug
030152  012137  002120'                 mov     (R1)+,PCSR1C
030156  012137  002122'                 mov     (R1)+,PCSR2C
030162  011137  002124'                 mov     (R1),PCSR3C
030166  013701  002106'                 mov     PCSR0,R1

030172  000303                          swab    r3                      ;reorient contents of pcsro
030174  110361  000001                  movb    r3,1(r1)                ;write one to clear
                                                                        ; ONLY CLEAR UPPER BYTE
030200  000303                          swab    r3                      ;reorient contents of pcsro

030202  032703  100400                  bit     #seri!fati,r3           ;any fatal status ??
030206  001403                          beq     10$

030210  005237  003002'                 inc     fatflg                  ;set flag
030214  000441                          br      90$                     ;exit

030216  032703  040000          10$:    bit     #pcei,r3                ;port command error interrupt?
030222  001402                          beq     30$                     ;no
030224  005237  003004'                 inc     pceflg                  ;yes, increment flag

030230  032703  010000          30$:    bit     #txi,r3                 ;transmit interrupt ??
030234  001402                          beq     40$                     ;no
030236  005037  003010'                 clr     xflag                   ;yes, set flag

030242  032703  004000          40$:    bit     #dni,r3                 ;command done ??
030246  001402                          beq     45$                     ;no
030250  005237  003012'                 inc     dniflg                  ;yes, count each dni

030254  032703  002000          45$:    bit     #rcbi,r3                ;recieve buffer unavailable?
030260  001405                          beq     50$                     ;no

030262  105737  001274'                 tstb    p$list                  ; are we listening?
030266  001014                          bne     90$                     ; YES, we'll have to ignore this
030270  005237  003014'                 inc     rbfcnt                  ; NO, count them

030274  032703  034000          50$:    bit     #rxi!txi!dni,r3         ;check for non-error interrupt
```

```
 2973 030300  001007                        bne     90$                     ;exit if one occured
 2974 030302  032703  142000                bit     #seri!prei!rcbi,r3      ;check for error interrupt
 2975 030306  001002                        bne     80$                     ;if one occured, incr. errflg
 2976 030310  005237  003016'                inc     bcount                  ;else, nonsense interrupt
 2977 030314  005237  003020'        80$:    inc     errflg
 2978 030320  012603                 90$:    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 occured.
 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 occured. 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 occured, 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::
 3015 030330                                P$POP   R1              ;MOVE COMMAND TYPE TO R1
 3016 030332  052701  000100                BIS     #INTE,R1        ;ADD INTERRUPT TO COMMAND
 3017 030336  010177  151544                MOV     R1,@PCSR0       ;MOV COMMAND TO PCSR0
 3018 030342                                CALL    WAIT            ;WAIT FOR DONE INTERRUPT
 3019 030350                         10$:    RETURN                  ;RETURN - ERROR INFO STILL ON
 3020                                                                ; PARAMETER STACK FROM WAIT SUB.
 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                                ;
```

```
                                ; Inputs -
                                ;               P1 - The DELUA/DEUNA Port Function mnemonic of the
                                ;                    desired function.
                                ; Outputs -
                                ;               P2 - Success report.  Contains 0 for success
                                ;                    -1 if a DELUA/DEUNA error occured,
                                ;                    This parameteris passed directly from the
                                ;                    COMAND sub and is not affected by FUNCT.
                                ;
                                ; Calling procedure - Call  FUNCT #<function type>
                                ;
                                ; Side effects - none
                                ;
                                ; Subordinate routines -
                                ;               COMAND - used to issue a GET COMMAND port command
                                ;
                                ; Register usage -
                                ;               R1 - contains the function type, which is transformed
                                ;                    to the address of the function specific PCB.
                                ;               R2 - contains the address of the DELUA/DEUNA PCB.
                                ;
                                ;--*

030352                          FUNCT:: P$POP    R1                     ; get function type into R1
030354  006301                         asl     R1                      ; multiply by two
030356  062701  002160'                 add     #funtab,R1              ; add function table offset
                                                                        ; R1 now contains address of address
                                                                        ;  of function specific PCB
030362  012702  002150'                 mov     #PCBB0, R2              ; put DELUA/DEUNA PCB into R2
030366  011101                         mov     (R1),R1                 ; put address of PCB into R1
030370  012122                         mov     (R1)+,(R2)+             ; mov pcb's
030372  012122                         mov     (R1)+,(R2)+             ; mov pcb's
030374  012122                         mov     (R1)+,(R2)+             ; mov pcb's
030376  012122                         mov     (R1)+,(R2)+             ; mov pcb's
030400                                 call COMAND #getfnt             ; issue get port function command
030412                                 return                          ; success info from COMAND subroutine
                                                                        ; is still on parameter stack
                                .sbttl  XMIT    Transmit DELUA/DEUNA frames

                                ;--*
                                ; Functional Description:
                                ;               This subroutine is used to transmit frames over the DELUA/
                                ;               DEUNA.  It sets up the transmit ring for the buffer to be
                                ;               transmitted, namely the status bits (STP,ENP,OWN) and message
                                ;               length.  Then a POLL DEMAND port command is issued to alert
                                ;               the device that we have something to transmit.
                                ;
                                ; Inputs - Implicit
                                ;               The buffer that is pointed to by the ring entry that is
                                ;               pointed to by XRGCUR has been loaded with the data that will
                                ;               be transferred.  Also, the variable BUFLEN has been set to
                                ;               the number of bytes to transmit.
                                ;
                                ; Outputs -     P1 - Success report =>  0 = success, -1 = failure
                                ;
                                ;       Implicit - 'RETRYS' : nonzero if transmit failed due to
```

```
                                ;                       traffic.
                                ;
                                ; Calling procedure: Call XMIT
                                ;                   P$POP P1
                                ;
                                ; Side effects - The ring pointer XRGNXT will be updated to point the next
                                ;                available entry after the transmit operation.
                                ;
                                ; Subordinate Routines -
                                ;               COMAND - issues poll demand
                                ;               GETXNX - updates transmit ring pointer
                                ;               REMAP  - used to remap memory so that the transmit ring may
                                ;                        be accessed
                                ;               RETMEM - used to return the mapping of memory to its original
                                ;                        state
                                ;
                                ; Register Usage - R1 points to timout timer location
                                ;                  R2 is used as a pointer if retrys is set
                                ;                  R3 is used to pass the success/failure message back
                                ;                  R4 is used as a pointer to ring entries or status info.
                                ;--*

030414                          XMIT::
030414                                  CALL    REMAP   #OTRING         ; enable access to transmit memory

030426  005037  003024'                 clr     retrys
030432  013704  002072'         1$:     mov     xrgcur,R4               ; move ring entry location into R4
030436  032764  100000  000004          bit     #own,4(R4)              ; make sure we own this
030444  001127                          bne     40$                     ;  else, bookkeeping error
030446  013714  003126'                 mov     buflen,(R4)             ; move buffer length into first word of
                                                                        ; next available ring entry
030452  052764  101400  000004          bis     #own!stp!enp,4(R4)      ; set ownership, start and end of frame bits
030460  012737  000001  003010' 20$:     mov     #1,xflag                ; set transmit flag
030466                                  call    comand #pdmd            ; issue pdmd command
030500                                  P$POP   R3                      ; check for errors
030502  001130                          bne     50$                     ; if yes, exit
030504  012701  002050'         22$:    mov     #TIMER2,R1              ; set up to wait for transmit to complete
030510  012711  000100                  mov     #100,(R1)
030514  005737  003010'         23$:    tst     xflag                   ; see if transmit done bit set
030520  001403                          beq     24$                     ;  if set, skip wait loop
030522  005711                          tst     (R1)                    ;  else, see if timeout yet
030524  001373                          bne     23$                     ;   no, wait
030526  000510                          br      45$                     ;   yes, exit
030530  032764  100000  000004  24$:    bit     #own,4(R4)              ; see who owns this entry
030536  001072                          bne     40$                     ; if DELUA/DEUNA still owns this, somethings wrong
030540  032764  040000  000004          bit     #errs,4(R4)             ; see if any errors
030546  001015                          bne     30$                     ;  if yes, branch and take care of them
030550                          26$:    CALL    GETXNX  #xrgcur         ; update "transmit ring current" pointer
030562  005003                          clr     R3                      ;  indicate success
030564  023737  002072'  002076'        cmp     xrgcur,xrgnxt           ; see if current pointer = next pointer
030572  001054                          bne     40$                     ;  if no, error
030574  005037  003024'                 clr     retrys                  ; let 'retrys' reflect success
030600  000473                          br      55$                     ; return
030602  032764  016000  000004  30$:    bit     #def!one!more,4(R4)     ; was message still sent?
030610  001357                          bne     26$                     ;  if yes, go to next one
030612  032764  002000  000006          bit     #rtry,6(R4)             ; else, did DELUA/DEUNA give up after 16 tries
030620  001434                          beq     32$                     ;  if not, fatal device error, exit
```

```
3144 030622  005237  003024'                inc     retrys                  ;  if yes, keep count of them
3145 030626  022737  000003  003024'        cmp     #3,retrys               ;  how many tries?
3146 030634  100440                         bmi     43$                     ;  give up after 3 attempts
3147 030636                                 call    getxnx  #xrgcur         ;  update pointers
3148 030650                                 call    getxnx  #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$:    errsf   14,emsg10,err1          ; transmit ring bookkeeping error
3162 030734  000413                         br      50$
3163
3164 030736                         43$:    errhrd  15,emsg49               ; indicate failed due to execessive ...
3165 030746  000406                         br      50$                     ; ... retries and split!!
3166
3167 030750  005237  003022'        45$:    inc     TIMOUT
3168 030754                                 errdf   16,emsg08,err1          ; report error
3169
3170 030764  012703  177777         50$:    mov     #-1,R3                  ; error indicator
3171
3172 030770                         55$:    CALL    RETMEM                  ; remap memory to its original value
3173 030776                                 return  R3                      ; return
3174
3175                                 .sbttl  RECEVE  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 RECEVE,
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 RECEVE
```

```
3201                                    ;                       P$POP  P1
3202                                    ;
3203                                    ; Side effects -
3204                                    ;               1.) The pointers RRGCUR and RRGNXT are updated.
3205                                    ;               2.) KPAR4 and KPAR5 are left mapping to the receive ring.  This
3206                                    ;                   is done because this structure is consistently accessed
3207                                    ;                   immediately after a call to RECEVE
3208                                    ;
3209                                    ; Subordinate Routines -
3210                                    ;               GETRNX - updates RRGCUR and RRGNXT
3211                                    ;               COMAND - used to issue poll demand
3212                                    ;               REMAP  - used to remap memory so that the receive ring may
3213                                    ;                        be accessed.
3214                                    ;               RELBUF - used to release unwanted receive buffers
3215                                    ;
3216                                    ; Register usage - R1 is used to hold current frame status information
3217                                    ;                  R2 counts the number of frames handled
3218                                    ;                  R4 points to the ring descriptor entry
3219                                    ;
3220                                    ;--+
3221
3222 031002                            RECEVE::
3223 031002  005002                            clr     R2                      ; clear frames handled counter
3224
3225 031004                            1$:     CALL    REMAP   #ORRING         ; allow access to receive ring
3226 031016  013704  002074'                   mov     rrgcur,R4               ; move current receive ring pointer to R4
3227 031022  016401  000004                    mov     4(R4),R1                ; move status of frame to R1
3228 031026  032701  100000                    bit     #own,R1                 ; see who owns this buffer
3229 031032  001070                            bne     60$                     ;  if una owns it, return
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'                   tstb    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$:    bit     #err,R1                 ; see if any errors
3252 031130  001421                            beq     20$                     ;  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
```

```
                                ;       to send (sys. ID) and receive (prom. mode) it gets a CRC error.
                                ;--*
                                
031132  105737  001274'        tstb    p$list                  ; Are we executing listen command
031136  001403                  beq     15$                     ; No, go log error
031140  032701  004000          bit     #crc,R1                 ; Is this a CRC error?
031144  001350                  bne     5$                      ; yes, just leave without logging error
                                                                ; else,
031146  005237  003026'  15$:   inc     rcverr                  ; increment receive error counter
031152                          printf  #recerr                 ; print error message
031172  000735                  br      5$                      ; update pointers and return
031174  005237  003030'  20$:   inc     rcvbuf                  ; increment good buffers received counter
031200  005202                  inc     R2                      ; keep count of how many buffers received
                                
031202                          CALL    GETRNX  #RRGCUR         ; update "receive ring current" pointer
031214                  60$:    return  R2                      ; return with number of entrys handled
                                
                                ;--*
                                ; Name - RELBUF                         Release a receive buffer
                                ;
                                ; Functional Description
                                ;               This routine is called to release a receive buffer to the
                                ;               DELUA/DEUNA.  It will set the ownership of a receive ring
                                ;               entry and then issue a poll demand port command to alert
                                ;               the device of an available buffer.
                                ;
                                ; Inputs - Explicit -
                                ;               P1 - pointer to receive ring entry
                                ;
                                ; Outputs - none
                                ;
                                ; Calling Procedure:    CALL RELFBUF P1
                                ;
                                ; Side Effects -
                                ;               1.) The ownership of the ring entry pointed to by P1 goes
                                ;                   to the device.
                                ;               2.) If the poll demand fails then an error is printed and
                                ;                   the diagnostic is exited
                                ;
                                ; Subordinate Routines - noe
                                ;
                                ; Register usage -
                                ;               R1 - pointer to receive ring entry
                                ;
                                ;--*
031220                  RELBUF::
031220                          P$POP   R1                      ; get pointer to receive ring entry
031222                          CALL    REMAP   #ORRING         ; allow access to receive ring
031234  052761  100000  000004        BIS     #OWN,4(R1)      ; release the buffer to the device
031242                          CALL    COMAND  #PDMD           ; issue poll demand port command
031254                          P$POP   R1                      ; get success indicator
031256  001404                  BEQ     10$                     ; SUCCESS, continue
031260                          ERRDF   17,EMSG09,ERR1          ; print error message
                                
031270                  10$:    CALL    RETMEM                  ; restore memory mapping
031276                          RETURN                          ; later ....
                                
```

```
 3315
 3320                                   ;--+
 3321                                   ; Functional Description:
 3322                                   ;
 3323                                   ;               This routine will convert a string of HEX characters into a
 3324                                   ;               right justified binary  stream  (with leading zeros),
 3325                                   ;               compatible  with Ethernet conventions.   The source string must
 3326                                   ;               be formatted using either  a  word  by  word  hex  description
 3327                                   ;               or a byte by byte hex description.  The  returned  string
 3328                                   ;               will  be  BYTE  oriented  as required by the Ethernet:
 3329                                   ;
 3330                                   ;               lo-byte-word0 hi-byte-word0 lo-byte-word1 hi-byte-word1, etc.
 3331                                   ;
 3332                                   ; Inputs -
 3333                                   ;               p1 - address of the source  (HEX)  string to be converted to
 3334                                   ;                    a binary  stream.
 3335                                   ;               p2 - address of the  desired  destination buffer which will
 3336                                   ;                    accept binary data
 3337                                   ;               p3 - length (in bytes) of the destination buffer
 3338                                   ;
 3339                                   ; Outputs -     p4 - zero if successful, -1 if buffer too long or odd number of
 3340                                   ;                    hex characters
 3341                                   ;
 3342                                   ;       Implicit - The  buffer  at  p2  will contain a right justified  binary
 3343                                   ;                  stream w/ leading zeros and  corresponding  to  hex string
 3344                                   ;                  at R5.
 3345                                   ;
 3346                                   ; Calling Procedure:    CALL EDPACK p1,p2,p3
 3347                                   ;                       P$POP   P4
 3348                                   ;
 3349                                   ; Side Effects - none
 3350                                   ;
 3351                                   ; Subordinate Routines -
 3352                                   ;               HXFORM - Strip non-HEX characters from input string
 3353                                   ;               HEXBIN - HEX to binary conversion
 3354                                   ;
 3355                                   ;---
 3356
 3357 031300                           locdst: .blkb   74.             ;max number of characters that may be enterred
 3358 031412  000000                   source: .word                   ;source address
 3359
 3360 031414                           EDPACK::
 3361 031414                                   p$pop   source,r4,r3            ;r4-destination, r3-number of chars rqd
 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
```

```
                                ;--+
                                ; Functional Description
                                ;               This routine is used to form a string of packed HEX characters.
                                ;               It accepts an input string and the number of characters
                                ;               to be used in the output sting.  Any spaces and dashes are
                                ;               stripped out of the string.  Invalid characters will cause
                                ;               an error to be returned.
                                ;
                                ; Inputs -      P1 - the address of the source string to be formatted.
                                ;               P2 - the address of a buffer to get the formatted string.
                                ;               P3 - the number of HEX characters to look for.
                                ;
                                ; Outputs -     P4 - pointer to the last valid charcter of the output string.
                                ;               P5 - success indicator - 0=success, -1=error.
                                ;
                                ; Calling Procedure - CALL HXFORM P1,P2,P3
                                ;                      P$POP     P4,P5
                                ;
                                ; Side effects - None
                                ;
                                ; Subordinate Routines - None
                                ;
                                ; Register Usage
                                ;               R1 - address of source string
                                ;               R2 - address of destin string
                                ;               R3 - number of HEX characters desired
                                ;               R4 - byte of source string/success indicator
                                ;
                                ;--+
031504                          HXFORM::
031504                                  P$POP   R1,R2,R3                ; Get inputs
                                
031512  112104                 5$:     MOVB    (R1)+,R4                ; get a byte of the source string
031514  120427  000040                  CMPB    R4,#40                  ; Are we looking at a space?
031520  001774                          BEQ     5$                      ; Yes, valid char., get next
031522  120427  000055                  CMPB    R4,#55                  ; Are we looking at a dash?
031526  001771                          BEQ     5$                      ; Yes, valid char., get next
                                
                                ;
                                ; Check to see if we've got a HEX digit.  ASCII range for HEX is 60 <= CHAR < 72
                                ; and 101 <= CHAR < 107
                                ;
                                
031530  120427  000060                  CMPB    R4,#60                  ; Is CHAR < 60?
031534  100417                          BMI     HXERR                   ; CHAR out of range - error
031536  120427  000072                  CMPB    R4,#72                  ; Is 60 <= CHAR < 72?
031542  100407                          BMI     10$                     ; CHAR is good
031544  120427  000101                  CMPB    R4,#101                 ; Is CHAR < 101?
031550  100411                          BMI     HXERR                   ; CHAR out of range - error
031552  120427  000107                  CMPB    R4,#107                 ; Is 101 <= CHAR < 107?
031556  100401                          BMI     10$                     ; CHAR is good
031560  000405                          BR      HXERR                   ; Else - error
                                
031562  110422                 10$:     MOVB    R4,(R2)+                ; put HEX digit in dest. string
031564  005303                          DEC     R3                      ; decrement # of chars. to find
031566  001351                          BNE     5$                      ; non-zero means more to do
031570  005004                          CLR     R4                      ; indicate success in R4
```

```
 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
 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 to hex digits
 3452                                 ;
 3453                                 ; Inputs -
 3454                                 ;               p1 - source string address  (delimitted 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                                         .even
 3479
 3480 031632                          HEXBIN::
 3481 031632                                 p$pop   r1,r2,hn        ;r1=source string address
 3482                                                                 ;r2=destination string address
 3483                                                                 ;hn=number of bytes required
 3484
 3485 031642  060237  031606'                add     r2,hn           ;hn now points to the last_byte_position+1
 3486
 3487 031646  012704  031610'          1$:     mov     #cmpstr,r4      ;pointer in the compare string
 3488 031652  121124                  2$:     cmpb    (r1),(r4)+      ;compare current char with a char in cmpstr
 3489 031654  001376                          bne     2$              ;repeat until character found in list
 3490 031656  005201                          inc     r1              ;point to the next ASCII byte
 3491 031660  162704  031611'                sub     #cmpstr+1,r4    ;r4 now contains the actual binary value for
 3492                                                                 ;the nibble described by the current byte.
```

```
 3493                                                           ;note: NIBBLE is the HI portion of the BYTE
 3494 031664  006304                        asl     r4         ;move nibble to the hi end of the byte
 3495 031666  006304                        asl     r4
 3496 031670  006304                        asl     r4
 3497 031672  006304                        asl     r4
 3498 031674  010403                        mov     r4,r3      ;save the hi nibble
 3499
 3500 031676  012704  031610'               mov     #cmpstr,r4 ;pointer into compare string
 3501 031702  121124              3$:       cmpb    (r1),(r4)+ ;compare current char with a char in cmpstr
 3502 031704  001376                        bne     3$         ;repeat until match found in cmpstr list
 3503 031706  005201                        inc     r1         ;point to the next ASCII byte
 3504 031710  162704  031611'               sub     #cmpstr+1,r4 ;r4 now contains the actual binary value for
 3505                                                           ;the nibble described by the current byte.
 3506                                                           ;note: NIBBLE is the HI portion of the BYTE
 3507 031714  050403                        bis     r4,r3      ;now the two characters have made a single byte
 3508                                                           ;now place the complete byte in the destination
 3509 031716  110322                        movb    r3,(r2)+   ;and point to the next destination byte
 3510 031720  020237  031606'               cmp     r2,hn      ;if the destination pointer [r2] reaches the
 3511 031724  100750                        bmi     1$         ;last character position+1 [hn] then done.
 3512 031726                                return                ;return to caller
 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                                ; Subordiate 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                hex:    .word
 3548
```

```
3549 031752                          BINHEX::
3550 031752                                  p$pop   r1,lst,r2       ;R1 has the input buffer address
3551                                                                 ;lst: has the number of bytes in input buffer
3552                                                                 ;R2 has the output buffer address
3553 031762  060137  031750'                   add     r1,lst          ;lst is now address of last source byte + 1
3554 031766  112103                  1$:     movb    (r1)+,r3        ;get the current byte and point to next byte
3555 031770  110304                          movb    r3,r4           ;separate nibbles and get characters separately
3556 031772  042703  177760                  bic     #177760,r3      ;only right binary nibble remains in r3
3557 031776  006204                          asr     r4              ;shift over for left binary nibble in 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,lst          ;result is negative until r1=lst
3568 032032  103755                          blo     1$              ;until r1=lst. (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                                 ; Functional Description:
3576                                 ;               This subroutine builds loop direct frames for transmission
3577                                 ;               from the DELUA/DEUNA. Source address, Destination address,
3578                                 ;               Prot. type, and loop direct header info are added
3579                                 ;               to the message buffer.  The message buffer is built
3580                                 ;               by a call to BLDBUF.
3581                                 ;
3582                                 ; Inputs -      P1 - The address of the destination address (from node table)
3583                                 ;               implicit - P$SIZE contains the size of the message buffer data
3584                                 ;                          XRGNXT points to the next available ring entry
3585                                 ;                          PHYADR holds the current local DELUA/DEUNA physical address
3586                                 ;
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                                 ;
3592                                 ; Calling procedure - CALL BLDLD P1
3593                                 ;
3594                                 ; Side effects - none
3595                                 ;
3596                                 ; Subordinate Routines -
3597                                 ;               BLDBUF - build a data buffer for transmit
3598                                 ;               GETXNX - update XRGNXT
3599                                 ;               REMAP  - used to remap memory so that the transmit ring may be
3600                                 ;                        accessed
3601                                 ;               RETMEM - used to return the mapping of memory to its original
3602                                 ;                        state
3603                                 ;
3604                                 ; Register usage - R1 holds address of destination address
3605                                 ;                  R2 is a pointer for the loop direct header info
```

```
                                        ;               R3 holds the frame length
                                        ;               R4 holds address of next ring entry data buffer
                                        ;
                                        ;--+

032040                                  BLDLD::
032040                                          P$POP   R1                      ; put address of dest. address in R1
032042                                          CALL    REMAP   @OTRING         ; allow access to transmit ring
032054  013704  002076'                        mov     xrgnxt,R4               ; move next frame address to R4
032060  032764  100000  000004                bit     #own,4(R4)              ; check ownership bit
032066  001075                                  bne     40$                     ; if don't own, bookkeeping error.
032070  016404  000010                          mov     10(R4),R4               ; point R4 to data block
032074  005064  000006                          clr     sourcc(R4)              ; leave blank space for source address
032100  005064  000010                          clr     sourcc+2(R4)            ;   six bytes worth
032104  005064  000012                          clr     sourcc+4(R4)
032110  013764  003034'  000014                mov     prot00,protot(R4)       ; move protocall type into header
032116  012702  003260'                        mov     @LOPDIR,R2              ; move loopdirect format header loc. to R2
032122  012264  000016                          mov     (R2)+,ldskip(R4)        ;  skip count
032126  011264  000020                          mov     (R2),ldfct1(R4)         ;  function code (forward)
032132  013764  002244'  000022                mov     PHYADR,ldadr1(R4)       ;  local node address
032140  013764  002246'  000024                mov     PHYADR+2,ldadr1+2(R4)   ;   six bytes
032146  013764  002250'  000026                mov     PHYADR+4,ldadr1+4(R4)   ;
032154  016264  000010  000030                mov     10(R2),ldfct2(R4)       ;  function code (reply)
032162  013764  002244'  000032                mov     PHYADR,ldadr2(r4)       ;  local node address
032170  013764  002246'  000034                mov     PHYADR+2,ldadr2+2(R4)   ;   six bytes
032176  013764  002250'  000036                mov     PHYADR+4,ldadr2+4(R4)   ;
032204                                          CALL    MOVEXT  @ONTAB,R1,@OTRING,R4,#3 ; move dest. addr. into frame
032232                                          CALL    BLDBUF R4,#ldata        ; build data buffer
032246                                          CALL    GETXNX  @XRGNXT         ; update pointer to next ring entry
032260  000405                                  br      60$                     ; exit

032262                                  40$:    errsf   18,emsg10,err1          ; transmit ring bookkeeping error
032272  000400                                  br      60$                     ; exit

032274                                  60$:    CALL    RETMEM                  ; remap memory to original
032302                                          RETURN
```

```
                                .sbttl  BLDFAS  Build frame for full assist transmission.

                                ;--+
                                ; Functional Description:
                                ;               This subroutine builds full assist frames for transmission
                                ;               from the DELUA/DEUNA.  A full assist is a loop through two
                                ;               nodes: the target and assist nodes.  The target node is the
                                ;               node that is being tested and the assist node is the node
                                ;               that is helping with the transmission to and the reception
                                ;               from the target node.  The full assist frame is sent from the
                                ;               NIE node to the assist node, which sends it to the target node,
                                ;               which sends it back to the assist node, which, finally
                                ;               returns it to the NIE node.
                                ;
                                ; Inputs -
                                ;               P1 - pointer to the ethernet address of the target node
                                ;               P2 - pointer to the ethernet address of the assist node
                                ;
                                ;       Implicit -
                                ;               P$SIZE - contains the size of the message buffer data
                                ;               XRGNXT - points to the next available ring entry
                                ;               PHYADR  - holds the current local node address
                                ;
                                ; Outputs - Implicit -
                                ;               A full assist loopback frame has been built in the buffer
                                ;               pointed to by the transmit ring entry pointed to by XRGNXT
                                ;
                                ; Calling Procedure - CALL BLDFAS P1
                                ;
                                ; Side Effects - XRGNXT is updated to point to the next transmit ring entry
                                ;
                                ; Subordinate Routines -
                                ;               BLDBUF - fills frame to be transmitted with data
                                ;               GETXNX - update current transmit ring pointer
                                ;               REMAP  - used to remap memory so that the transmit ring may be
                                ;                        accessed
                                ;               RETMEM - used to return the mapping of memory to its original
                                ;                        state
                                ;
                                ; Register usage - R1 holds address of target node address
                                ;                  R2 holds address of assist node address
                                ;                  R3 holds the frame length
                                ;                  R4 holds address of next ring entry data buffer
                                ;
                                ;--+

C32304                          BLDFAS::
032304                                  P$POP   R1,R2                   ; put address of target address into R1
                                                                        ;  and address of assist address into R2

032310                                  CALL    REMAP   #OTRING         ; enable access to transmit memory

032322  013703  002076'                 mov     xrgnxt,R3               ; move next frame address to R3
032326  032763  100000  000004          bit     #own,4(R3)              ; check ownership bit
032334  001144                          bne     40$                     ; if don't own, bookkeeping error,
032336  016304  000010                  mov     10(R3),R4               ; point R4 to buffer
```

```
 3701                                   ;--+
 3702                                   ;       DELUA/DEUNA will add in source address.
 3703                                   ;--+
 3704
 3705 032342  005064  000006                    clr     sourcc(R4)              ; leave blank space for source address
 3706 032346  005064  000010                    clr     sourcc+2(R4)            ;   six bytes worth
 3707 032352  005064  000012                    clr     sourcc+4(R4)
 3708
 3709                                   ;--+
 3710                                   ;       Add protocol type, skip count, and function code fields to frame
 3711                                   ;--+
 3712
 3713 032356  013764  003034' 000014            mov     prot00,protot(R4)       ; move protocall type into header
 3714 032364  012764  000000  000016            mov     #0,faskip(R4)           ;  skip count
 3715 032372  012764  000002  000020            mov     #2,fafct1(R4)           ;  function code (forward)
 3716 032400  012764  000002  000030            mov     #2,fafct2(R4)           ;  function code (forward)
 3717 032406  012764  000002  000040            mov     #2,fafct3(R4)           ;  function code (forward)
 3718 032414  012764  000001  000050            mov     #1,fafct4(R4)           ;  function code (reply)
 3719
 3720                                   ;--+
 3721                                   ;       Our physical address is the third forward address.  This completes
 3722                                   ;       the loop.
 3723                                   ;--+
 3724
 3725 032422  013764  002244' 000042            mov     phyadr,faadr3(R4)       ;  local node address
 3726 032430  013764  002246' 000044            mov     phyadr+2,faadr3+2(R4)   ;   six bytes
 3727 032436  013764  002250' 000046            mov     phyadr+4,faadr3+4(R4)   ;
 3728
 3729                                   ;--+
 3730                                   ;       Our physical address is also the reply address.  This will allow
 3731                                   ;       the DELUA/DEUNA to recognize the reply to the loop message
 3732                                   ;--+
 3733
 3734 032444  013764  002244' 000052            mov     phyadr,faadr4(R4)       ;  local node address
 3735 032452  013764  002246' 000054            mov     phyadr+2,faadr4+2(R4)   ;   six bytes
 3736 032460  013764  002250' 000056            mov     phyadr+4,faadr4+4(R4)   ;
 3737
 3738                                   ;--+
 3739                                   ;       Now add all portions of the frame that come from the node table.
 3740                                   ;       Namely, destination, target node, and assist node
 3741                                   ; -+
 3742
 3743 032466                                    CALL    MOVEXT  #ONTAB,R2,#OTRING,R4,#3 ; move in dest. addr.
 3744 032514  062704  000022                    ADD     #FAADR1,R4              ; point R4 to first forward addr.
 3745 032520                                    CALL    MOVEXT  #ONTAB,R1,#OTRING,R4,#3 ; move in first forward addr.
 3746 032546  062704  000010                    ADD     #FAADR2-FAADR1,R4       ; point R4 to second forward addr.
 3747 032552                                    CALL    MOVEXT  #ONTAB,R2,#OTRING,R4,#3 ; move in second forward addr.
 3748
 3749 032600                                    CALL    REMAP   #OTRING         ; allow access to transmit ring
 3750 032612  016304  000010                    MOV     10(R3),R4               ; point R4 back to beginning buffer
 3751 032616                                    CALL    BLDBUF R4,#FDATA2       ; fill data field
 3752 032632                                    CALL    GETXNX  #XRGNXT         ; update pointer to next ring entry
 3753 032644  000405                            br      50$                     ; exit
 3754
 3755 032646                            40$:    errsf   19,emsg10,err1          ; transmit ring bookkeeping error
 3756 032656  000400                            br      50$                     ; exit
 3757
```

```
3758 032660                          50$:    CALL    RETMEM                  ; remap memory to or ginal
3759 032666                                  RETURN
3760
```

```
                                .sbttl  BLDREQ  Build Request ID Frames for transmit.

                                ;--+
                                ; Functional Description:
                                ;               This subroutine builds Request ID frames for transmission
                                ;               from the DELUA/DEUNA.  Source address, destination address,
                                ;               protocall type, sequence number and Request ID
                                ;               header info are built into the buffer.
                                ;
                                ; Inputs - Implicit -
                                ;               The destination address is contained in ADRBUF.
                                ;
                                ; Outputs - Implicit -
                                ;               The buffer pointed to by the transmit ring entry pointed
                                ;               to by XRGNXT contains a request ID message.
                                ;
                                ; Calling Procedure - CALL BLDREQ
                                ;
                                ; Side Effects -
                                ;               XRGNXT - updated to point to next transmit ring entry
                                ;
                                ; Subordinate Routines -
                                ;               GETXNX - updates XRGNXT
                                ;               REMAP  - used to remap memory so that the transmit ring may be
                                ;                        accessed
                                ;               RETMEM - used to return the mapping of memory to its original
                                ;                        state
                                ;
                                ; Register Usage -
                                ;               R2 - is a pointer for Request ID header info.
                                ;               R4 - holds address of next ring entry data buffer.
                                ;
                                ;--+

032670                          BLDREQ::
032670                                  CALL    REMAP   #OTRING         ; allow access to transmit ring
032702  013704  002076'                mov     XRGNXT,R4               ; move next frame address to R4
032706  032764  100000  000004        bit     #own,4(R4)              ; check ownership bit
032714  001050                        bne     40$                     ; if don't own, bookkeeping error
032716  016404  000010                mov     10(R4),R4               ; point R4 to data block
032722  012737  000100  003126'        mov     #100,buflen             ; move buffer size to buflen
032730  005064  000006                clr     sourcc(R4)              ; leave blank space for source addr.
032734  005064  000010                clr     sourcc+2(R4)            ;  six bytes worth
032740  005064  000012                clr     sourcc+4(R4)
032744  013764  003036'  000014        mov     prot02,protot(R4)       ; move protocall type into header
032752  012702  003252'                mov     #REQID,R2               ; move Request ID header loc. to R2
032756  012264  000016                mov     (R2)+,header(R4)        ;  byte count
032762  012264  000020                mov     (R2)+,header+2(R4)      ;  function code (request ID)
032766  011264  000022                mov     (R2),header+4(R4)       ;  reciept no.
032772                                  CALL    MOVEXT  #ONTAB,#ADRBUF,#OTRING,R4,#3    ; set up destination addr. of frame
033022                                  CALL    GETXNX #XRGNXT          ; update pointer to next ring entry
033034  000404                        br      50$                     ; exit
033036                          40$:    errsf   20,emsg10,err1          ; transmit ring bookkeeping error
033046                          50$:    CALL    RETMEM                  ; return memory mapping to its origin
033054                                  RETURN
```

```
                                .sbttl  GET?NX  Get next transmit or recieve ring entry

                                ;--+
                                ; Functional Description
                                ;               This subroutine gets the next transmit or recieve ring
                                ;               entry.  It is entered at seperate points depending on
                                ;               which ring is being used.
                                ;
                                ; Inputs -      P1 - The address of the ring pointer to be updated.
                                ;
                                ; Outputs -     The ring pointer is updated to point to the next available
                                ;                 entry.
                                ;
                                ; Calling procedure - CALL  GETXNX  @P1          ; for transmit updates
                                ;                     CALL  GETRNX  @P1          ; for recieve updates
                                ;
                                ; Side effects - None
                                ;
                                ; Subordinate Routines - none
                                ;
                                ; Register Usage - R1 points to the first entry in the ring
                                ;                  R2 points to the last entry in the ring
                                ;                  R3 is the address of the ring pointer to be updated
                                ;
                                ;--+

033056                          GETRNX::
033056  013701  002070'                 mov     rrgsrt,R1               ; move first ring entry to R1
033062  013702  002104'                 mov     rrglst,R2               ; move last ring entry to R2
033066  000404                          br      GETCOM                  ; go to common code
033070                          GETXNX::
033070  013701  002066'                 mov     xrgsrt,R1               ; move first ring entry to R1
033074  013702  002102'                 mov     xrglst,R2               ; move last ring entry to R2
033100                          GETCOM: P$POP   R3                      ; get address of ring pointer in R3
033102  021302                          cmp     (R3),R2                 ; see if pointer points to last ring
033104  001403                          beq     15$                     ;  if yes, branch
033106  062713  000012                  add     #10.,(R3)               ;  else, add entry length to pointer
033112  000401                          br      25$                     ; exit
033114  010113                  15$:    mov     R1,(R3)                 ; point pointer to first entry in ring
033116                          25$:    RETURN


                                .sbttl  BLDBUF  Build Message Buffers

                                ;--+
                                ; Functional Description
                                ;               This routine fills a transmit buffer with data.  It will load
                                ;               bytes into the buffer to pad the data field out to P$SIZE bytes.
                                ;
                                ; Inputs -
                                ;               P1 - address of the beginning of a transmit buffer
                                ;               P2 - number of bytes already loaded into data field of
                                ;                    the transmit buffer to be worked on
                                ;
                                ;       Implicit -
                                ;               P$SIZE contains the size the buffer is to be
```

```
                                ;               P$TYPE contains the message type
                                ;
                                ; Outputs - Implicit -
                                ;               Buffer starting at location P1 contains a message P$SIZE bytes
                                ;               long using the message type specified by P$TYPE.
                                ;
                                ; Calling proceedure: Call BLDBUF P1,P2
                                ;
                                ; Side effects -
                                ;               XFER - gets loaded with the number of bytes that will be
                                ;                      transferred -- used by summary routine
                                ;               BUFLEN - loaded with the length of the transmit buffer
                                ;               CMPBUF - address of the data field of the transmit buffer to
                                ;                        be used in data compare routine
                                ;
                                ; Subordinate Routines - none
                                ;
                                ; Register usage - R1 - scratch
                                ;                  R2 - (message type X 2), used as offset for pointers
                                ;                  R3 - points to the next byte of the buffer under construction
                                ;                  R4 - points to the last byte of the buffer under construction
                                ;
                                ;--+
                                BLDBUF::
033120                                  P$POP   R3,R1                   ; put buffer address into R3
033120                                                                  ; and number of bytes in buffer in R1
033124                                  CALL    REMAP   #OTRING         ; allow access to transmit ring
                                ;--+
                                ;       set up the boundaries of the data transfer
                                ;--+
033136  062703  000016                  add     #16,R3                  ; point R3 past header info
033142  013704  001172'                 mov     P$SIZE,R4               ; put size into R4
033146  060304                          add     R3,R4                   ; make R4 = last byte of data buffer
033150  010337  003130'                 MOV     R3,CMPBUF               ; store pointer to data field for data
                                                                        ; compare
033154  060103                          add     R1,R3                   ; point R3 past data already in buffer
                                ;--+
                                ;       Set up transfer size and buffer length
                                ;--+
033156  012737  000016  003126'         MOV     #16,BUFLEN              ; buffer length = header ...
033164  063737  001172' 003126'         ADD     P$SIZE,BUFLEN           ; ... + data field
033172  013737  003126' 003120'         MOV     BUFLEN,XFER             ; transfer size for summary
                                ;--+
                                ;       Set up pointer to message to fill with
                                ;--+
033200  013702  001170'                 mov     P$TYPE,R2               ; put message type into R2
033204  006302                          asl     R2                      ; multiply by 2
033206  016201  001450'                 mov     MSGAD(R2),R1            ; point R1 to first byte of stored message
033212  005037  003032'                 clr     COUNT                   ; clear byte counter
033216  005237  003032'         10$:    inc     COUNT                   ; count no. of bytes copied
```

```
 3933 033222  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
 3941 033250                                  CALL    RETMEM                  ; restore memory mapping
 3942 033256                                  RETURN                          ;  else, return
 3943
```

```
                                .sbttl  DATCMP  Compare data buffers

                                ;--+
                                ; Functional Description
                                ;               This subroutine compares two data buffers byte by byte.
                                ;               If comparison errors occured, location, expected data
                                ;               and recieved data are printed out for the first five
                                ;               errors.  The total number of errors is also printed.
                                ;
                                ; Inputs -      P1 - The size (in bytes) of the buffer to be compared.
                                ;               P2 - The address of buffer to compare other buffer against.
                                ;               P3 - The address of the second buffer.
                                ;
                                ; Outputs -     P4 - The number of comparison errors.
                                ;
                                ; Calling Procedure - CALL DATCMP P1,P2,P3
                                ;                      P$POP P4
                                ;
                                ; Subordinate Routines - none
                                ;
                                ; Side effects - none
                                ;
                                ; Register Usage - R1 - number of words to compare
                                ;                  R2 - pointer to data in transmit buffer
                                ;                  R3 - pointer to data in receive buffer
                                ;                  R4 - contains the word offset (words from beginning of data)
                                ;
                                ;--+

033260                          DATCMP::
033260                                  P$POP   R1,R2,R3                ; put compare size in R1
                                                                        ;  R2 gets transmit data address
                                                                        ;  R3 gets receive data address
033266  005037  003110'                 CLR     TEMP                    ; init. return value
033272  105737  001303'                 TSTB    P$NCMP                  ; has no compare been selected?
033276  001402                          BEQ     1$                      ; branch if yes
033300  000137  033562'                 JMP     30$                     ; leave
033304                          1$:
033304  012704  177777                  MOV     #-1,R4                  ; initialize byte offset
033310  042701  000001                  BIC     #BIT0,R1                ; make even number of word compares

033314  005204                  10$:    inc     R4                      ; increment offset counter
033316                                  CALL    CMPEXT  #OTRING,R2,#ORRING,R3,#1 ; compare a word
033344                                  P$POP   R0                      ; get compare indicator

033346  001462                          beq     20$                     ;  if same, branch
033350  005237  003110'                 inc     temp                    ; increment error counter
033354  023727  003110' 000001          cmp     temp,#1                 ; is this the first error?
033362  003010                          bgt     15$                     ;    NO, skip header
033364                                  PRINTX  #CMPERH                 ;    YES, print a header

033404  022737  000005  003110' 15$:    cmp     #5,temp                 ; if more than 5 errors,
033412  002440                          blt     20$                     ; don't print message
033414                                  CALL    REMAP   #OTRING         ; allow access to transmit buffer
033426                                  printx  #cmper1,R4,(R2)         ; print expected word
033452                                  CALL    REMAP   #ORRING         ; allow access to receive buffer
```

```
 4002 033464                                PRINTX  @CMPER2,(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
 4012 033562                         30$:   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 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 exit.  An error is reported if the end of the table
 4026                                ;               is reached.
 4027                                ;
 4028                                ; Inputs -      P1 - The number of nodes to update (1 or 2).
 4029                                ;               P2 - The address of the first node address.
 4030                                ;               P3 - The address of the second node address if P1 = 2 or
 4031                                ;                    blank if P1 = 1.
 4032                                ;               P4 - page register value for accessing the structure that
 4033                                ;                    contains the node addresses.
 4034                                ;
 4035                                ;       Implicits -
 4036                                ;               The summary counters: S.NREC, S.REC, S.LEN, S.COMP, S.BYTE,
 4037                                ;               and S.XFER
 4038                                ;
 4039                                ; Outputs -     The summary table is updated.
 4040                                ;
 4041                                ; Calling procedure - CALL WRITES P1,P2(,P3)
 4042                                ;
 4043                                ; Side effects - The summary counters are cleared.
 4044                                ;
 4045                                ; Subordinate Routines -
 4046                                ;               CMPTWO - routine to compare two strings
 4047                                ;
 4048                                ; Register Usage -
 4049                                ;               R1 points to the current location in the summary table.
 4050                                ;               R2 points to the node to be updated's address.
 4051                                ;               R3 is scratch
 4052                                ;               R4 holds the second node to be updated address.
 4053                                ;
 4054                                ;--+
 4055
 4056 033570                         WRITES::
 4057 033570                                P$POP   temp                 ; see how many nodes to write
 4058 033574  023727  003110' 000001        cmp     temp,#1              ;  if only one, get address
```

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

```
 4112
 4113                                   .sbttl  BINDEC  Convert a 32 bit binary number to decimal
 4114
 4115                                   ; -+
 4116                                   ; Functional Description:
 4117                                   ;               This subroutine converts a 32 bit binary number to
 4118                                   ;               a decimal number represented as an asciz string.
 4119                                   ;
 4120                                   ; Inputs -      P1 - The address of the first word of binary data
 4121                                   ;                    bits 0-15.  The second word, bits 16-31, is
 4122                                   ;                    expected to immediately follow the first word.
 4123                                   ;
 4124                                   ; Outputs -     The ascii string will be located starting at DECSTR
 4125                                   ;
 4126                                   ; Calling Procedure: CALL BINDEC P1
 4127                                   ;
 4128                                   ; Side effects - none
 4129                                   ;
 4130                                   ; Subordinate Routines - none
 4131                                   ;
 4132                                   ; Register Usage - R1 points to bits 0-15 of binary data
 4133                                   ;                  R2 points to bits 16-31 of binary data
 4134                                   ;                  R3 points to the output string
 4135                                   ;                  R4 points to the powers of 10 table
 4136                                   ;
 4137                                   ;--+
 4138
 4139 034122                           BINDEC::
 4140 034122                                   P$POP   R1                      ; put address of binary word into R1
 4141 034124  010546                           mov     R5,-(SP)
 4142 034126  012137  003112'                  mov     (R1)+,temp1             ; put low word in TEMP1
 4143 034132  011137  003114'                  mov     (R1),temp2              ; put high word in TEMP2
 4144 034136  012703  034324'                  mov     #DECSTR,R3              ; put address of ouput string into R3
 4145 034142  012704  034254'                  mov     #TENPWR,R4              ; address of ten power table
 4146 034146  012705  034256'                  mov     #TENPWR+2,R5
 4147 034152  012737  000012  034242'         mov     #10.,4$
 4148 034160  005037  034340'          1$:     clr     part                    ; clear partial counter
 4149 034164  161437  003112'          2$:     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
 4163 034244  001345                           bne     1$                      ; if no, branch
 4164 034246  105023                           clrb    (R3)+                   ; if yes, terminate with zero
 4165 034250  012605                           mov     (SP)+,R5
 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
```

```
                                .SBTTL  COMMAND LINE TRAVERSE ROUTINES

                                ;++
                                ;       P$TRV SUBROUTINE
                                ;
                                ;PARSE THE COMMAND LINE SUBROUTINE
                                ;TAKE ACTIONS (VIA ACTION TREE) AS PARSING LINE
                                ;PARSING DIRECTIONS FROM "CLI PARSING NODES"
                                ;   REGS USED:
                                ;
                                ;       R1,R5=SCRATCH                           P$NUM=NUMERIC CODE FROM DATA
                                ;       R2=ACTION CODE PARAMETER FROM TREE
                                ;       R3=PARSE TREE POINTER
                                ;       R4=INPUT STRING POINTER
                                ; CALLING SEQUENCE:
                                ;       JSR     PC,P$TRV
                                ;--

034342                          P$TRV::
034342  013704  001260'                MOV     P$BUFA,R4
034346  013703  001262'                MOV     P$TREE,R3
034352  121327  000003          P$TR5:  CMPB    (R3),#3                 ;SEE IF ONE OF FIRST THREE SPECIAL CODES
034356  003405                        BLE     5$                      ;IF YES, DON'T CHECK INPUT STRING
034360  105714                        TSTB    (R4)                    ;SEE IF ANY CHARS LEFT IN INPUT STRING
034362  001441                        BEQ     P$EXIT                  ;BR IF NO
034364  121327  000013                CMPB    (R3),#11.               ;SEE IF SPECIAL CLI CHAR CODE OR ASCII
034370  003023                        BGT     20$                     ;BR IF REGULAR ASCII CHAR.
034372  111301          5$:     MOVB    (R3),R1                 ;GET SPECIAL CHAR CODE INTO R5
034374  006301                        ASL     R1
034376  016101  034412'                MOV     10$(R1),R1              ;BUILD TRAVERSE ROUTINE ADDRESS
034402  062701  034412'                ADD     #10$,R1
034406  004711                        JSR     PC,(R1)                 ;JSR TO SPECIAL CLI TRAVERSE ROUTINE
034410  000760                        BR      P$TR5                   ;GO SEE IF MORE OF STRING LEFT

034412  000114          10$:    .WORD   TRVERR-10$              ;TRAVERSE TABLE FOR "CLI FUNCTIONS"
034414  000134                        .WORD   TRVEXI-10$              ;1
034416  000152                        .WORD   TRVBR-10$               ;2
034420  000162                        .WORD   TRVBIF-10$              ;3
034422  000204                        .WORD   TRVSPA-10$              ;4
034424  000270                        .WORD   TRVNUM-10$              ;5
034426  000612                        .WORD   TRVALP-10$              ;6

034430  000000                        .WORD   0                       ; *** was .WORD TRVALN-10$ ***
034432  000270                        .WORD   TRVOCT-10$              ;8
034434  000256                        .WORD   TRVDEC-10$              ;9
034436  000656                        .WORD   TRVSTR-10$              ;10

                                ;NOT A SPECIAL CODE

034440  121314          20$:    CMPB    (R3),(R4)               ;SEE IF FIRST CHAR OF STRING IS A MATCH
034442  001403                        BEQ     22$                     ;BR IF A MATCH
034444  004737  034510'                JSR     PC,TRVBRC               ;IF NOT A MATCH, GO TAKE MISS BRANCH
034450  000740                        BR      P$TR5                   ; THEN GO BACK PT'G TO MISS NODE
034452  005204          22$:    INC     R4                      ;IF A MATCH, INCR. CHAR POINTER
034454  004737  034470'                JSR     PC,TRVACT               ; GO DO ACTION DEFINED BY
```

```
4250 034460  062703  000004              ADD     #4,R3                   ; ACTION CODE IN CLI NODE, THEN
4251                                                                      ; ADJUST PTR TO NEXT CLI NODE
4252 034464  000732                      BR      P$TR5
4253
4254 034466  000207              P$EXIT: RTS     PC                      ;RETURN FROM PARSER
4255
4256                             ;-------------------------------------------------------------------------  --
4257
4258                             ;GOTO USER ACTION ROUTINE
4259 034470  116302  000001      TRVACT: MOVB    1(R3),R2                ;GET ACTION CODE FROM CLI NODE
4260 034474  042702  177400              BIC     #177400;R2              ;CLEAR ANY SIGN EXTENSION
4261 034500  013701  001264'              MOV     P$ACT,R1                ;GET ADDRESS OF CLI ACTION ROUTINE
4262 034504  004711                      JSR     PC,(R1)                 ;GO DO ACTION DEFINED BY CODE
4263 034506  000207                      RTS     PC                      ;RETURN TO CALLING CODE
4264
4265                             ;TAKE BRANCH IN TREE
4266 034510  016301  000002      TRVBRC: MOV     2(R3),R1                ;GET BRANCH DISPLACEMENT FROM TREE
4267 034514  060103                      ADD     R1,R3                   ; AND POINT R3 TO THE "MISS" NODE
4268 034516  000207                      RTS     PC                      ; RETURN TO P$TRV
4269
4270                             ;NO BRANCH TAKEN
4271 034520  062703  000004      TRVNOB: ADD     #4,R3                   ;THINGS OK, UPDATE R3 TO POINT TO NEXT
4272 034524  000207                      RTS     PC                      ; NODE AND RETURN TO P$TRV
4273
4274                             ;-------------------------------------------------------------------------
4275                             ;ERROR HANDLING
4276 034526  004737  034470'    TRVERR: JSR     PC,TRVACT               ;TAKE ERROR ACTION
4277 034532  112737  177777  001301'        MOVB    #-1,P$GDBD              ;SET ERROR RETURN FLAG
4278 034540  005726                      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                             ;EXIT ACTION CODE
4282 034546  004737  034470'    TRVEXI: JSR     PC,TRVACT               ;TAKE EXIT ACTION
4283 034552  105037  001301'              CLRB    P$GDBD                  ;SET GOOD/BAD FLAG TO "SUCCESS (0)"
4284 034556  005726                      TST     (SP)+                   ;GET RID OF "JSR PUSH TO TRVEXI"
4285 034560  000137  034466'              JMP     P$EXIT                  ;RETURN DIRECT TO EXIT OF P$TRV ROUTINE
4286
4287                             ;BRANCH ACTION CODE
4288 034564  004737  034470'    TRVBR:  JSR     PC,TRVACT               ;GO TAKE BRANCH ACTION
4289 034570  000137  034510'              JMP     TRVBRC
4290
4291                             ;BRANCH-IF ACTION CODE
4292 034574  004737  034470'    TRVBIF: JSR     PC,TRVACT
4293 034600  105737  001301'              TSTB    P$GDBD                  ;SEE IF P$GDBD SET OR CLEARED BY ACTION
4294 034604  001402                      BEQ     1$                      ;IF CLEAR FALL THRU TO NEXT NODE
4295 034606  000137  034510'              JMP     TRVBRC                  ;ELSE TAKE THE "MISS" BRANCH
4296 034612  000137  034520'      1$:     JMP     TRVNOB                  ;JUST UPDATE TO NEXT NODE IF THINGS OK
4297
4298                             ;SPACE ACTION CODE
4299 034616  005001              TRVSPA: CLR     R1                      ;CLEAR "SPACE OR TAB FOUND" FLAG
4300 034620  121427  000011      1$:     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     TRVNOB                  ;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$NUM                   ;SHIFT CURRENT VALUE TO MAKE ROOM
 4358 035066  103440                          BCS     7$                      ;ERROR IF NUMBER TOO BIG
 4359 035070  013737  001270' 001266'         MOV     P$NUM,P$CNT             ;SAVE FOR LATER IN CASE DECIMAL RADIX
 4360 035076  006337  001270'                 ASL     P$NUM
 4361 035102  103432                          BCS     7$                      ;ERROR IF NUMBER TOO BIG
 4362 035104  006337  001270'                 ASL     P$NUM
 4363 035110  103427                          BCS     7$                      ;ERROR IF NUMBER TOO BIG
```

```
 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
 4367 035130  103417                          BCS     7$                      ;ERROR IF NUMBER TOO BIG
 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                      ;RESTORE R5
 4376 035160  004737  034470'                JSR     PC,TRVACT               ;SINCE NUMERIC FOUND, GO TAKE ACTION
 4377 035164  000137  034520'                JMP     TRVNOB                  ;GO POINT R3 TO NEXT NODE
 4378
 4379 035170                          7$:     PRINTF  #CLINBG                 ;PRINT NUMBER TOO BIG ERROR
 4380 035210  112737  177777  001301'        MOVB    #-1,P$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     TRVNOB                  ;THEN UPDATE R3 TO NEXT NODE -NO BRANCH
 4397 035264  000137  034510'          3$:   JMP     TRVBRC                  ;NONE FOUND, TAKE MISS BRANCH
 4398
 4399
 4400
 4401 035270                          TRVSTR: PUSH    R5                      ;SAVE R5
 4402 035272  010401                          MOV     R4,R1                   ;POINT R1 TO CMD STRING
 4403 035274  010305                          MOV     R3,R5
 4404 035276  062705  000006                  ADD     #6,R5                   ;POUNT R5 TO MATCH STRING FROM CLI NODE
 4405 035302  005037  001266'                CLR     P$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 TRVNOB 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$MERR
 4449                              ;       ----------------+----------+--------------+------   ---- ----+- -----
 4450                              ;       ILLEGAL CHAR.   !   -1     ! ILL. CHAR.   !                  ! N/A
 4451                              ;       ADR./ASSIST     !    0     ! END OF LINE  !     CASIST       ! N/A
 4452                              ;       ADR./TARGET     !          !              !                  !
 4453                              ;       ADR./           !    0     ! END OF LINE  !     CTARGT       ! N/A
 4454                              ;       ADR.            !          !              !                  !
 4455                              ;       ADR./CHAR. OR   !          !              !                  !
 4456                              ;       "OPR SEL/CHAR.  !          !              !                  !
 4457                              ;         OTHER THAN "A"!   -1     ! /            !     CTARGT       ! N/A
 4458                              ;         "T" OR BLANK  !          !              !                  !
 4459                              ;       ""              !    0     ! CHAR. AFTER " !                  ! -1
 4460                              ;       "OPR SEL"       !    0     ! CHAR. AFTER " !     OPRSEL       !  0
 4461                              ;
 4462                              ;       CALLING PROCEDURE -  JSR  PC,TRVADR
 4463                              ;       REGISTER USAGE -     R1 IS USED AS A COUNTER TO REPORT ERROR MESSAGES
 4464                              ;                               IF NULL STRINGS ARE ENTERED.
 4465                              ;                            R4 POINTS TO THE NEXT CHAR. IN THE COMMAND LINE
 4466                              ;
 4467                              ;--+
 4468
 4469 035366  005001          TRVADR: CLR     R1                      ;CLEAR HEX DIGIT FOUND FLAG
 4470 035370  121427  000000  1$:     CMPB    (R4),#0                 ;SEE IF NUL CHAR.
 4471 035374  001435                  BEQ     20$                     ; IF YES, RETURN
 4472 035376  121427  000040          CMPB    (R4),#40                ;SEE IF ILLEGAL CHARACTER
 4473 035402  002426                  BLT     10$                     ;IF YES; BRANCH TO ERROR ROUTINE
 4474 035404  001002                  BNE     4$                      ;branch if not a space
 4475 035406  005204                  INC     R4                      ; skip space
 4476 035410  000767                  BR      1$                      ; check next character
 4477 035412  121427  000042  4$:     CMPB    (R4),#42                ;SEE IF CHAR. IS A '"'
```

```
 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          6$:     CMPB    (R4),#57                ;SEE IF CHAR. IS A "/"
 4484 035442  001420                          BEQ     30$                     ;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$:    MOV     #CASIST,CFLAG           ;SET ASSIST FLAG
 4513 035572  000422                          BR      50$
 4514 035574  005701                  40$:    TST     R1                      ;SEE IF ANY CHARACTERS TYPED
 4515 035576  001404                          BEQ     45$                     ;IF NO, BRANCH TO 45$
 4516 035600  012737  000006  002024'         MOV     #OPRSEL,CFLAG           ;SET OPERATOR SELECTED FLAG
 4517 035606  000414                          BR      50$                     ;RETURN
 4518 035610                          45$:    PRINTF  #NULSTR                 ;PRINT NULL STRING ERROR MESSAGE
 4519 035630  112737  177777  001304'         MOVB    #-1,P$MERR              ;SET OPER. SELECTED MSG. ERROR FLAG
 4520 035636  005204                          INC     R4                      ;MOVE CMD. LINE POINTER TO NEXT CHAR.
 4521 035640  000207                  50$:    RTS     PC                      ;RETURN
 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
```

```
                                ;*****************************************************************************
                                ;       THIS SECTION, WHICH IS OPTIONAL, CONTAINS THE CODE FOR PRINTING
                                ;       STATISTICAL INFORMATION GATHERED BY THE DIAGNOSTIC.  IT IS
                                ;       EXECUTED BY THE OPERATOR COMMAND "PRINT" OR BY THE MACRO CALL
                                ;       "DORPT".  USE THE PRINTS MACRO TO PRINT THE INFORMATION
                                ;       USE FORMAT STATEMENTS AS IN THE PRINTB/PRINTX MACROS.  IT IS
                                ;       THE PROGRAMMER'S RESPONSIBILTY TO DEVISE AND IMPLEMENT THE
                                ;       FORM AND CONTENT OF THE STATISTICS.
                                ;*****************************************************************************

035642  004737  042674'                JSR     PC,ACTSUM
035646                                  EXIT    RPT

                                ;*****************************************************************************
                                ;     INSERT LOCAL STORAGE THAT IS USED ONLY
                                ;     DURING THE REPORT SECTION.
                                ;*****************************************************************************

                                ;*****************************************************************************
                                ;     INSERT MESSAGES THAT ARE USED ONLY
                                ;     DURING THE REPORT SECTION.
                                ;*****************************************************************************

                                        .EVEN

035652                                  ENDRPT
```

```
                                .SBTTL  PROTECTION TABLE

                                ;++
                                ; THIS TABLE IS USED BY THE RUNTIME SERVICES
                                ; TO PROTECT THE LOAD MEDIA.
                                ;--

035654                                  BGNPROT

035654  177777                          -1              ;OFFSET INTO P-TABLE FOR CSR ADDRESS
035656  177777                          -1              ;OFFSET INTO P-TABLE FOR MASSBUS ADDRESS
035660  177777                          -1              ;OFFSET INTO P-TABLE FOR DRIVE NUMBER

035662                                  ENDPROT

                                ;*******************************************************************************
                                ;       INSERT BYTE OFFSET FOR DATA NOTED IN COMMENTS ABOVE.  (OFFSET
                                ;       REFERS TO THE NUMBER OF BYTES FROM THE BEGINNING OF A PTABLE
                                ;       ENTRY TO THE ITEM IN QUESTION.)  IF THE PARTICULAR
                                ;       ITEM DOES NOT APPLY, LEAVE ENTRY AS -1.  WHEN THE RUNTIME
                                ;       SERVICES EXECUTES A GPHARD, IT USES THESE OFFSETS (IF NOT
                                ;       SET TO -1) TO GET THE ITEMS AND COMPARE WITH THOSE SAVED
                                ;       IN THE XXDP+ MONITOR.  IF THE UNIT BEING REQUESTED MATCHES THE
                                ;       LOAD DEVICE, THE RUNTIME SERVICES RETURN AN INCOMPLETE FLAG ON
                                ;       THE GPHARD.
                                ;*******************************************************************************
```

```
 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
 4605                           ;*****************************************************************************
 4606                           ;       THE INITIALIZE CODE IS EXECUTED UNDER FIVE CONDITIONS.  THERE
 4607                           ;       ARE SUPERVISOR EVENT FLAGS THAT ARE USED TO LET THE
 4608                           ;       DIAGNOSTIC KNOW UNDER WHICH CONDITION THE EXECUTION IS TAKING
 4609                           ;       PLACE.  THE EVENT FLAGS ARE READ USING THE "READEF" MACRO.
 4610                           ;       THE CONDITIONS UNDER WHICH THE INIT CODE IS EXECUTED AND THE
 4611                           ;       CORRRESPONDING EVENT FLAGS ARE:
 4612                           ;               START COMMAND           EF.START
 4613                           ;               RESTART COMMAND         EF.RESTART
 4614                           ;               CONTINUE COMMAND        EF.CONTINUE
 4615                           ;               POWERDOWN/POWERUP       EF.PWR
 4616                           ;               NEW PASS                EF.NEW
 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                           ;*****************************************************************************
 4627                           ;--+
 4628                           ; Functional Desciption:
 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 availble
 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
```

```
                                ;
                                ;               8.) Call MEMMAP to format extended memory
                                ;
                                ;               9.) set processor priority to ZERO
                                ;
                                ;               10.) CALL UNAINI to initialize the device we are running on
                                ;
                                ;               11.) print out header information
                                ;
                                ;               12.) setup system clock interrupt service routine address and
                                ;                   vector and enable clock
                                ;
                                ;
                                ; Inputs - none
                                ;
                                ; Outputs - A header message will be printed
                                ;
                                ; Calling Procedure: Invoked by the DRS at either a START, RESTART, or CONTINUE
                                ;
                                ; Side Effects - listed above
                                ;
                                ; Subordinate Routines -
                                ;               UNAINI  - initialize the DELUA/DEUNA
                                ;               FUNCT   - perform an ancillary port command
                                ;               DEVSTOP - stop the DELUA/DEUNA
                                ;
                                ; Register Usage -
                                ;               R2,R3   - scratch
                                ;
                                ;--*
                                INIT:
035662 022737 000020 002024'            CMP     #CEXIT,CFLAG            ;SEE IF EXIT COMMAND TYPED
035670 001004                           BNE     INIT1                   ; IF NO, DO INIT CODE
035672 005037 002024'                   CLR     CFLAG                   ; ELSE, CLEAR EXIT FLAG
035676 000137 037276'                   JMP     INICLN                  ; EXIT INIT CODE
035702                          INIT1:  READEF  #EF.START               ;IF HERE BECAUSE OF "START", DO INIT
035710                                  BCOMPLETE       START
035712                                  READEF  #EF.RESTART             ;IF HERE BECAUSE OF "RESTART", DO SOME INIT
035720                                  BNCOMPLETE      5$
035722 000137 037214'                   JMP     RESTRT
035726                          5$:     READEF  #EF.CONTINUE            ;IF HERE BECAUSE OF "CONTINUE", EXIT
035734                                  BNCOMPLETE      10$
035736 000137 037214'                   JMP     RESTRT
035742                          10$:    READEF  #EF.NEW                 ;IF HERE ON NEW PASS, SKIP SOME INIT
035750                                  BNCOMPLETE      15$
035752 000137 037250'                   JMP     NEW
035756 000137 037276'           15$:    JMP     INICLN                  ;IF DON'T KNOW WHY WE'RE HERE, EXIT
035762                          START:  I$STACK #STACK5,SP              ;SET PARAMETER STACK POINTER
035770                                  MEMORY  FRESIZ                  ;GET FREE MEMORY INFO
035776 013737 002134' 002136'           MOV     FRESIZ,FREMEM           ;SIZE OF FREE MEMORY IN FRESIZ
036004 062737 000002 002136'            ADD     #2,FREMEM               ;START OF FREE MEMORY IN FREMEM
036012 012702 002026'                   MOV     #CLKCSR,R2              ;SETUP R2 AS A PRT. TO CLOCK INFO. BLOCK
036016                                  CLOCK   L,R1                    ;GET LINE CLOCK INFO
036026                                  BNCOMPLETE      20$             ;IF NONE, SEE IF P CLOCK PRESENT
036030 004737 027014'                   JSR     PC,CLKSET               ;SET UP CLOCK INFO TABLE AND VECTOR
036034 012737 000100 002036'            MOV     #LCLKEN,CLKEN           ;SET UP THE ENABLE LINE CLOCK DATA
```

```
4711 036042  000430                          BR      30$
4712 036044                          20$:    CLOCK   P,R1                    ;GET P CLOCK INFO
4713 036054                                  BNCOMPLETE      25$             ;IF NO CLOCK, ERROR
4714 036056  004737  027014'                 JSR     PC,CLKSET               ; ELSE SET UP CLOCK INFO AND VECTOR
4715 036062  062737  000002  002026'        ADD     #2,CLKCSR               ; POINT CLKCSR TO P-CLK COUNT SET REG.
4716 036070  012777  001600  143730         MOV     #PCLKCT,@CLKCSR         ;LOAD CLK SET REG. WITH COUNT VALUE
4717 036076  162737  000002  002026'        SUB     #2,CLKCSR               ;POINT CLKCSR BACK TO P-CLK CSR
4718 036104  012737  000111  002036'        MOV     #PCLKEN,CLKEN           ;SETUP TO ENABLE P-CLK DATA
4719 036112  000404                         BR      30$
4720
4721 036114                          25$:    ERRDF   21,EMSG51,ERR1          ; THERE AIN'T NO CLOCK - DEATH!!
4722
4723 036124                          30$:    GPHARD  #0,R1                   ;GET P-TAB POINTER FOR THIS UNIT
4724 036134                                  BCOMPLETE       35$             ;THIS ONE IS NOT AVAILABLE
4725 036136  000137  037276'                 JMP     INICLN
4726
4727 036142  012137  002126'          35$:    MOV     (R1)+,UNACSR            ;SAVE CSR
4728 036146  012137  002130'                  MOV     (R1)+,UNAVEC            ;SAVE VECTOR
4729 036152  012137  002132'                  MOV     (R1)+,UNAPRI            ;SAVE PRIORITY
4730 036156                                  SETVEC  UNAVEC,#UNAISR,UNAPRI   ;SETUP DELUA/DEUNA INTERRUPT VECTOR
4731 036204  013737  002126'  002106'        MOV     UNACSR,PCSR0            ;PCSR0
4732 036212  013737  002106'  002110'        MOV     PCSR0,PCSR1
4733 036220  062737  000002  002110'        ADD     #2,PCSR1                ;PCSR1
4734 036226  013737  002110'  002112'        MOV     PCSR1,PCSR2
4735 036234  062737  000002  002112'        ADD     #2,PCSR2                ;PCSR2
4736 036242  013737  002112'  002114'        MOV     PCSR2,PCSR3
4737 036250  062737  000002  002114'        ADD     #2,PCSR3                ;PCSR3
4738
4739 036256  013703  002110'                 MOV     PCSR1,R3                ; get address of PCSR1 in R3
4740 036262  011302                          MOV     (R3),R2                 ; move value in PCSR1 into R2
4741 036264  042702  177617                  BIC     #177617,R2              ; isolate device id field of PCSR1
4742                                                                         ; it is bits 4-6
4743 036270  010237  000524'                 MOV     R2,DEVICE               ; move value into R2: 0=DEUNA non-0=DELUA
4744
4745
4746 036274                                  CALL    MEMMAP                  ; setup data structures in extended mem.
4747
4748 036302  005037  002770'                 CLR     S.NREC                  ; CLEAR SUMMARY DATA COUNTERS
4749 036306  005037  002766'                 CLR     S.REC
4750 036312  005037  002772'                 CLR     S.LEN
4751 036316  005037  002774'                 CLR     S.COMP
4752 036322  005037  002776'                 CLR     S.BYTE
4753 036326  005037  003000'                 CLR     S.XFER
4754
4755 036332  013737  002034'  002044'        MOV     CLKHZ,TIMTCK            ;LOAD TICKS/SEC
4756 036340                                  SETVEC  CLKVEC,#CLKINT,CLKBR    ;SETUP CLOCK INTERRUPT VECTOR
4757 036366  013777  002036'  143432         MOV     CLKEN,@CLKCSR           ;SET ENABLE BITS IN THE CLOCK TO START
4758 036374                                  SETPRI  #PRI00                  ;SET PRIORITY=0 TO ALLOW FOR INTERRUPTS
4759 036402                                  CALL    UNAINI                  ;INITIALIZE THE DELUA/DEUNA
4760
4761                                 ;--+
4762                                 ;       Read the devices default physical address.  If successful, print
4763                                 ;       it out, else, tell user of error and proceed.
4764                                 ;--+
4765 036410                                  CALL    FUNCT  #RDDEFA          ;READ DELUA/DEUNA DEFAULT PHYSICAL ADDRESS
4766 036422                                  P$POP   R2                      ;CHECK FOR ERROR
4767 036424  001405                          BEQ     40$
```

```
 4768 036426                                ERRSOFT 22,EMSG52               ; INDICATE ERROR
 4769 036436  000423                        BR      45$                     ; DON'T TRY TO PRINT
 4770 036440                        40$:    CALL    BINHEX #PCBB2,#6,#STRBUF        ;PUT ADDRESS INTO HEX FORMAT
 4771 036462                                PRINTS  #HDMSG1,#STRBUF         ;PRINT ADDRESS
 4772
 4773                                ;---+
 4774                                ;       Read ROM firmware version number.  If successful, print it out,
 4775                                ;       else, tell user of error and proceed
 4776                                ;---+
 4777 036506                        45$:    CALL    FUNCT  #RDSTA           ;READ STATUS TO GET ROM VERSION
 4778 036520                                P$POP   R2                      ;CHECK FOR ERROR
 4779 036522  001405                        BEQ     47$
 4780 036524                                ERRSOFT 23,EMSG53               ; INDICATE ERROR
 4781 036534  000415                        BR      50$                     ; DON'T TRY TO PRINT
 4782
 4783 036536  113702  002152'       47$:    MOVB    PCBB2,R2                ;ONLY WANT LOWEST 6 BITS
 4784 036542  142702  000300                BICB    #300,R2
 4785 036546                                PRINTS  #HDMSG2,R2              ;PRINT ROM VERSION
 4786
 4787                                ;---+
 4788                                ;       Now try to print BOOT select options.  The options can be obtained
 4789                                ;       by reading an internal location of the device.  Unfortunately they
 4790                                ;       are neither at the same address nor the same bits of the associated
 4791                                ;       word.  Some contortions must be gone through to print the info ...
 4792                                ;       ... oh well ...
 4793                                ;---+
 4794 036570                        50$:    PRINTS  #HDMSG3                 ;PRINT MORE HEADER INFO
 4795 036610  012703  002626'               MOV     #UCB20,R3               ;SET UP FUNCTION CONTROL BLOCK
 4796 036614  012723  000002                MOV     #2,(R3)+                ; MOVE 2 BYTES...
 4797 036620  012723  003110'               MOV     #TEMP,(R3)+             ; INTO LOCATION TEMP...
 4798 036624  005023                        CLR     (R3)+                   ; HDBB<17:16>
 4799 036626  005737  000524'               TST     DEVICE                  ; What kind of device is this?
 4800 036632  001404                        BEQ     55$                     ; If zero then DEUNA
 4801 036634  012723  000002                MOV     #2,(R3)+                ; else, DELUA IDBB<15:0>
 4802 036640  012723  000030                MOV     #30,(R3)+               ; IDBB<23:16>
 4803
 4804 036644                        55$:    CALL    FUNCT  #DMPMEM          ;DUMP INTERNAL MEMORY
 4805 036656                                P$POP   R2                      ;CHECK FOR ERROR
 4806 036660  001405                        BEQ     60$                     ; NO ERROR
 4807 036662                                ERRSOFT 24,EMSG18               ; REPORT ERROR AS SOFT ...
 4808 036672  000524                        BR      90$                     ; ... AND SKIP STATUS INFO
 4809
 4810 036674  013703  003110'       60$:    MOV     TEMP,R3                 ;PUT RESULT INTO R3
 4811
 4812                                ;---+
 4813                                ;       For the DELUA, the status bits are 15:13 -- the DEUNA 12:10, so
 4814                                ;       need to shift right if a DELUA
 4815                                ;---+
 4816 036700  005737  000524'               TST     DEVICE                  ; IS DEVICE DEUNA?
 4817 036704  001403                        BEQ     62$                     ; YES, NO SHIFT
 4818 036706  006203                        ASR     R3                      ; SHIFT STATUS ...
 4819 036710  006203                        ASR     R3                      ; ... THREE BITS ...
 4820 036712  006203                        ASR     R3                      ; ... TO THE RIGHT.
 4821
 4822 036714  032703  002000        62$:    BIT     #BIT10,R3               ;DETERMINE STATUS
 4823 036720  001430                        BEQ     65$
 4824 036722  032703  004000                BIT     #BIT11,R3
```

```
 4825 036726  001441                          BEQ     70$
 4826 036730  005737  000524'                 TST     DEVICE                  ; Is this DEUNA?
 4827 036734  001411                          BEQ     63$                     ; YES -- special select for DEUNA
 4828 036736                                  PRINTS  #HDMSG7                 ; else, remote boot not enabled
 4829 036756  000446                          BR      80$                     ;
 4830
 4831 036760                          63$:    PRINTS  #HDMSG4                 ; BIT10!BIT11 = REMOTE AND POWER UP BOOT ENABLED
 4832 037000  000435                          BR      80$
 4833
 4834 037002  032703  004000          65$:    BIT     #BIT11,R3
 4835 037006  001422                          BEQ     75$
 4836 037010                                  PRINTS  #HDMSG6                 ; BIT10 = REMOTE BOOT ENABLED
 4837 037030  000421                          BR      80$
 4838
 4839 037032                          70$:    PRINTS  #HDMSG5                 ; BIT11 = REMOTE BOOT ENABLED WITH ROM
 4840 037052  000410                          BR      80$
 4841
 4842 037054                          75$:    PRINTS  #HDMSG7                 ; REMOTE BOOT NOT ENABLED
 4843
 4844                                 ;--+
 4845                                 ;       Now look at self-test status and print it out
 4846                                 ;--+
 4847 037074  032703  010000          80$:    BIT     #BIT12,R3
 4848 037100  001411                          BEQ     85$
 4849 037102                                  PRINTS  #HDMSG8                 ; BIT12 = SELF TEST ENABLED
 4850 037122  000410                          BR      90$
 4851
 4852 037124                          85$:    PRINTS  #HDMSG9                 ; SELF TEST DISABLED
 4853
 4854 037144  012737  000000  001170' 90$:    MOV     #ALPHA,P$TYPE           ;SET MESSAGE DEFAULT VALUES
 4855 037152  012737  001000  001172'         MOV     #512.,P$SIZE
 4856 037160  012737  000001  001174'         MOV     #1,P$CPYS
 4857
 4858 037166  023737  002034' 002044'         CMP     CLKHZ,TIMTCK            ; THESE WON'T BE EQUAL IF CLOCK ...
 4859 037174  001004                          BNE     95$                     ; ... CLOCK IS WORKING
 4860 037176                                  ERRDF   25,EMSG51,ERR1          ; REPORT ERROR AND ABORT
 4861
 4862 037.. ..                        95$:    CALL    DEVSTOP                 ; stop the DEUNA/DELUA
 4863
 4864 037214  105037  001275'     RESTRT: CLRB    P$BLD
 4865 037220  105037  001276'             CLRB    P$HLP
 4866 037224  105037  001303'             CLRB    P$NCMP
 4867 037230  105037  001306'             CLRB    P$BONC
 4868 037234  105037  001305'             CLRB    P$TEXT
 4869 037240  005037  002040'             CLR     TIMMIN                  ;CLEAR TIME SINCE-START-LOCATIONS
 4870 037244  005037  002042'             CLR     TIMSEC
 4871
 4872 037250  013777  002036' 142550 NEW:     MOV     CLKEN,@CLKCSR           ;SET ENABLE BITS IN THE CLOCK TO START
 4873 037256                                  READEF  #EF.START               ; If here because of start, exit
 4874 037264                                  BCOMPLETE INIEXI                ;
 4875 037266                                  SETPRI  #PRI00                  ; Else, adjust priority level to enable interrupts
 4876 037274  000401                          BR      INIEXI                  ;EXIT
 4877 037276                          INICLN: DOCLN                           ;ABORT PASS
 4878 037300                          INIEXI: EXIT    INIT                    ;EXIT INIT SECTION
 4879
 4881                                 ;*****************************************************************************
 4882                                 ;       INSERT LOCAL STORAGE THAT IS USED ONLY
```

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

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

```
                                .SBTTL  CLEANUP CODING SECTION

                                ;++
                                ; THE CLEANUP CODING SECTION CONTAINS THE CODING THAT IS PERFORMED
                                ; AFTER THE HARDWARE TESTS HAVE BEEN PERFORMED.
                                ;--

037310                                  BGNCLN

                                ;*****************************************************************************
                                ;       INSERT YOUR CLEANUP CODING.  THIS CODING SHOULD
                                ;       RESTORE YOUR TEST-DEVICE TO A NEUTRAL STATE.
                                ;       THIS CODE WILL BE EXECUTED AFTER EACH PASS AND AFTER THE
                                ;       PROGRAM IS INTERRUPTED BY "^C".
                                ;*****************************************************************************

                                ;--+
                                ; Name -                                Clean up code
                                ;
                                ; Functional Desciption:
                                ;               The clean-up code is used to leave the DELUA/DEUNA in a
                                ;               known state.  This will result in the following steps:
                                ;
                                ;               1.) wait one second for all port commands to complete
                                ;
                                ;               2.) Stop the DELUA/DEUNA causing it to transition to the
                                ;                   ready state
                                ;
                                ;               3.) clear the DELUA/DEUNA's multicast address list, and
                                ;
                                ;               4.) if we have got here after the listen command then take
                                ;                   the device out of promiscuous mode
                                ;
                                ; Inputs - none
                                ;
                                ; Outputs - none
                                ;
                                ; Calling Procedure: gets called by the DRS
                                ;
                                ; Side Effects - listed above
                                ;
                                ; Subordinate Routines -
                                ;               DEVSTOP - stop the DELUA/DEUNA
                                ;               FUNCT   - issue an ancillary port command
                                ;
                                ; Register Usage -
                                ;               R2      - function return status
                                ;
                                ;--+

037310                                  SETPRI  #PRI00                  ; Let device and clock interrupt

037316  012737  000062  002046'         MOV     #62,TIMER1              ; Set up for one second loop
037324  005737  002046'         5$:     TST     TIMER1                  ; Have we timed out?
037330  001375                         BNE     5$                      ; No, keep looping
037332  005037  003012'                 CLR     DNIFLG                  ; clear done interrupt flag
```

CLEANUP CODING SECTION

```
 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 03747A                                    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                                           .EVEN
 5012
 5013 037506                                    ENDCLN
```

```
                                .SBTTL  DROP UNIT SECTION

                                ;++
                                ; THE DROP-UNIT SECTION CONTAINS THE CODING THAT CAUSES A DEVICE
                                ; TO NO LONGER BE TESTED.
                                ;-

037510                                  BGNDU

                                ;*******************************************************************************
                                ;       INSERT DROP CODE HERE.  THIS CODE WILL BE EXECUTED AFTER
                                ;       A "DROP" COMMAND OR A "DODU" MACRO EXECUTION.  THE PURPOSE
                                ;       OF THIS CODE IS TO DO ANY NECESSARY HOUSEKEEPING AFTER A
                                ;       UNIT HAS BEEN DROPPED.  THIS SECTION IS OPTIONAL.
                                ;*******************************************************************************

03/510                                  EXIT    DU

                                ;*******************************************************************************
                                ;     INSERT LOCAL STORAGE THAT IS USED ONLY
                                ;     DURING THE DROP-UNIT SECTION.
                                ;*******************************************************************************

                                ;*******************************************************************************
                                ;     INSERT MESSAGES THAT ARE USED ONLY
                                ;     DURING THE DROP-UNIT SECTION.
                                ;*******************************************************************************

                                        .EVEN

037514                                  ENDDU
```

```
 5051                                   .SBTTL  ADD UNIT SECTION
 5052
 5053                                   ;++
 5054                                   ; THE ADD-UNIT SECTION CONTAINS ANY CODE THE PROGRAMMER WISHES
 5055                                   ;  TO BE EXECUTED IN CONJUNCTION WITH THE ADDING OF A UNIT BACK
 5056                                   ;  TO THE TEST CYCLE.
 5057                                   ;--
 5058
 5059 037516                                   BGNAU
 5060
 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                                   ;*****************************************************************************
 5069
 5070 037516                                   EXIT    AU
 5071
 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                                   ;*****************************************************************************
 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 Desciption:
 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                                   ;               The flow of control of the routine is as follows:
 5101                                   ;
 5102                                   ;               REPEAT
 5103                                   ;
 5104                                   ;                       CLEAR all variables associated with command parse
 5105                                   ;
 5106                                   ;                       READ command line typed by user
 5107                                   ;
 5108                                   ;                       PARSE the command line
 5109                                   ;                               (* the parse may result in the execution
 5110                                   ;                                  of certain action routines *)
 5111                                   ;
```

```
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 ^C 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$TRV   - parsing routine
5147                                    ;               EXEHLP  - execute the help command
5148                                    ;               EXEBLD  - execute the build command
5149                                    ;               EXEBNC  - execute the bounce command
5150                                    ;               EXELIS  - execute the listen command
5151                                    ;
5152                                    ; Register Usage - None
5153                                    ;
5154                                    ;--*
5155
5156
5157 037524                                    BGNTST
5158
5159 037524  105037  001301'          GETCL:  CLRB    P$GDBD                  ;CLEAR CMD LINE PARSING ERROR FLAG
5160 037530  105037  001300'                  CLRB    P$NNUF                  ;CLEAR NOT-ENOUGH FLAG
5161 037534  105037  001274'                  CLRB    P$LIST                  ;CLEAR LISTEN FLAG
5162 037540  105037  001275'                  CLRB    P$BLD                   ;CLEAR BUILD FLAG
5163 037544  105037  001306'                  CLRB    P$BONC                  ;CLEAR BOUNCE FLAG
5164 037550  105037  001276'                  CLRB    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$
5175 037634                                  PRINTF  #CLIERM                 ;IF NOT PRINT ERROR MESSAGE
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$
5180 037666                                  PRINTF  #CLINUF                 ;IF NOT PRINT ERROR MESSAGE
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,EXEHLP               ; 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$:
5198 037760  105737  001274'                 TSTB    P$LIST                  ; listen command?
5199 037764  001402                         BEQ     50$                     ; NAY!!
5200 037766  004737  056272'                 JSR     PC,EXELIS               ; execute listen command
5201
5202 037772  022737  000020  002024' 50$:    CMP     #CEXIT,CFLAG            ;WAS EXIT COMMAND TYPED?
5203 040000  001402                         BEQ     70$                     ;YES, LEAVE!!
5204 040002  000137  037524'                 JMP     GETCL                   ;IF NOT GET NEW COMMAND LINE
5205
5206 040006                          70$:    EXIT    TST                     ;  ELSE EXIT
5207
5208                                 .SBTTL  CLI ACTION TABLE AND ROUTINES
5209                                 ;       USER MUST CLEAR/SET P$GDBD IF USE "CLIBIF" IN CONNECTION WITH ACTION
5210                                 ;       R2 WILL HOLD ACTION CODE FROM PARSING (CLI) NODE
5211 040012                          CLIACT:
5212 040012  006302                         ASL     R2              ;MULTIPLY ACTION CODE BY 2
5213 040014  016202  040030'                 MOV     10$(R2),R2      ;OFFSET VALUE
5214 040020  062702  040030'                 ADD     #10$,R2         ;ADD BASE VALUE
5215 040024  004712                         JSR     PC,(R2)         ;GO DO ACTION
5216 040026  000207                         RTS     PC              ;RETURN TO TRVACT
5217
5218                                                                 ;BRIEF DESCRIPTION OF ACTION TAKEN
5219 040030  000152                  10$:    .WORD   ACTNUL-10$      ;0-NULL
5220 040032  000210                          .WORD   ACTHLP-10$      ;1-HELP
5221 040034  000262                          .WORD   ACTNOD-10$      ;2-NODE
5222 040036  000600                          .WORD   ACTBLD-10$      ;3-BUILD
5223 040040  005116                          .WORD   ACTRUN-10$      ;4-RUN SPECIFIED TEST
5224 040042  007322                          .WORD   ACTPAT-10$      ;5-SET 'MESSAGE PATTERN' TEST FLAG
5225 040044  011562                          .WORD   ACTSAV-10$      ;6-SAVE NODE TABLE
```

```
 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 'OALT' 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 - calulate 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$NNUF              ;SET FLAG TO SAY NEED MORE OF COMMAND
 5277
 5278                                   ;
 5279                                   ;ACTION ROUTINE TO DO NOTHING
 5280                                   ;
 5281
 5282 040202  000207                   ACTNUL: RTS     PC                      ;RETURN TO PARSER
 5283
 5284                                   ;
 5285                                   ;ACTION ROUTINE TO SET QUICK BUILD FLAG
 5286                                   ;
 5287
 5288 040204  000240                   ACTSQK: NOP
 5289 040206  105037  001300'                  CLRB    P$NNUF
 5290 040212  000207                          RTS     PC
 5291
 5292
 5293                                   ;
 5294                                   ; ACTION ROUTINE TO CLEAR QUICK BUILD FLAG
 5295                                   ;
 5296
 5297 040214  000240                   ACTCQK: NOP
 5298 040216  105037  001300'                  CLRB    P$NNUF
 5299 040222  000207                          RTS     PC
 5300
 5301                                   ;
 5302                                   ; ACTION ROUTINE TO SET NOCOMPARE FLAG
 5303                                   ;
 5304 040224  105037  001300'          ACTCMP: CLRB    P$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 - EXEHLP
 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,EXEHLP
 5325                                   ;
 5326                                   ; Side Effects - none
```

```
 5327                                   ;
 5328                                   ; Subordinate Routines - none
 5329                                   ;
 5330                                   ; Register Usage -
 5331                                   ;
 5332                                   ;--+
 5333 040250                            EXEHLP::
 5334 040250                                    P$PUSH  R1                      ; save R1
 5335 040252  012701  001310'                      MOV     #HLPTAB,R1              ; point R1 to table of addresses of help
 5336                                                                           ; messages
 5337 040256                            10$:    PRINTF  (R1)+                   ; print a line of help message
 5338 040274  020127  001412'                      CMP     R1,#HLPEND              ; at end of table?
 5339 040300  001366                               BNE     10$                     ; NO, go print more
 5340
 5341 040302  105037  001276'                      CLRB    P$HLP                   ; clear the help flag
 5342 040306                                    P$POP   R1                      ; restore R1
 5343 040310  000207                               RTS     PC                      ; and take off hose-head
 5344
 5345                                   ;
 5346                                   ;ACTION ROUTINE TO READ IN NODE PHY. ADDRESS, STORE IT IN ADRBUF
 5347                                   ;AND ENTER IT INTO THE NODE TABLE
 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
 5361 040424  001021                            BNE     17$                     ;IF YES, PRINT ERROR MESSAGE
 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$
 5372 040540  012737  017527' 001066' 20$:     MOV     #ARGTY6,KEYWD2          ;MOVE 'ASSIST' INTO KEYWD2
 5373 040546  012737  000001  001200'          MOV     #CASIST,NODTY
 5374 040554  012737  100000  001202' 25$:     MOV     #NODTBL,SLOT            ;POINT SLOT TO START OF NODE TABLE
 5375 040562                                    CALL    ENTRND                  ;CALL ROUTINE TO ENTER NODE IN TABLE
 5376 040570                                    P$POP   R1                      ;CHECK RESULTS
 5377 040572  001015                            BNE     50$                     ;IF NODE TABLE FULL, RETURN
 5378 040574  012737  017434' 001064'          MOV     #CMDTY7,KEYWD1          ;ELSE, MOVE "NODE" INTO KEYWD1
 5379 040602                                    PRINTS  #MSG2,#STRBUF           ;INDICATE IF TARGET OR ASSIST
 5380 040626  000207                    50$:    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                                   ;               NODTBL - 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                                   ;               RECEVE - routine to receive frames
 5440                                   ;               GETRNX - update receive ring pointers
```

```
5441                                    ;               CMPEXT - compare received addresses with node table entries
5442                                    ;               MOVEXT - 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$POP   R2                      ; see what find slot has to say
5460 040744  001402                            BEQ     3$                      ; branch if there is an empty slot
5461 040746  000137  041662'                   JMP     80$                     ; else, leave
5462 040752                            3$:
5463 040752                                    CALL    DEVSTART                ; start up the DELUA/DEUNA
5464 040760                                    call    funct  #wdmula          ; write multicast address list
5465 040772                                    P$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$:    clr     temp                    ; clear 'no. nodes in last min.' counter
5469 041012  005037  003112'                   clr     temp1                   ; clear node type argument (set to target)
5470 041016  005037  003114'                   clr     temp2                   ; set interval counter
5471 041022  012737  000012  003116'           mov     #12,temp3               ; set 'mins. since last new node' counter
5472 041030  012737  100000  001202'           mov     #nodtbl,slot            ; set slot to begining of node table
5473 041036                            19$:
5474 041036  012737  000074  002052'           mov     #60.,timers
5475 041044                            20$:
5476 041044                                    break                           ; allow for control c interruption
5477 041046  005737  002052'                   tst     timers                  ; see if interval is up
5478 041052  001002                            bne     201$                    ; Its's not, keep going
5479 041054  000137  041506'                   jmp     40$                     ;
5480
5481 041060                            201$:   CALL    RECEVE                  ;  else, check for reception of id message
5482 041066                                    P$POP   R2                      ; R2 holds no of messages received
5483 041070  001765                            beq     20$                     ; if none, keep looking
5484 041072  012737  000013  003116'           mov     #13,temp3               ; got one : reset 'mins. since new node'
5485 041100  013703  002100'                   mov     rrgnxt,R3               ;           save receive ring pointer
5486 041104                                    CALL    GETRNX  #RRGNXT         ; update pointer
5487 041116  016304  000010                    MOV     10(R3),R4               ; point R4 to receive buffer
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     @sourcc,R4              ; point R4 to node address
5501 041154  012702  100000                    mov     @nodtbl,R2              ; point R2 to node table
5502 041160                            21$:
5503 041160                                    CALL    CMPEXT  @ONTAB,R2,@ORRING,R4,@3 ; see if node already on table
5504 041206                                    P$POP   R1
5505 041210  001476                            beq     30$                     ; if same, don't add to table
5506 041212                            22$:
5507 041212  062702  000010                    add     @10,R2                  ; point to next table entry
5508 041216  020227  110000                    CMP     R2,@NODEND              ; check to see if end of table
5509 041222  001356                            bne     21$                     ; if no, compare next entry
5510
5511                                   ;--+
5512                                   ;       After all entries in the node table have been checked and a match
5513                                   ;       has not been found, try to add the new node address to the table.
5514                                   ;--+
5515
5516 041224                                    CALL    FINDSL                  ; Look for an empty entry in the table
5517 041232                                    P$POP   R2                      ; get table full indicator
5518 041234  001071                            bne     35$                     ; non-zero return means table full
5519
5520                                   ;--+
5521                                   ;       Add node address and node type to node table
5522                                   ;--+
5523
5524 041236  013702  001202'                   mov     slot,R2                 ; point R2 to slot in node table
5525 041242                                    CALL    MOVEXT  @ORRING,R4,@ONTAB,R2,@3 ; move addr. into node table
5526
5527                                   ;--+
5528                                   ;       Now add address to default node table
5529                                   ;--+
5530
5531 041270  062702  010000                    ADD     @DEFNOD,R2              ; point R2 entry in default addr. table
5532 041274  162704  000006                    sub     @sourcc,R4              ; point R4 back to start of frame
5533 041300                                    call    getida  R4,@7           ; get address of default hardware address
5534 041314                                    p$pop   r1                      ; r1 points to default hardware address
5535 041316                                    CALL    MOVEXT  @ORRING,R1,@ONTAB,R2,@3 ; save default address
5536
5537                                   ;--+
5538                                   ;       Get node type and store it in default node table
5539                                   ;--+
5540 041344                                    call    getida  R4,@144         ; get node type address
5541 041360                                    p$pop   r1                      ; r1 points to node type
5542 041362  111101                            movb    (r1),r1                 ; put node type in r1
5543 041364                                    CALL    REMAP   @ONTAB          ; allow access to node table
5544 041376  110162  000007                    MOVB    R1,7(R2)                ; save node type in default table
5545
5546 041402  005237  003110'                   inc     temp                    ; increment 'nodes in last min.' counter
5547 041406                            30$:    CALL    RELBUF  R3              ; release buffer to DELUA/DEUNA
5548 041416  000612                            br      20$                     ; check for more input
5549
5550 041420                            35$:
5551 041420                                    CALL    RELBUF  R3              ; release buffer to DELUA/DEUNA
5552 041430  012737  000005  002052'           mov     @5, TIMERS              ; allow 5 seconds for cleanup
5553
5554 041436                            36$:
```

```
5555 041436                                        CALL    RECEVE                  ; keep fetching frames until they stop
5556 041444                                        P$POP   R2
5557 041446  001413                                BEQ     38$                     ; branch if none received
5558 041450  013703  002100'                       MOV     RRGNXT,R3               ; point R3 to received entry
5559 041454                                        CALL    RELBUF  R3              ; release buffer to DELUA/DEUNA
5560 041464                                        CALL    GETRNX  @RRGNXT         ; update ring pointer
5561 041476                                38$:
5562 041476  005737  002052'                       TST     TIMERS                  ; is time up?
5563 041502  001355                                BNE     36$                     ; branch if time is not up
5564 041504  000431                                BR      50$                     ; yes, leave
5565 041506                                40$:
5566 041506  005337  003116'                       dec     temp3                   ; see if 10 mins since last node
5567 041512  001426                                beq     50$                     ;  if yes, exit
5568 041514  005237  003114'                       inc     temp2                   ; see if time is up
5569 041520  023727  003114' 000050                cmp     temp2,#40.
5570 041526  001420                                beq     50$                     ; if yes, exit
5571 041530                                        PRINTS  #bldmsg,temp,temp2      ;  else, print "still working" message
5572 041560  005037  003110'                       clr     temp
5573 041564  000137  041036'                       JMP     19$                     ; do it again
5574 041570                                50$:
5575 041570                                        PRINTS  #blddon,temp2           ; print "build complete" message
5576 041614  012737  000000  002326'               mov     #0,$wdmc+4              ; clear multicast address list
5577 041622                                        call    funct   #WDMULA         ;  write 0 into list length
5578 041634                                        P$POP   R2                      ; check for error
5579 041636  001404                                beq     55$                     ; cont ue if ok
5580 041640                                        errdf   29,emsg25,err1          ;  else, report error
5581 041650                                55$:
5582 041650  004737  051006'                       jsr     pc,actsnd               ; print node table
5583 041654  012737  000400  002326'               mov     #400,$wdmc+4            ; reset multicast list for 1 entry
5584 041662                                80$:
5585 041662  105037  001275'                       CLRB    P$BLD                   ; clear build flag
5586 041666                                        CALL    DEVSTOP                 ; stop the DELUA/DEUNA
5587 041674                                        CALL    RETMEM                  ; return memory to original mapping
5588 041702                                        P$POP   R1,R2,R3,R4             ; restore registers
5589 041712  000207                                RTS     PC
5590                                       ;
5591                                       ; ACTION ROUTINE TO CALCULATE ADDRESS FROM LOGICAL NODE NUMBER
5592                                       ;
5593 041714                                ACTBLG: P$PUSH  R2                      ;SAVE R2
5594 041716                                        CALL    REMAP   #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$:     PRINTF  #EMSG46                 ; report it
5607 042000  112737  177777  001301'               MOVB    #-1,P$GDBD              ; set error
5608 042006  000410                                BR      20$                     ; leave
5609 042010                                10$:
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'                CLRB    P$AERR                  ; clear address error flag
 5614 042030                         20$:
 5615 042030                                 P$POP   R2                      ; restore regs
 5616 042032                                 CALL    RETMEM                  ; restore memory mapping
 5617 042040  000207                         RTS     PC                      ;continue
 5618
 5619                                ;--+
 5620                                ; Name   ACTSBB                         Switch for bounce actions routines
 5621                                ;
 5622                                ; Functional Description:
 5623                                ;               This routine is a simple multiplexor between two action
 5624                                ;               routines for the BOUNCE command.  The reason for it is that
 5625                                ;               for the first node specified in the bounce command a different
 5626                                ;               action will take place other than for the rest of the nodes
 5627                                ;               specified in the command.  Namely, the first node specified
 5628                                ;               will be used as the destination of the bounced message, whereas
 5629                                ;               the remaining nodes (if there are any specified) will be
 5630                                ;               used as forward loop request fields.  The routine simply
 5631                                ;               compares XRGNXT to XRGCUR.  If they are equal it calls ACTIBB
 5632                                ;               else it calls ACTFBB.
 5633                                ;
 5634                                ; Inputs - none
 5635                                ;
 5636                                ; Outputs - none
 5637                                ;
 5638                                ; Calling procedure: JSR PC,ACTSBB
 5639                                ;
 5640                                ; Side effects -
 5641                                ;               1.) will invoke one of the two action routines named above
 5642                                ;
 5643                                ; Subordinate Routines -
 5644                                ;               ACTIBB - initialize bounce buffer
 5645                                ;               ACTFBB - fill bounce buffer
 5646                                ;
 5647                                ; Register Usage - none
 5648                                ;
 5649                                ;--+
 5650 042042                        ACTSBB::
 5651 042042  023737  002076' 002072'        CMP     XRGNXT,XRGCUR                   ; has a buffer been allocated?
 5652 042050  001003                         BNE     10$                             ; Yes, call ACTFBB
 5653
 5654 042052  004737  042066'                JSR     PC,ACTIBB                       ; Else, call ACTIBB
 5655 042056  000402                         BR      20$                             ; ... and exit
 5656
 5657 042060  004737  042224'        10$:    JSR     PC,ACTFBB                       ; ....
 5658 042064  000207                 20$:    RTS     PC                              ; DONE!!
 5659
 5660
 5661                                ;--+
 5662                                ; Name - ACTIBB                 Initialize the bounce buffer
 5663                                ;
 5664                                ; Functional Description:
 5665                                ;               This action routine is called to initialize a transmit
 5666                                ;               buffer to be used in the BOUNCE command.  Also, it
 5667                                ;               initializes some pointers that the BOUNCE routine must
 5668                                ;               know about.
```

```
5669                                    ;
5670                                    ; Inputs - Implicit
5671                                    ;               ADRBUF - contains six bytes of destination address
5672                                    ;
5673                                    ; Outputs - none
5674                                    ;
5675                                    ; Calling Procedure: JSR PC,ACTIBB
5676                                    ;
5677                                    ; Side Effects -
5678                                    ;               1.) Transmit buffer pointed to by XRGNXT is initialized for
5679                                    ;                   bounce command
5680                                    ;               2.) Variables initialized:
5681                                    ;                   BNCBUF - pointer to beginning of transmit buffer
5682                                    ;                   BNCCNT - number of loop information bytes -- set to 2 for
5683                                    ;                            skip count
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                            ACTIBB::
5694 042066                                    P$PUSH  R1                      ; Save R1
5695 042070                                    CALL    DEVSTART                ; start up the DELUA/DEUNA
5696 042076                                    CALL    REMAP   #OTRING         ; allow access to transmit ring
5697 042110  013701  002076'                   MOV     XRGNXT,R1               ; point R1 to next entry in ring
5698 042114  016137  000010  002062'          MOV     10(R1),BNCBUF           ; save pointer to transmit buffer
5699 042122  016101  000010                    MOV     10(R1),R1               ; point R1 to transmit buffer
5700
5701 042126  013711  001070'                   MOV     ADRBUF,(R1)             ; store six ...
5702 042132  013761  001072'  000002          MOV     ADRBUF+2,2(R1)          ; ... bytes of destination address ...
5703 042140  013761  001074'  000004          MOV     ADRBUF+4,4(R1)          ; ... in transmit buffer
5704
5705 042146  013761  003034'  000014          MOV     PROTOO,PROTOT(R1)       ; fill in protocol type
5706
5707 042154  005061  000016                    CLR     16(R1)                  ; skip count equals zero
5708 042160  012737  000002  002064'          MOV     #2,BNCCNT               ; two bytes of data are in data
5709                                                                           ; field (skip count)
5710
5711 042166  112737  177777  001306'          MOVB    #-1,P$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 - ACTFBB                         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 -
```

```
                                ;               ADRBUF - contains the address to forward to
                                ;
                                ; Outputs - none
                                ;
                                ; Calling Procedrue: JSR PC,ACTFBB
                                ;
                                ; Side Effects -
                                ;               1.) A forward function is added to the buffer pointed to by
                                ;                   BNCBUF
                                ;               2.) BNCCNT is update to reflect the addition of data to the
                                ;                   buffer
                                ;
                                ; Subordinate Routines -
                                ;               REMAP - remap a portion of virtual memory
                                ;               RETMEM - restore memory mapping
                                ;
                                ; Register Usage -
                                ;               R2 - pointer to transmit buffer
                                ;
                                ;--+
042224                          ACTFBB::
042224                                  P$PUSH  R2                      ; save R2
042226                                  CALL    REMAP   #OTRING         ; allow access to transmit ring
042240  013702  002062'                MOV     BNCBUF,R2               ; point R2 to transmit buffer
042244  062702  000016                  ADD     #16,R2                  ; point R2 past header info
042250  063702  002064'                ADD     BNCCNT,R2               ; point R2 past info already in data field
                                ;--+
                                ;       Update count of information contained in this bounce buffer.
                                ;       If the result is greater than the message size then abort attempt
                                ;--+
042254  062737  000010  002064'         ADD     #10,BNCCNT              ; update bounce count
042262  023737  002064' 001172'         CMP     BNCCNT,P$SIZE           ; Is this greater than message size
042270  003414                          BLE     10$                     ; NO!
042272  112737  177777  001301'         MOVB    #-1,P$GDBD              ; indicate bad command to parser
042300                                  PRINTF  #EMSG45                 ; Tell user of problem
042320  000410                          BR      20$                     ; and take off
                                ;
042322  012722  000002          10$:    MOV     #2, (R2)+               ; set forward function code
042326  013722  001070'                 MOV     ADRBUF, (R2)+           ; set 6 bytes of forwarding address
042332  013722  001072'                 MOV     ADRBUF+2, (R2)+
042336  013722  001074'                 MOV     ADRBUF+4, (R2)+
                                ;
042342                          20$:    CALL    RETMEM                  ; restore memory mapping
042350                                  P$POP   R2                      ; restore R2
042352  000207                          RTS     PC                      ; return
                                ;
                                ;
                                ;--+
                                ; Name - EXEBNC                         Execute bounce command
                                ;
                                ; Functional Description:
                                ;               This routine is called to carry out the Bounce command
                                ;               of the NI Exercisor.  The bounce command is a function supplied
                                ;               to the user so that he/she may choose any path of nodes
                                ;               on the NI to loop a packet through.
                                ;                       To carry out this function a loop request message
```

```
5783                    ;               is created with each of the nodes specified in the input
5784                    ;               command line used as a forwarding field of the message.
5785                    ;               To complete the loop the last forwarding field along with
5786                    ;               the reply field is set to our own address.  For example, if
5787                    ;               the following command were input:
5788                    ;
5789                    ;               NIE > bounce/AA0004000010,N3,N5,N7
5790                    ;
5791                    ;               then this loop request message would result:
5792                    ;
5793                    ;                       +-------------------------+
5794                    ;                       )      DESTINATION        )
5795                    ;                       )   AA-00-04-00-00-10     )
5796                    ;                       +-------------------------+
5797                    ;                       )      SOURCE             )
5798                    ;                       +-------------------------+
5799                    ;                       )      PROTOCOL TYPE      )
5800                    ;                       +-------------------------+
5801                    ;                       )      FORWARD            )
5802                    ;                       )        N3               )
5803                    ;                       +-------------------------+
5804                    ;                       )      FORWARD            )
5805                    ;                       )        N5               )
5806                    ;                       +-------------------------+
5807                    ;                       )      FORWARD            )
5808                    ;                       )        N7               )
5809                    ;                       +-------------------------+
5810                    ;                       )      FORWARD            )
5811                    ;                       ) OUR ETHERNET ADDRESS    )
5812                    ;                       +-------------------------+
5813                    ;                       )      REPLY              )
5814                    ;                       ) OUR ETHERNET ADDRESS    )
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 BNCBUF 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                            EXEBNC: P$PUSH  R2                      ; save r2 and r3
5856 042356                                    CALL    REMAP   #OTRING         ; allow access to transmit ring
5857
5858                                    ;--+
5859                                    ;       Position the pointer to the transmit buffer so that it points to
5860                                    ;       where more loop info should be added.
5861                                    ;--+
5862 042370  [illegible]  002062'               MOV     BNCBUF,R2               ; let R2 point to transmit buffer
5863 042374  [illegible]  [illegible]           ADD     #16,R2                  ; point R2 past header info
5864 042400  063702  002064'                    ADD     BNCCNT,R2               ; point R2 past loop data already in
5865                                                                            ; buffer
5866                                    ;--+
5867                                    ;       Update the count of loop information in the bounce buffer.  If it
5868                                    ;       is greater than the message size (P$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$:
5879
5880                                    ;--+
5881                                    ;       Add last forward address and the reply message to the bounce buffer.
5882                                    ;       They will both be the device's physical address.
5883                                    ;--+
5884 042452  012722  000002                    MOV     #2, (R2)+               ; put our address as forwarding address
5885 042456  013722  002244'                   MOV     PHYADR, (R2)+
5886 042462  013722  002246'                   MOV     PHYADR+2, (R2)+
5887 042466  013722  002250'                   MOV     PHYADR+4, (R2)+
5888 042472  012722  000001                    MOV     #1, (R2)+               ; set reply message
5889 042476  013722  002244'                   MOV     PHYADR, (R2)+           ; put our address in here
5890 042502  013722  002246'                   MOV     PHYADR+2, (R2)+         ; 6 bytes worth
5891 042506  013722  002250'                   MOV     PHYADR+4, (R2)+
5892
5893 042512                                    CALL    BLDBUF  BNCBUF,BNCCNT   ; fill the buffer with data patterns
5894
5895 042530                                    CALL    XMIT                    ; transmit the buffer
5896 042536                                    P$POP   R2                      ; 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$:
 5902 042552                                  CALL    RUNCOM                  ; execute common receive
 5903 042560                                  P$POP   R2                      ; get results
 5904 042562  001410                          BEQ     40$                     ; branch if no error
 5905 042564  112737  177777  001301'         MOVB    #-1, P$GDBD             ; set error flag
 5906 042572                                  ERRSOFT 30,EMSG34
 5907 042602  000410                          BR      50$                     ; leave
 5908 042604                          40$:
 5909 042604                                  PRINTF  #OK                     ; say it arrived a okay
 5910 042624                          50$:
 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 Desciption:
 5934                                 ;               This action routine is called to print out the summary
 5935                                 ;               data counters kept by the NIE.
 5936                                 ;
 5937                                 ; Inputs - Implicit -
 5938                                 ;               STATBL  - table containing the summary data
 5939                                 ;
 5940                                 ; Outputs -
 5941                                 ;               1.) summary data is printed at the user terminal
 5942                                 ;
 5943                                 ; Calling Procedure: JSR PC,ACTSUM
 5944                                 ;
 5945                                 ; Side Effects - none
 5946                                 ;
 5947                                 ; Subordinate Routines -
 5948                                 ;               BINHEX  - convert binary data to HEX character string
 5949                                 ;               BINDEC  - convert binary data to decimal character string
 5950                                 ;               REMAP   - used to map summary table into page registers
 5951                                 ;               RETMEM  - restore memory mapping
 5952                                 ;
 5953                                 ; Register Usage -
```

```
 5954                                   ;               R1      - pointer to summary table
 5955                                   ;               R2,R3,R4 - summary data
 5956                                   ;
 5957                                   ;--*
 5958
 5959 042674  105037  001300'          ACTSUM: CLRB    P$NNUF                  ;CLEAR NOTNUF FLAG
 5960 042700                                   CALL    REMAP   #OSTAB          ; allow access to summary table
 5961 042712                                   P$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     5$                      ;  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 transfered
 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$
6023 043262  000137  044026'                   JMP     70$                     ; IF NOT, EXIT ACTION ROUTINE
6024
6025 043266                            5$:     P$PUSH  R1,R2,R3,R4             ; save registers
6026 043276  105037  001300'                   CLRB    P$NNUF                  ;CLEAR NOTNUF FLAG
6027 043302                                    CALL    CMPTWO #ADRBUF,#ILLADR,#3 ; see if illegal address
6028 043324                                    P$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$:    CALL    CMPTWO  #ADRBUF,#PHYADR,#3 ; see if address is own (host node)
6034 043376                                    P$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$:    mov     #timers,R4              ; set up for 10 second timout
6057 043534  012714  000012                    mov     #10.,(R4)
6058
6059 043540                            30$:    break
6060 043542  005714                            tst     (R4)                    ; see if time has expired
6061 043544  001431                            beq     35$                     ;  if yes, branch
6062 043546                                    CALL    RECEVE                  ; check for answer
6063 043554                                    P$POP   R2                      ;  R2 holds no. of buffers received
6064 043556  001770                            beq     30$                     ;  if no buffers recieved, loop
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     sircpt(R4),#"MR         ; see if message recieved 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$:    inc     s.rec                   ; increment 'received messages' counter
 6081 043660  062737  000056  003000'           add     #46.,s.xfer             ; update 'bytes transfered' counter
 6082
 6083 043666                                    call    prntid  r4              ; Print the system id info
 6084
 6085 043676                            50$:    CALL    REMAP   #ORRING         ; 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                            52$:    call    writes  #1,R4,#orring   ; update summary table
 6089 043740                                    CALL    RELBUF  R3              ; release buffer to DELUA/DEUNA
 6090
 6091 043750  005061  000002            55$:    clr     2(R1)                   ; disable transmit padding
 6092 043754                                    CALL    FUNCT  #WDMODE
 6093 043766                                    P$POP   R2                      ; check for error
 6094 043770  001404                            BEQ     65$                     ; ain't none
 6095 043772                            60$:    errdf   32,emsg23,err1          ; error -- can't write mode
 6096
 6097 044002                            65$:    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
 6102
 6103
 6104                                   ;
 6105                                   ;ACTION ROUTINE TO CHECK FOR ADDITION PARAMETER CHANGE INPUTS
 6106                                   ;AND PRINT OUT NEW PARAMETER INFO WHEN ALL INPUT ARE PROCESSED
 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$:    MOV     #CMDTY6,KEYWD1
 6112 044042  013701  001170'                   MOV     P$TYPE,R1               ;GET MESSAGE TYPE ASCII STRING ADDRESS
 6113 044046  006301                            ASL     R1                      ;INTO R1
 6114 044050  062701  001414'                   ADD     #MSGTAB,R1
 6115 044054                                    PRINTF  #MSGPRM                 ;PRINT 'MESSAGE' COMMAND MESSAGE
 6116 044074                                    PRINTF  #MSG4,(R1),P$SIZE,P$CPYS        ;PRINT MSG PARAMETERS
 6117 044126  105037  001300'                   CLRB    P$NNUF                  ;CLEAR NOTNUF FLAG
 6118 044132  000207                    50$:    RTS     PC
 6119
 6120
 6121                                   ;
 6122                                   ;ACTION ROUTINE TO RETURN CONTROL TO THE SUPERVISOR
 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 CBOBUF
 6135 044150                                   CALL    EDPACK #CBOBUF,#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'                  CLRB    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$TYPE           ;SET MESSAGE TYPE
 6159 044256  000207                   RTS     PC
 6160
 6161
 6162                                  ;
 6163                                  ;ACTION ROUTINE TO SET MESSAGE TYPE = ALL ONES FLAG
 6164                                  ;
 6165
 6166 044260  012737  000001  001170' ACTONE: MOV     #ONES,P$TYPE            ;SET MESSAGE TYPE
 6167 044266  000207                   RTS     PC
 6168
 6169
 6170                                  ;
 6171                                  ;ACTION ROUTINE TO SET MESSAGE TYPE = ALL ZEROS FLAG
 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$TYPE           ;SET MESSAGE TYPE
 6200 044326  000207                   RTS     PC
 6201
 6202
 6203                                 ;
 6204                                 ;ACTION ROUTINE TO SET MESSAGE TYPE = OPERATOR SELECTED INPUT
 6205                                 ;
 6206
 6207 044330  105037  001304' 000000 ACTOPR: CLRB    P$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                                 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'         MOVB    #-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 EXISTA
 6258                                         ;
 6259
 6260 044516  023727  001270'  000037  ACTSZE: CMP     P$NUM,#31.              ;CHECK FOR VALID SIZE RANGE
 6261 044524  003410                           BLE     10$
 6262 044526  022737  002673  001270'         CMP     #1467.,P$NUM
 6263 044534  003404                           BLE     10$                     ;IF VALID CONTINUE
 6264 044536  013737  001270'  001172'         MOV     P$NUM,P$SIZE            ;SET MESSAGE SIZE
 6265 044544  000410                           BR      20$
 6266 044546                           10$:    PRINTF  #SIZLMT                 ;PRINT SIZE LIMITS EXCEEDED MESSAGE
 6267 044566  004737  044030'          20$:    JSR     PC,ACTMSG               ;CHECK FOR ADDITIONAL COMMANDS
 6268 044572  000207                           RTS     PC
 6269
 6270
 6271                                         ;
 6272                                         ;ACTION ROUTINE TO INPUT COPIES PARAMETER, CHECK TO SEE IF IT IS
 6273                                         ;WITHIN LEGAL LIMITS, CHANGE PARAMETER AND THEN RETURN TO SEE IF
 6274                                         ;MORE INPUT PARAMETERS EXIST
 6275                                         ;
 6276
 6277 044574  023727  001270'  000000  ACTCPY: CMP     P$NUM,#0                ;CHECK FOR VALID COPIES RANGE
 6278 044602  003410                           BLE     10$
 6279 044604  022737  000400  001270'         CMP     #256.,P$NUM
 6280 044612  003404                           BLE     10$                     ;IF VALID, CONTINUE
 6281 044614  013737  001270'  001174'         MOV     P$NUM,P$CPYS            ;SET MESSAGE COPIES
 6282 044622  000410                           BR      20$
 6283 044624                           10$:    PRINTF  #CPYLMT                 ;PRINT COPY LIMIT EXCEEDED MESSAGE
 6284 044644  004737  044030'          20$:    JSR     PC,ACTMSG               ;CHECK FOR ADDITIONAL COMMANDS
 6285 044650  000207                           RTS     PC
 6286
 6287
 6288                                         ;
 6289                                         ;ACTION ROUTINE TO CLEAR NODE SPECIFIED BY PHYSICAL ADDRESS FROM NODE TABLE
 6290                                         ;
 6291
 6292 044652  105037  001300'          ACTNAD: CLRB    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$:    CLR     (R3)+                   ;ELSE, CLEAR NODE FROM TABLE
 6310 044772  005023                    CLR     (R3)+
 6311 044774  005023                    CLR     (R3)+
 6312 044776  005013                    CLR     (R3)
 6313 045000                            PRINTF  @ADRDEL                 ;PRINT NODE DELETED FROM TABLE MESSAGE
 6314
 6315 045020                    30$:    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                    A' 'IAL: 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                            LOGNM::
 6356 045126                                    P$POP   R1                      ; Get address of node
 6357 045130  162701  100000                    SUB     #NODTBL,R1              ; Make it an offset from base
 6358 045134  006201                            ASR     R1                      ;       DIVIDE
 6359 045136  006201                            ASR     R1                      ;         BY
 6360 045140  006201                            ASR     R1                      ;       EIGHT
 6361 045142                                    RETURN  R1                      ; return the logical value
 6362
 6363
 6364                                   ;--*
 6365                                   ; Name - ACTRUN                         Run a specified test
 6366                                   ;
 6367                                   ; Functional Desciption:
 6368                                   ;               This routine is called by the parse routine to run
 6369                                   ;               the user specified test.  It looks at the variable
 6370                                   ;               KEYWD1 to determine which test it should call up, then
 6371                                   ;               invokes the appropriate test.  Also, it keeps track
 6372                                   ;               of the pass count and calls the specified test the
 6373                                   ;               appropriate number of times.
 6374                                   ;
 6375                                   ; Inputs - Implicit -
 6376                                   ;               KEYWD1  - contains integer representing a test number
 6377                                   ;               P$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                ; start up the DELUA/DEUNA
 6401 045166  022737  000032  001064' 5$:      CMP     #CRNALL,KEYWD1          ; SEE IF 'ALL' TEST
 6402 045174  001004                            BNE     10$                     ;  IF NO, CONTINUE
 6403 045176                                    CALL    RUNALL                  ;  IF YES, DO ALLNODE
 6404 045204  000423                            BR      30$
 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$
 6409 045226  022737  000043  001064' 15$:     CMP     #CDIR,KEYWD1            ; IS IT 'DIRECT' TEST
```

```
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 Desciption:
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                                  printe  #pesabt                 ;  else abort test and print message
6474 045352  000137  045746'        3$:     jmp     32$
6475 045356  012737  100000  001202' 5$:     mov     #modtbl,slot            ; move node table address to slot
6476 045364                                  CALL    FULSLT                  ; find first entry
6477 045372  013701  001202'                 mov     slot,R1                 ;  and put target address into R1
   5 045376  013737  001174' 003122' 10$:    mov     P$CPYS,cpycnt           ; set up loop for no. of copies
   3 045404  062737  000010  001202'         add     #10,slot                ; update slot
 ..0 045412                                  CALL    FULSLT                  ; get next assist node from table
6481 045420  013702  001202'                 mov     slot,R2
6482 045424  022737  177777  001202'         cmp     #-1,slot                ;  see if at end of table
6483 045432  001530                         beq     25$                     ;  if yes, br
6484 045434                          15$:    CALL    BLDFAS  R1,slot         ; build full assist message
6485 045450                                  CALL    XMIT                    ; transmit message
6486 045456                                  P$POP   R3                      ;  check results
6487 045460  001346                         BNE     10$                     ; transmit failed -- try next pair
6488
6489 045462                          17$:    CALL    REMAP   #ONTAB          ; allow access to node table
6490 045474                                  call    binhex  R1,#6,#strbuf   ; set up buffers for error print ...
6491 045514                                  call    binhex  r2,#6,#strbu1   ; ... if necessary
6492
6493 045534                                  CALL    LOGNM   R1              ; put the logical node name for ...
6494 045544                                  P$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$:    errsoft 33,emsg42,ERR3          ;  ... and print failing nodes
6504 045636  000410                         br      101$
6505
6506 045640                          21$:    printb  #okfu
6507 045660  005337  003122'        101$:   dec     cpycnt                  ; decrement 'copies' counter
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$
6511 045714  062701  000010        25$:    add     #10,R1                  ; point R1 to next target node
6512 045720  010137  001202'                 mov     R1,slot                 ; update slot
6513 045724                                  CALL    FULSLT                  ; get address from table
6514 045732  013701  001202'                 MOV     SLOT,R1
6515 045736  022737  177777  001202'         cmp     #-1,slot                ; see if end of table
6516 045744  001214                         bne     10$                     ;  if no, continue else, finished
6517 045746                          32$:    RETURN
6518
6519                                 ;
6520                                 ;ACTION ROUTINE TO SET 'RUN LOOP DIRECT' FLAG
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 Desciption:
 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$POP   R3                      ; ... by slot and store in R1
 6585 046120                                CALL    REMAP   #ONTAB          ; allow access to node table
 6586 046132                                CALL    BINHEX  SLOT,#6,#STRBUF ; STRBUF holds address of node that w ll
 6587                                                                       ; be looped directly to
 6588
 6589 046154                        15$:    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$TYPE,R1
 6593 046216  006301                        ASL     R1
 6594 046220  062701  001414'               ADD     #MSGTAB,R1
 6595 046224                                PRINTB  #MESPA1,(R1)
 6596
 6597 046246                        16$:    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$:    CALL    RUNCOM                  ; do recieve loop
 6603 046300                                P$POP   R4                      ; get results
 6604 046302  001407                        beq     29$                     ;  if no errors, continue
 6605
 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                  ; decrement 'copies' counter
 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' ACTRNL: 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
```

```
                                        ;
                                        ; Calling Procedure - CALL RUNLUP
                                        ;
                                        ; Side effects - The user sees information on the success or failure of each
                                        ;                attempted looping of a frame.
                                        ;
                                        ; Register Usage -
                                        ;                R1 - Pointer into the node table.  This node will be used to
                                        ;                     assist in the looping.
                                        ;                R2 - Pointer into the node table.  This node will be used as
                                        ;                     the target of the looping.
                                        ;                R3 - Integer representing the logical node name of the assist
                                        ;                     node.
                                        ;                R4 - Integer representing the logical node name of the target
                                        ;                     node.
                                        ;
                                        ;--+
046424  012737  100000  001202' RUNLUP: MOV     #NODTBL,SLOT            ; move node table address to slot
046432                                  CALL    FULSLT                  ; see if table empty
046440  022737  177777  001202'         CMP     #-1,SLOT                ;
046446  001014                          BNE     5$                      ;  if no, continue
046450                                  PRINTF  #TABEMT,#NOD            ;  else, print "Table empty" message
046474  000137  047054'                 JMP     50$
046500  012737  100000  001202' 5$:     MOV     #NODTBL,SLOT            ; move node table address to slot
046506                                  CALL    FULSLT                  ; get first node in node table
046514  013737  001202'  003112'        MOV     SLOT,TEMP1              ; save first node to pair with last
046522  013737  001174'  003122' 10$:    MOV     P$CPYS,CPYCNT           ; set up for no. of copies
046530  013701  001202'                 MOV     SLOT,R1                 ; R1 points to assist node
046534  062737  000010  001202'         ADD     #10,SLOT                ; point SLOT to next entry in node table
046542                                  CALL    FULSLT                  ; get next node in table
046550  022737  177777  001202'         CMP     #-1,slot                ; see if at end of table
046556  001003                          BNE     15$                     ;
046560  013702  003112'                 MOV     TEMP1,R2                ; Use first node in node table as target
046564  000402                          BR      20$                     ; This will be the last loop tested
046566  013702  001202'         15$:    MOV     SLOT,R2                 ; R2 Points to target node
046572                           20$:    CALL    BLDFAS  R2,R1           ; build full assist message
046604                                  CALL    XMIT                    ; transmit message
046612                                  P$POP   R4                      ; check results
046614  001077                          BNE     35$                     ; transmit failed -- try next pair
046616                           25$:    CALL    LOGNM   R1              ; get logical node name for assist ...
046626                                  P$POP   R3                      ; ... and put it in R3
046630                                  CALL    LOGNM   R2              ; get logical node name for target ...
046640                                  P$POP   R4                      ; ... and put it in R4
046642                                  PRINTB  #TSTMS4,#ARGTY7,R4,#ARGTY6,R3   ;   assist node =
                                        ;
                                        ; Set up STRBUF, STRBU1 with addresses of the two nodes involved in this test
                                        ;
046676                                  CALL    REMAP   #ONTAB          ; allow access to node table
046710                                  CALL    BINHEX  R2,#6,#STRBUF   ; STRBUF has target node
046730                                  CALL    BINHEX  R1,#6,#STRBU1   ; STRBU1 has assist node
```

```
 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$:  CALL    RETMEM                  ; restore memory mapping
 6712 047062                                RETURN
 6713
 6714                                ;--+
 6715                                ; Name - RUNCOM                         Common receive code
 6716                                ;
 6717                                ; Functional Desciption:
 6718                                ;               This routine will perform the reception of loopback
 6719                                ;               messages transmitted by any of the loopback tests.
 6720                                ;               It will wait for ten seconds for the reply to the loopback
 6721                                ;               message.  If it successfully receives the message, it
 6722                                ;               performs a data comparison on what was transmitted to what
 6723                                ;               was received.
 6724                                ;                       The success of these operations will be returned
 6725                                ;               to the caller.
 6726                                ;
 6727                                ; Inputs - none
 6728                                ;
 6729                                ; Outputs -     P1 - 0 = successful reception of loop message/ -1 = no success
 6730                                ;
 6731                                ; Calling Procedure:    CALL RUNCOM
 6732                                ;                       P$POP   P1
 6733                                ;
 6734                                ; Side Effects -
 6735                                ;               1.) summary data counters are modified on error
 6736                                ;
 6737                                ; Subordinate Routines -
 6738                                ;               RECEVE  - 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     retrys                  ; see if failed due to excessive collisions
 6753 047070  001402                          beq     34$                     ;  if not, then try to receive
 6754 047072  000137  047330'                 jmp     50$                     ;  else, take off
 6755
 6756 047076  012704  002052'          34$:    mov     #timers,R4              ; set up for 10 second timout
 6757 047102  012714  000012                  mov     #10.,(R4)
 6758 047106  005002                          clr     R2                      ; clear results register
 6759 047110                           35$:    break
 6760 047112  005714                          tst     (R4)                    ; see if time has expired
 6761 047114  001475                          beq     40$                     ;  if yes, branch
 6762 047116                                  CALL    RECEVE                  ; check for answer
 6763 047124                                  P$POP   R1                      ;  R2 holds no. of buffers received
 6764 047126  001770                          beq     35$                     ;  if no buffers recieved, loop
 6765 047130  063737  003120' 003000'        add     xfer,s.xfer             ; update bytes transfered sum. counter
 6766 047136  005237  002766'                 inc     s.rec                   ; update frames received sum. counter
 6767 047142  013703  002100'                 mov     RRGNXT,R3               ; get receive ring pointer
 6768 047146                                  CALL    GETRNX #RRGNXT          ; update pointer
 6769 047160  016301  000006                  mov     6(R3),R1                ; get frame length from discriptor
 6770 047164  042701  170000                  bic     #170000,R1              ;  zero out excess infor
 6771 047170  162701  000004                  sub     #4,R1                   ;  subtract crc bytes
 6772 047174  020137  003126'                 cmp     R1,buflen               ; check for length error
 6773 047200  001416                          beq     37$                     ;  if OK, br
 6774 047202  005237  002772'                 inc     s.len                   ;  else, update length errors counter
 6775 047206                                  printx  #lgerms,buflen,R1       ;  print length error message
 6776 047234  000435                          br      50$                     ;  and exit
 6777
 6778 047236  016301  000010           37$:    mov     10(R3),R1               ; point R1 to message buffer
 6779 047242  062701  000016                  add     #16,R1                  ; point R1 past header info
 6780 047246  005011                          clr     (R1)                    ; clear skip count for compare
 6781 047250  063737  001172' 002776'        add     P$SIZE,s.byte           ; update bytes compared summary counter
 6782 047256                                  CALL    DATCMP P$SIZE,CMPBUF,R1 ; check for data compare errors
 6783 047276                                  P$POP   R1                      ;  check results
 6784 047300  001413                          beq     50$                     ;   if errors,
 6785 047302  060137  002774'                 add     R1,s.comp               ;   update compare errors summary counter
 6786 047306  000410                          br      50$
 6787
 6788 047310  005237  002770'          40$:    inc     s.nrec                  ; update messages not received counter
 6789 047314  012737  017233' 001066'        mov     #noresp,keywd2          ; move 'no responce' to error indicator
 6790 047322  012702  177777                  mov     #-1,R2                  ; indicate error to R2
 6791 047326  000404                          br      60$                     ; skip to exit
 6792
 6793 047330                           50$:    CALL    RELBUF  R3              ; release buffer to DELUA/DEUNA
 6794 047340                           60$:    CALL    RETMEM                  ; restore memory mapping
 6795 047346                                   return  R2                      ; return
 6796
 6797
 6798                                  ;
 6799                                  ;ACTION ROUTINE TO SET 'RUN PATTERN' FLAG
 6800                                  ;
 6801
 6802 047352  012737  000005  001064' ACTPAT: MOV     #CPATRN,KEYWD1          ;SET FLAG
 6803 047360  000207                          RTS     PC
 6804
 6805
 6806
 6807                                  ;--+
 6808                                  ; Name - RUNPAT                         run pattern test
```

```
                                ;
                                ; Functional Desciption:
                                ;               This routine implements the NIE pattern test.  It is
                                ;               identical to the loop direct test with the exception that
                                ;               it will loop a frame containing each of the defined data
                                ;               types.
                                ;
                                ; Inputs - none
                                ;
                                ; Outputs - none
                                ;
                                ; Calling Procedure:    CALL RUNPAT
                                ;
                                ; Side Effects - none
                                ;
                                ; Subordinate Routines -
                                ;               DIRCOM  - direct loop test for each pattern
                                ;
                                ; Register Usage -
                                ;               R1      - return status of DIRCOM
                                ;
                                ;--+
047362                          RUNPAT: P$PUSH  P$TYPE                  ; save type parameter
047366  005037  001170'                 clr     P$TYPE                  ; set type to first type
047372                          5$:     CALL    dircom                  ; send messages
047400                                  P$POP   R1                      ; get results to keep stack in order
047402  001403                          beq     10$                     ; if OK, cont
047404  022701  000001                  cmp     #1,R1                   ;  else, was table empty
047410  001406                          beq     15$                     ;  if yes, return
047412  005237  001170'         10$:    inc     P$TYPE                  ; set to next type
047416  022737  000005  001170'         cmp     #5,P$TYPE               ; see if done all of them
047424  002362                          bge     5$                      ;  if not, do more
047426                          15$:    P$POP   P$TYPE                  ; restore message type
047432                                  return
                                ;
                                ;ACTION ROUTINE TO SHOW THE CURRENT MESSAGE PARAMETERS
                                ;
047434  013701  001170'         ACTSMS: MOV     P$TYPE,R1               ;GET MESSAGE TYPE INTO R1
047440  006301                          ASL     R1                      ;MULTIPLY BY 2
047442  062701  001414'                 ADD     #MSGTAB,R1              ;ADD MESSAGE TABLE OFFSET
047446                                  PRINTF  #MSGPRM                 ;PRINT MESSAGE PARAMETER MESSAGE
047466                                  PRINTF  #MSG4,(R1),P$SIZE,P$CPYS        ;PRINT PARAMETERS
047520  105037  001300'                 CLRB    P$NNUF
047524  000207                          RTS     PC
                                ;
                                ;ACTION ROUTINE TO CLEAR THE CURRENT MESSAGE PARAMETERS AND
                                ;RESET THEM TO THE DEFAULT VALUE
                                ;
047526  012737  000000  001170' ACTCMS: MOV     #ALPHA,P$TYPE           ;RESET TYPE
047534  012737  001000  001172'         MOV     #512.,P$SIZE            ;RESET SIZE
047542  012737  000001  001174'         MOV     #1,P$CPYS               ;RESET COPIES
047550                                  PRINTF  #CLRMSG                 ;PRINT MESSAGE PARAMETERS RESET MESSAGE
```

```
 6866 047570                            PRINTF  #MSG4,MSGTAB,P$SIZE,P$CPYS      ;PRINT PARAMETERS
 6867 047624  105037  001300'           CLRB    P$NNUF                  ;CLEAR NOTNUF FLAG
 6868 047630  000207                    RTS     PC
 6869
 6870
 6871                                   ;
 6872                                   ;ACTION ROUTINE TO SET SHOW COUNTERS FLAG
 6873                                   ;
 6874
 6875 047632                    ACTCNT: CALL    DEVSTART                ; start up the DELUA/DEUNA
 6876 047640                            CALL    FUNCT #RDCNTS           ;READ COUNTERS
 6877 047652                            P$POP   R1                      ;CHECK RESULT
 6878 047654  001402                    BEQ     21$                     ;BRANCH IF ERROR
 6879 047656  000137  050762'           JMP     40$
 6880                                                                   ;PRINT COUNTER INFO
 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$
 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            ;MOVE NODE TABLE ADDRESS INTO SLOT
 6933 051020                              CALL    FULSLT                  ;SEE IF TABLE EMPTY
 6934 051026  022737  177777  001202'     CMP     #-1,SLOT                ;IF YES, DON'T PRINT HEADER
 6935 051034  001510                      BEQ     15$
 6936 051036                              PRINTF  #NTBHDR                 ;PRINT NODE TABLE HEADER
 6937 051056                     10$:     CALL    FULSLT                  ;FIND LOCATION IN TABLE WITH AN ADDRESS
 6938 051064  022737  177777  001202'     CMP     #-1,SLOT                ;CHECK IF AT END OF TABLE
 6939 051072  001503                      BEQ     20$                     ;IF YES, RETURN
 6940 051074                              CALL    NTEXTI                  ;SET UP NODE TABLE INFO FOR PRINT
 6941 051102                              PRINTF  #NODADR,#STRBUF         ;PRINT CURRENT NODE ADDRESS
 6942 051126                              PRINTF  #DEFADR,#STRBU1         ;PRINT PHYSICAL ADDRESS
 6943 051152                              PRINTF  #LOGNAM,LOGVAL          ;PRINT LOGICAL NAME
 6944 051176                              PRINTF  #NETADR,AREA,DECNET     ;PRINT DECNET NODE NUMBER
 6945 051226                              PRINTF  TYPADR                  ;PRINT NODE TYPE
 6946 051246  062737  000010  001202'     ADD     #8.,SLOT                ;INCR. SLOT TO POINT TO NEXT TABLE ENTRY
 6947 051254  000700                      BR      10$                     ;CONTINUE UNTIL ALL ENTRIES PRINTED
 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$PUSH  R2                      ; save R2
 6959 051306                              CALL    REMAP   #ONTAB          ; allow access to node table
 6960 051320  013702  001270'             MOV     P$NUM,R2                ;PUT NODE LOGICAL NUMBER INTO R2
 6961 051324  006302                      ASL     R2                      ;MULTIPLY BY 8
 6962 051326  006302                      ASL     R2                      ;NODE TABLE ADDRESS =
 6963 051330  006302                      ASL     R2                      ;  (LOG. NO. X 8) + #NODTBL
 6964 051332  062702  100000             ADD     #NODTBL,R2              ;ADD OFFSET
 6965 051336  005022                      CLR     (R2)+                   ; clear ...
 6966 051340  005022                      CLR     (R2)+                   ; ... 8 byte ...
 6967 051342  005022                      CLR     (R2)+                   ; ... entry of ...
 6968 051344  005012                      CLR     (R2)                    ; ... node table
 6969 051346                              P$POP   R2                      ; restore R2
 6970 051350  105037  001300'             CLRB    P$NNUF                  ;CLEAR NOTNUF FLAG
 6971 051354                              PRINTF  #LOGDEL,P$NUM           ;PRINT MESSAGE INDICATING DELETION
 6972 051400                              CALL    RETMEM                  ; restore memory mapping
 6973 051406  000207                      RTS     PC                      ;RETURN
 6974
 6975                                     ;
 6976                                     ;ACTION ROUTINE TO INITIATE A DELUA/DEUNA PORT COMMAND
 6977                                     ;
 6978
 6979 051410  105037  001300'  ACTFCT:    CLRB    P$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$:     CALL    DEVSTOP                 ; STOP THE DELUA/DEUNA
 6986 051456  000207                            RTS     PC
 6987
 6988                                   ;
 6989                                   ;ACTION ROUTINE TO CLEAR SUMMARY TABLE
 6990                                   ;
 6991
 6992 051460  105037  001300'           ACTCSU: CLRB    P$NNUF                  ;CLEAR 'NOT ENOUGH' COUNTER
 6993 051464                                    P$PUSH  R2                      ;SAVE R2
 6994 051466                                    CALL    REMAP   #OSTAB          ;ALLOW ACCESS TO SUMMARY TABLE
 6995 051500  012702  100000                    MOV     #STATBL,R2              ;MOVE SUMMARY TABLE ADDRESS TO R2
 6996 051504  005022                    5$:     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                                   ; Inputs - 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 - node
 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'          MOV     #NODTBL,SLOT            ; point SLOT to beginning of node table
 7059 051634  013702  001202'          10$:    MOV     SLOT,R2                 ; point R2 to current node table entry
 7060 051640                                   CALL    FULSLT                  ; point SLOT  to full entry in node table
 7061 051646  022737  177777  001202'          CMP     #-1,SLOT                ; Are we at the end of the node table
 7062 051654  001522                           BEQ     30$                     ; 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 ouput
 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'                CLRB    P$NNUF                  ; clear not enough flag
 7104 052134  000207                         RTS     PC                      ; ... and return
 7105
 7106                                 ;--*
 7107                                 ; Functional Description
 7108                                 ;               This routine is designed to take a string of ascii text
 7109                                 ;               and store it on the load medium.  The file that is being
 7110                                 ;               written is assumed to be already open.
 7111                                 ;
 7112                                 ; Inputs -      P1 - Address of a character string
 7113                                 ;               P2 - Number of characters to be output to the load medium
 7114                                 ;
 7115                                 ; Outputs - outputs P2 bytes from string P1 to load medium
 7116                                 ;
 7117                                 ; Calling Procedure - CALL OUTBLK P1,P2
 7118                                 ;
 7119                                 ; Side effects - None
 7120                                 ;
 7121                                 ; Subordinate routines - None
 7122                                 ;
 7123                                 ; Register Usage -
 7124                                 ;       R1 -    pointer to character string
 7125                                 ;       R2 -    count of bytes to output
 7126                                 ;--*
 7127 052136                          OUTBLK: P$POP   R1,R2                   ; get input parameters
 7128
 7129 052142                          10$:    PUTBYT  (R1)                    ; output a byte
 7130
 7131 052150  005201                         INC     R1                      ; point R1 to next byte
 7132 052152  005302                         DEC     R2                      ; decrement number of bytes to output
 7133 052154  001372                         BNE     10$                     ; go on if there's more to do
 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(O) node entries, thus the length of the character string
 7147                                 ;               will not exceed five ("N" + 4 digits).
 7148                                 ;
 7149                                 ; Inputs - Implicit
 7150                                 ;               LOGVAL - word containing the logical node name to be formatted
```

```
7151                                    ;
7152                                    ; Outputs - Explicit
7153                                    ;               P1 - the number of characters in the formatted string
7154                                    ;
7155                                    ;          - Implicit
7156                                    ;               STRBUF  - will contain the formatted output string
7157                                    ;
7158                                    ; Calling Procedure - CALL FORLOG
7159                                    ;                     P$POP P1
7160                                    ;
7161                                    ; Side effects - STRBUF is modified
7162                                    ;
7163                                    ; Subordinate Routines - None
7164                                    ;
7165                                    ; Register Usage -
7166                                    ;       R1 -    Value to format
7167                                    ;       R2 -    scratch
7168                                    ;       R3 -    digit counter
7169                                    ;       R4 -    scratch
7170                                    ;--+
7171 052160  112737  000116  001116'  FORLOG: MOVB    #116,STRBUF             ; put an 'N' in STRBUF
7172 052166  013701  001162'                  MOV     LOGVAL,R1               ; get value to format
7173                                    ;--+
7174                                    ;       Determine how many digits are needed to represent the logical
7175                                    ;       node number.  This can be ascertained by comparing the number
7176                                    ;       to powers of eigth.  For example, if the number is less than
7177                                    ;       8-squared (100(0)), it can be represented in two digits.
7178                                    ;--+
7179 052172  012703  000001                  MOV     #1,R3                   ; there will be at least one digit
7180 052176  020127  000010                  CMP     R1,#10                  ; represent # w/ 1 digit?
7181 052202  002411                          BLT     10$                     ; 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                                    ;       Adding 60(0) to a single digit creates its ascii representation
7204                                    ;--+
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                                 ; Name - ACTUSF                 ACTION ROUTINE TO UNSAVE THE NODE TABLE
 7220                                 ;
 7221                                 ; Functional Description
 7222                                 ;               This routine is used to restore the node table from a file
 7223                                 ;               located on the load medium.  It assumes that the file will
 7224                                 ;               be in the following format:
 7225                                 ;
 7226                                 ;               CURRENT ADDRESS DEFAULT ADDRESS LOGICAL NAME DEVICE
 7227                                 ;
 7228                                 ;               The file is sequential read with each valid entry resulting
 7229                                 ;               in the addition of a node to the node table.  If a line is
 7230                                 ;               of an invalid form or it reads "empty slot", a slot in the
 7231                                 ;               node table will be left empty.  This is to preserve the
 7232                                 ;               original structure of the node table and also the correspon-
 7233                                 ;               dence of logical node names to node addresses.
 7234                                 ;
 7235                                 ; Inputs - Implicit - Address of a string that names the file is in CBOADR
 7236                                 ;        - Explicit - Takes input from a file on the load medium
 7237                                 ;
 7238                                 ; Outputs - Implicit - The node table is restored from the file
 7239                                 ;
 7240                                 ; Calling Procedure - JSR PC,ACTUSF
 7241                                 ;
 7242                                 ; Side effects - The old node table will be wiped out in lieu of the new one
 7243                                 ;
 7244                                 ; Subordinate Routines
 7245                                 ;               RDLIN - read line of an open file
 7246                                 ;               NXTDEL - find next delimiter in a string
 7247                                 ;               NXTNDL - find next non-delimiter in a string
 7248                                 ;               EDPACK - edit data frame
 7249                                 ;               ENTRND - enter node into node table
 7250                                 ;
 7251                                 ; Register Usage
 7252                                 ;               R1 -    Scratch
 7253                                 ;               R2 -    Node type - target or assist
 7254                                 ;               R3 -    Pointer to line of input from file
 7255                                 ;               R4 -    pointer to node table
 7256                                 ;
 7257                                 ;--+
 7258 052270                         ACTUSF:
 7259 052270                                 P$PUSH  R1,R2,R3,R4             ; save registers
 7260 052300                                 CALL    REMAP   #ONTAB          ; allow access to node table
 7261 052312  012704  077770                MOV     #NODTBL-10,R4           ; let R4 point to node table
 7262 052316                                 OPEN    CBOADR                  ;open file, name=asciz string
 7263 052324                                 BCOMPLETE       1$              ;return if successful
 7264 052326                                 PRINTF  #OPNERR,CBOADR          ; else print "open error"
```

```
 7265 052352  000137  053020'            JMP     30$                     ; ... and leave
 7266 052356  062704  000010      1$:    ADD     #10,R4                  ; point R4 to next node in table
 7267 052362  012703  000526'            MOV     #FILLIN,R3              ; point R3 to buffer for input line
 7268 052366                             CALL    RDLIN                   ;read a line at a time
 7269 052374                             P$POP   R1                      ; Get success of read in R1
 7270 052376  001402                     BEQ     2$                      ; non-zero means EOF
 7271 052400  000137  053020'            JMP     30$
 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                      3$:    CALL    NXTNDL  R3              ; Point R3 to current address
 7279 052446                             P$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$:   MOV     ADRBUF,(R4)             ; first two bytes
 7295 052506  013764  001072'  000002     MOV     ADRBUF+2,2(R4)          ; second two bytes
 7296 052514  013764  001074'  000004     MOV     ADRBUF+4,4(R4)          ; last two bytes
 7297
 7298 052522                      21$:   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  1. 761  177777  000001  26$:     MOVB    #-1,1(R1)               ; move unknown identifier into table
7355 053014  000137  052356'           JMP     1$
7356
7357 053020                    30$:     CLOSE                           ; close the open file
7358 053022                             P$POP   R1,R2,R3                ; restore registers
7359 053030                             RETURN
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                          15$:    RETURN  R1                      ; return results
 7382 053112  013737  001070' 001076' ACTSOU: MOV     ADRBUF,SOUFIL           ; store 6 bytes of source filter
 7383 053120  013737  001072' 001100'         MOV     ADRBUF+2,SOUFIL+2       ;
 7384 053126  013737  001074' 001102'         MOV     ADRBUF+4,SOUFIL+4       ;
 7385 053134  112737  177777  001253'         MOVB    #-1,SOUFLG              ; set source filter presence flag
 7386 053142  105037  001300'                 CLRB    P$NNUF                  ; clear not enough flag
 7387 053146  000207                          RTS     PC
 7388
 7389 053150  013737  001070' 001104' ACTDES: MOV     ADRBUF,DESFIL           ; store 6 bytes of destination filter
 7390 053156  013737  001072' 001106'         MOV     ADRBUF+2,DESFIL+2       ;
 7391 053164  013737  001074' 001110'         MOV     ADRBUF+4,DESFIL+4       ;
 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              ; set listen command flag
 7398 053214  105037  001300'                 CLRB    P$NNUF                  ; clear "not enough" flag
 7399 053220  000207                          RTS     PC
 7400
 7401 053222  004737  053322' ACTPRO: JSR     PC,XSTRIN               ; Put protocol type in CBOBUF
 7402 053226                          CALL    EDPACK  #CBOBUF,#PROFIL,#2 ;STORE PROTOCOL FILTER
 7403 053250                          P$POP   R0                      ; get return status
 7404 053252  105700                  TSTB    R0                      ; was this a successful call?
 7405 053254  001416                  BEQ     5$                      ; yes, take off!
 7406 053256                          PRINTF  #CPROER                 ; else print error
 7407 053276  105037  001300'         CLRB    P$NNUF                  ; clear "not enough" flag
 7408 053302  112737  177777  001301'         MOVB    #-1,P$GDBD              ; set bogus command flag
 7409 053310  000403                  BR      10$                     ; exit!
 7410 053312  112737  177777  001255' 5$:     MOVB    #-1,PROFLG              ; set protocol filter presence flag
 7411 053320  000207                  10$:    RTS     PC
 7412
 7413 053322                          XSTRIN: P$PUSH  R1,R2,R3                ; save these registers
 7414 053330  013701  001166'         MOV     CBOADR,R1               ; get address of string to extract
 7415 053334  012702  001042'         MOV     #CBOBUF,R2              ; get address of buffer to hold it
 7416 053340  121127  000057  10$:    CMPB    (R1),#057               ; Is this char. a "/"?
 7417 053344  001407                  BEQ     20$                     ; Yes!!
 7418 053346  121127  000054         CMPB    (R1),#054               ; 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                  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                                   ;               EOFF    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                                   ; THE FOLLWING TWO LINES ARE EQUIVALENT TO DRS GETBYTE CALL.  THEY HAVE
 7461                                   ; ERROR RIGHT NOW -- SHOULD DO A MOVB AND THEY ARE DOING A MOV OF RESULT
 7462                                   ;*************************************************************************
 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                                   ; Name - SELMSG                         OPERATOR SELECTED MESSAGE STORAGE
 7492                                   ;
```

```
 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                   1$:     TSTB    (R3)            ; look for end of string
 7523 053506  001405                           BEQ     3$              ; 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                   3$:     SUB     R1, R3          ; calculate number of bytes
 7529 053524                                   CALL    HXFORM  R1, #OPSLBF, R3 ; convert to hex
 7530 053542                                   P$POP   R0,R4           ; get return status
 7531 053546  001420                           BEQ     12$             ; branch if success
 7532 053550  112737  177777  001301'          MOVB    #-1,P$GDBD      ; set error flag
 7533 053556                                   ERRSOFT 38,EMSG44
 7534 053566  000412                           BR      13$
 7535 053570                           4$:
 7536 053570  005003                           CLR     R3              ;CLEAR CHARACTER COUNTER
 7537 053572  105711                   5$:     TSTB    (R1)            ;CHECK FOR END OF STRING
 7538 053574  001403                           BEQ     10$             ;GO TO 10$ IF END
 7539 053576  112122                           MOVB    (R1)+,(R2)+     ;ELSE, MOVE BYTE TO OUTPUT BUFFER
 7540 053600  005203                           INC     R3              ;COUNT NUMBER OF CHARACTERS IN INPUT BUFFER
 7541 053602  000773                           BR      5$              ;GO DO MORE CHARACTERS
 7542 053604  112712  000000           10$:    MOVB    #0,(R2)         ;PUT ZERO AT END OF OUTPUT BUFFER
 7543 053610  010337  001446'          12$:    MOV     R3,MSG6C        ;STORE NUMBER OF CHARACTERS FOR USE IN BUF. BUILDING
 7544 053614                           13$:    RETURN
 7545
 7546                                   ;--*
 7547                                   ; Name - ENTRND                                 ENTER NODE IN TABLE
 7548                                   ;
 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 TRANSFERED
 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                                   ;--+
 7594                                   ; Name - FINDSL                         FIND EMPTY SLOT IN NODE TABLE
 7595                                   ;
 7596                                   ; Functional Description
 7597                                   ;               This routine is used to find an empty slot in the node table.
 7598                                   ;
 7599                                   ; Inputs - none
 7600                                   ;
 7601                                   ; Outputs - Explicit -
 7602                                   ;               P1 - zero if found a slot, -1 if no room in the node table
 7603                                   ;
 7604                                   ;            Implicit -
 7605                                   ;               SLOT - contains address of empty slot in node table
 7606                                   ;
```

```
 7607                                   ; Calling Procedure: CALL FINDSL
 7608                                   ;                    P$POP P1
 7609                                   ;
 7610                                   ; Side Effects - none
 7611                                   ;
 7612                                   ; Subordinate Routines -
 7613                                   ;               REMAP  - map node table into memory
 7614                                   ;               RETMEM - restore memory mapping
 7615                                   ;
 7616                                   ; Register Usage -
 7617                                   ;               R2 - pointer into node table
 7618                                   ;
 7619                                   ;--+
 7620 053714                            FINDSL: CALL    REMAP   @ONTAB          ;ALLOW ACCESS TO NODE TABLE
 7621 053726  012702  100000                    MOV     @NODTBL,R2              ;MOVE ADDRESS OF NODE TABLE TO R2
 7622 053732  022712  000000            10$:    CMP     @0,(R2)                 ;SEE IF SLOT EMPTY
 7623 053736  001422                            BEQ     20$                     ;IF YES, BR 20$
 7624 053740  062702  000010                    ADD     @8.,R2                  ;ELSE MOVE POINTER TO NEXT ENTRY LOC.
 7625 053744  020227  110000                    CMP     R2,@NODEND              ;SEE IF AT END OF NODE TABLE
 7626 053750  001370                            BNE     10$                     ;IF NOT, CONTINUE LOOKING
 7627 053752                                    PRINTF  @TABFUL,@NOD            ;ELSE, PRINT TABLE FULL MESSAGE
 7628 053776                                    P$PUSH  @-1                     ;PUT TABLE FULL INDICATION ON STACK
 7629 054002  000404                            BR      30$                     ;RETURN
 7630 054004  010237  001202'      20$:    MOV     R2,SLOT                 ;MOVE ADDRESS OF EMPTY LOC. INTO SLOT
 7631 054010                                    P$PUSH  @0                      ;PUT LOC. FOUND INDICATION ON STACK
 7632 054014                            30$:    CALL    RETMEM                  ;RESTORE MEMORY MAPPING
 7633 054022                                    RETURN                          ;RETURN
 7634
 7635                                   ;--+
 7636                                   ; Name - FULSLT                                        FULL SLOT ROUTINE
 7637                                   ;
 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                            FULSLT: CALL    REMAP   @ONTAB  ;ALLOW ACCESS TO NODE TABLE
 7661 054036  013701  001202'                   MOV     SLOT,R1         ;MOVE SLOT LOCATION TO R1
 7662 054042  020127  110000            10$:    CMP     R1,@NODEND      ;SEE IF AT END OF NODE TABLE
 7663 054046  001406                            BEQ     15$             ;IF YES, BR 15$
```

```
 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          20$:    ADD     #8.,R1          ;INCR. POINTER TO NEXT LOCATION
 7671 054100  000760                          BR      10$             ;CHECK NEXT LOC.
 7672 054102                          30$:    CALL    RETMEM          ;RESTORE MEMORY MAPPING
 7673 054110                                  RETURN                  ;RETURN
 7674
 7675                                 ;--+
 7676                                 ; Name - CMPTWO                         COMPAIR TWO BUFFERS
 7677                                 ;
 7678                                 ; Functional Description
 7679                                 ;               This routine does a word by word comparison of two buffers
 7680                                 ;               of arbitrary length.  It will report the likeness of the
 7681                                 ;               two buffers.
 7682                                 ;
 7683                                 ; Inputs - Explicit -
 7684                                 ;               P1 - address of first buffer
 7685                                 ;               P2 - address of second buffer
 7686                                 ;               P3 - number of words to compare
 7687                                 ;
 7688                                 ; Outputs - Explicit -
 7689                                 ;               P4 - 0 = buffers contained exact same data; -1 = they differed
 7690                                 ;
 7691                                 ; Calling Procedure: CALL CMPTWO P1,P2,P3
 7692                                 ;                    P$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                                   ;               set up a "record" of information on that particular node.
 7722                                   ;               Included in the information will be: current physical address,
 7723                                   ;               default physical address, device type attached to the node,
 7724                                   ;               logical node name, and DECnet address (AREA.NODE_NUMBER).
 7725                                   ;
 7726                                   ; Inputs - Implicit -
 7727                                   ;               SLOT - contains address of node to work on
 7728                                   ;
 7729                                   ; Outputs - Implicit -
 7730                                   ;               STRBUF - contains current physical address of node
 7731                                   ;               STRBU1 - contains default physical address of node
 7732                                   ;               LOGVAL - integer representing logical node number
 7733                                   ;               DECNET - DECnet node number
 7734                                   ;               AREA   - DECnet area number
 7735                                   ;
 7736                                   ; Calling Procedure: CALL NTEXTI
 7737                                   ;
 7738                                   ; Side Effects - none
 7739                                   ;
 7740                                   ; Subordinate Routines -
 7741                                   ;               BINHEX - convert node address into ascii string
 7742                                   ;               GETTYP - set device type attached to node
 7743                                   ;               REMAP  - map node table into memory
 7744                                   ;               RETMEM - restore memory mapping
 7745                                   ;
 7746                                   ; Register Usage -
 7747                                   ;               R1, R2, R3 - scratch
 7748                                   ;
 7749                                   ;--*
 7750 054144                           NTEXTI:
 7751                                   ;--*
 7752                                   ;       Setup the current node address in the buffer STRBUF
 7753                                   ;--*
 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
 7762 054200  013703  001202'                  MOV     SLOT,R3                 ;GET POINTER TO NODE TABLE
 7763 054204  062703  010000                   ADD     @DEFNOD,R3              ;POINT R3 TO DEFAULT HARDWARE ADDR.
 7764 054210                                   CALL    BINHEX  R3,#6,@STRBU1   ;CONVERT BINARY ADDRESS TO ASCII
 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
 7777 054244  013702  001202'                  MOV     SLOT,R2                 ;POINT R2 TO NODE TABLE
```

```
 7778 054250  162702  100000                    SUB     #NODTBL,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$:      ADD     #2, R1                  ; point to decnet address
 7796 054322  011137  002054'                   MOV     (R1),DECNET             ; and buffer it
 7797 054326  042737  176000  002054'           BIC     #176000,DECNET          ;clear area number
 7798 054334  011137  002056'                   MOV     (R1), AREA
 7799 054340  042737  001777  002056'           BIC     #1777,AREA              ;clear node number
 7800 054346  012701  000012                    MOV     #10.,R1
 7801 054352                         35$:
 7802 054352  006037  002056'                   ROR     AREA                    ;shift it into position for print
 7803 054356  005301                            DEC     R1
 7804 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$pop   R1                      ; Get address of system id
7837 054366                                  CALL    REMAP   #ORRING         ; allow access to receive ring
7838 054400  010137  003110'                mov     R1,temp                 ; save it in TEMP
7839
7840 054404  062701  000006                add     #sourcc,R1              ; point R1 to source address
7841 054410                                  call    binhex  R1,#6,#strbuf   ; put address in strbuf
7842 054430                                  printf  #simsg1,#strbuf         ; print remote node current address
7843
7844 054454  013701  003110'                mov     temp,R1                 ; restore address of system id
7845 054460  016137  000016  003112'        mov     siccou(R1),temp1        ; save char. count
7846 054466  162737  000004  003112'        sub     #4,temp1                ; skip code, pad, and receipt number
7847
7848 054474                                  call    getida  temp,#144       ; get address of device type
7849 054512                                  p$pop   R2                      ; save address in R2
7850 054514                                  PRINTF  #SIMSG7                 ; print device field label
7851 054534                                  call    GETTYP  R2              ; get the device type
7852 054544                                  PRINTF  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  @smsg10,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  @smsg11,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  @smsg12,@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  @smsg13,4(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                          ;       The field that is being looked at is relevant only to POSEIDON
 7946                          ;       communication servers at present.  If further COM devices make use
 7947                          ;       of this field then this section will have to be expanded accordingly
 7948                          ;--+
```

```
 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$:   CMP     (R1),#106.              ; Is this the Server's Location?
 8014 055734  001030                        BNE     POSEXIT                 ; NO, didn't find match ... just exit
 8015 055736                                PRINTF  #POSLOC                 ; print field identifier message
 8016
 8017                                ;--+
 8018                                ;       The value for these fields are represented as counted ascii strings.
 8019                                ;       The length of the string is just the INFO LENGTH field of the particular
 8020                                ;       system ID field.  To allow the printing of the string, attach a NULL
 8021                                ;       byte to the end of it
 8022                                ;--+
 8023 055756  062701  000004          60$:   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                                ; Functional Description
 8036                                ;               This subroutine takes a system id message and a field type
 8037                                ;               identifier and searches for the specific field.  It returns
 8038                                ;               the address of the value for the given field.
 8039                                ;
 8040                                ; Inputs -      P1 - address of a buffer holding a system id message
 8041                                ;               P2 - field type identifier to search for
 8042                                ;
 8043                                ; Outputs -     P3 - address of the value for the given field
 8044                                ;                    If no match is found, zero is returned
 8045                                ;
 8046                                ; Calling procedure - call GETIDA P1,P2
 8047                                ;
 8048                                ; Side effects -
 8049                                ;               1.) This routine leaves the receive ring mapped into KPAR4,5
 8050                                ;
 8051                                ; Register Usage - R1 - points to buffer that holds the system id message
 8052                                ;                  R2 - holds field type identifier to look for
 8053                                ;                  R3 - holds character count of message
 8054                                ;
 8055                                ;--+
 8056
 8057 056020                        GETIDA:
 8058 056020                                p$pop   R1,R2                   ; get address of string to search for
 8059 056024                                p$push  temp                    ; need a temporary var., so save 'temp'
 8060 056030                                CALL    REMAP   #ORRING         ; allow access to receive ring
 8061
 8062 056042  016103  000016                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                                                                        ; receipt number
 8065 056052  062701  000024                add     #siffid,R1              ; point R1 to first field ID
 8066
 8067 056056  012704  003110'       10$:    mov     #temp,R4                ; let R4 point to temporary storage
 8068 056062  112124                movb    (R1)+,(R4)+             ; save a byte of field identifier
 8069 056064  112124                movb    (R1)+,(R4)+             ; save a byte of field identifier
 8070 056066  023702  003110'       cmp     temp,R2                 ; have we found the desired field?
 8071 056072  001412                beq     20$                     ; yes, return it
 8072
 8073 056074  112104                movb    (R1)+,R4                ; get byte that has length field
 8074
 8075 056076  162703  000003                sub     #3,R3                   ; decrement character count for fields
 8076 056102  160403                sub     R4,R3                   ;
 8077 056104  001003                bne     15$                     ; keep going if more characters
 8078 056106  012701  000000                mov     #0,R1                   ; didn't find it
 8079 056112  000404                br      22$                     ; return error indicator
 8080
 8081 056114  060401        15$:    add     R4,R1                   ; let R1 point to next field
 8082 056116  000757                br      10$                     ; continue to look
 8083
 8084 056120  062701  000001  20$:    add     #1,R1                   ; point R1 to field value
 8085 056124                22$:    p$pop   temp                    ; restore value in 'temp'
 8086 056130                        return  R1                      ; return address
 8087
 8088                       .sbttl  PRTTYP  print the device type
 8089
 8090                       ;---+
 8091                       ;       PRTTYP                          PRINT DEVICE TYPE
 8092                       ;
 8093                       ;       INPUTS                  P1 - ADDRESS OF A BYTE THAT IS NODE TYPE
 8094                       ;       EXPLICIT OUTPUTS        NONE
 8095                       ;       IMPLICIT OUTPUTS        THE NODE TYPE WILL BE PRINTED IN PSEUDO-ENGLISH
 8096                       ;       SUBORDINATE ROUTINES    NONE
 8097                       ;       CALLING SEQUENCE        CALL PRTTYP P1
 8098                       ;
 8099                       ;---+
 8100 056134                GETTYP:
 8101 056134                        P$POP   R2                      ; get address node type
 8102 056136  122712  000001        CMPB    #IDTUNA,(R2)            ; DELUA/DEUNA?
 8103 056142  001004                BNE     50$                     ; branch if not
 8104 056144  012737  012746' 001164'       MOV     #UNA,TYPADR             ; save una description
 8105 056152  000446                BR      100$                    ; leave
 8106 056154  122712  000005        50$:    CMPB    #IDTQNA,(R2)            ; QNA?
 8107 056160  001004                BNE     60$                     ; branch if not
 8108 056162  012737  012756' 001164'       MOV     #QNA,TYPADR             ; save qna description
 8109 056170  000437                BR      100$                    ; leave
 8110 056172  122712  000011        60$:    CMPB    #IDTLUA,(R2)            ; LUA?
 8111 056176  001004                BNE     70$                     ; branch if not
 8112 056200  012737  012766' 001164'       MOV     #LUA,TYPADR             ; save LUA description
 8113 056206  000430                BR      100$                    ; leave
 8114 056210  122712  000003        70$:    CMPB    #IDTCNA,(R2)            ; CNA?
 8115 056214  001004                BNE     80$                     ; branch if not
 8116 056216  012737  012776' 001164'       MOV     #CNA,TYPADR             ; save CNA description
 8117 056224  000421                BR      100$                    ; leave
 8118 056226  122712  000013        80$:    CMPB    #IDTCSA,(R2)            ; CSA?
 8119 056232  001004                BNE     90$                     ; 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' 95$:    MOV     #UNKNWN,TYPADR          ; save 'unknown' description
 8127 056270                          100$:   RETURN
 8128
 8129
 8130                                ;--*
 8131                                ; Name - EXELIS                         Execute the Listen Command
 8132                                ;
 8133                                ; Functional Description
 8134                                ;               This routine implements the LISTEN command of the NIE.
 8135                                ;               The purpose of the LISTEN command is to be able to monitor
 8136                                ;               the activity of nodes on a network.
 8137                                ;                       Listening on the network consists of receiving
 8138                                ;               all frames that pass a user specified filter.  The filter
 8139                                ;               may be on the frame's destination address, source address,
 8140                                ;               protocol type, or any combination of the three.
 8141                                ;                       A log will be kept containing information on frames
 8142                                ;               that pass the filter(s) including: destination address,
 8143                                ;               source address, protocol type, packet length, and number
 8144                                ;               of receipts.  If a frame's characteristics match the first
 8145                                ;               four then the number of receipts counter is incremented.
 8146                                ;               A maximum of 30 entries will be stored in the log.
 8147                                ;                       A list of source addresses of frames that pass the
 8148                                ;               filters will also be kept along with a count of the number
 8149                                ;               of times that source address has been heard from
 8150                                ;                       The routine will print information on frames that pass
 8151                                ;               filters every one millisecond or if there are no frames
 8152                                ;               outstanding in the receive ring.
 8153                                ;                       The only way to stop listening is to type a control-C.
 8154                                ;
 8155                                ; Inputs - none
 8156                                ;
 8157                                ; Outputs - Implicit
 8158                                ;               LISLOG - log containing frame characteristics
 8159                                ;               LISNUM - the number of times the LISTEN command has been
 8160                                ;                        entered since the log has been cleared
 8161                                ;               LISSEC - total number of seconds of listening
 8162                                ;               LISMIN - total nubmer of minutes of listerning
 8163                                ;               LISFSC - seconds to fill log
 8164                                ;               LISFMN - minutes to fill log
 8165                                ;               ADRLIS - source address list
 8166                                ;
 8167                                ; Calling Procudure: JSR PC,EXELIS
 8168                                ;
 8169                                ; Side Effects -
 8170                                ;               1.) control will pass to the DRS upon control-C
 8171                                ;
 8172                                ; Subordinate Routines -
 8173                                ;               CMPTWO - buffer comparison
 8174                                ;               RECEVE - receive frames
 8175                                ;               PRLENT - print a listen event
 8176                                ;
```

```
 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                ; start up the DELUA/DEUNA
 8187 056300  012702  002566'                   MOV     #$WDMO,R2               ; get address of PCB for write mode
 8188 056304  012762  100000  000002           MOV     #100000,2(R2)           ; set promiscuous mode bit
 8189 056312                                    CALL    FUNCT   #WDMODE         ; execute write mode port command
 8190 056324                                    P$POP   R2                      ; get error status
 8191 056326  001404                            BEQ     5$                      ; no error, continue
 8192 056330                                    ERRDF   39,EMSG23,ERR1          ; report error
 8193
 8194 056340  105737  001234'          5$:     TSTB    LISNUM                  ; Is this the first listen?
 8195 056344  001007                            BNE     10$                     ; no, don't initialize
 8196 056346  005037  001242'                   CLR     LISMIN                  ; reset minutes since start
 8197 056352  005037  001244'                   CLR     LISSEC                  ; reset seconds since start
 8198 056356  012737  000001  002052'          MOV     #1,TIMERS               ; set print out for every millisecond
 8199
 8200 056364  013737  001242'  002040' 10$:     MOV     LISMIN,TIMMIN           ; reset value that clock serv. routine shngles
 8201 056372  013737  001244'  002042'          MOV     LISSEC,TIMSEC           ;
 8202 056400                                    PRINTF  #LISHD1                 ; print listen header
 8203 056420                                    PRINTF  #NEWLI1                 ; CR-LF
 8204 056440  105237  001234'                   INCB    LISNUM                  ; update number of listens
 8205
 8206 056444                            20$:    BREAK                           ; allow for contol-c interruption
 8207 056446                                    CALL    RECEVE                  ; see if any frames have arrived
 8208 056454                                    P$POP   R2                      ; R2 positive means yes
 8209 056456  001772                            BEQ     20$                     ; didn't get anything, keep looking
 8210
 8211 056460                            25$:    BREAK                           ; allow for control-c interruption
 8212 056462  013737  002040'  001242'          MOV     TIMMIN,LISMIN           ; update total minutes and seconds
 8213 056470  013737  002042'  001244'          MOV     TIMSEC,LISSEC           ; since start of listen
 8214 056476  013703  002100'                   MOV     RRGNXT,R3               ; get receive ring pointer
 8215 056502                                    CALL    GETRNX,#RRGNXT          ; update receive next pointer
 8216 056514  016337  000006  001240'          MOV     6(R3),LBYTEC            ; save message buffer length
 8217 056522  042737  170000  001240'          BIC     #170000,LBYTEC          ; clear status bits
 8218 056530  016302  000010                    MOV     10(R3),R2               ; point R3 to message buffer
 8219
 8220                                   ;--*
 8221                                   ;       Test to see if the received frame passes the user specified filters
 8222                                   ;--*
 8223
 8224 056534  105737  001254'                   TSTB    DESFLG                  ; see if a dest. filter has been specified
 8225 056540  001412                            BEQ     40$                     ; no dest. filter
 8226 056542                                    CALL    CMPTWO  R2,#DESFIL,#3   ; check against filter
 8227 056562                                    P$POP   R1                      ; get equals indicator
 8228 056564  001036                            BNE     55$                     ; not equal, don't proceed!
 8229
 8230 056566  062702  000006            40$:    ADD     #SOURCC,R2              ; point R2 to source address of received frame
 8231 056572  105737  001253'                   TSTB    SOUFLG                  ; see if source filter has been specified
 8232 056576  001412                            BEQ     50$                     ; no source filter
 8233 056600                                    CALL    CMPTWO  R2,#SOUFIL,#3   ; check against filter
```

```
 8234 056620                                P$POP   R1                      ; get equals indicator
 8235 056622  001017                        BNE     55$                     ; not equal, don't proceed
 8236
 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                        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$:    INC     LPACNM                  ; incremnet number of frames that passed filter
 8253
 8254                               ;--+
 8255                               ;       Now we've got a frame that has made it through the specified filters.
 8256                               ;       R3 points to the buffer that contains the frame.  Log information in
 8257                               ;       listen log and address list.
 8258                               ;
 8259                               ;       If all four fields - destination, source, protocol type, and character
 8260                               ;       count - match an entry in the listen log, update the count for that
 8261                               ;       entry.  If not and there is room in the log, make a new entry.
 8262                               ;--+
 8263 056702  012704  100000                MOV     #LISLOG,R4              ; point R4 to listen log
 8264 056706  016302  000010                MOV     10(R3),R2               ; point R2 to receive buffer
 8265
 8266                               ;--+
 8267                               ;       NOTE: the listen log has been set up such that individual entries have
 8268                               ;       fields that are in the same relative locations as those in the received
 8269                               ;       frame.
 8270                               ;--+
 8271
 8272 056712  020437  001232'      70$:    CMP     R4,LISNXT               ; have we checked all entries?
 8273 056716  001434                        BEQ     85$                     ; yes, try to add a new entry
 8274 056720                                CALL    CMPEXT  #ORRING,R2,#OLLOG,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   #OLLOG          ; 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  @ORRING,R2,@OLLOG,R4,#7 ; move dest., source., and p.t. into log
 8295 057044                                    CALL    REMAP   @OLLOG          ; 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  @ORRING,R2,@OLLOG,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   @OLLOG          ; 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  @ORRING,R2,@OLLOG,R4,#3 ; store source address
 8331 057264                                    CALL    REMAP   @OLLOG          ; 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 nothings there, then print out the information from the
 8339                                   ;       last frame processed.
 8340                                   ;--+
 8341
 8342 057312                          140$:     CALL    RECEVE                  ; See if anything's arrived
 8343 057320                                    P$POP   R2                      ; R2 is nozero 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
```

```
 8348 057332  012737  000001  002052'          MOV     #1,TIMERS               ; reload timer
 8349
 8350 057340                          150$:     CALL    PRLENT  10(R3),LBYTEC   ; sho user what we have!
 8351
 8352 057356                          160$:     CALL    RELBUF  R3              ; release recieve buffer
 8353 057366  000137  056444'                   JMP     20$                     ; and keep it going
 8354                                 ;--+
 8355                                 ;       Can't get out of this routine other than by control-C
 8356                                 ;--+
 8357
 8358
 8359 057372                          ACTSLI: P$PUSH  R2,R3                   ; Save R2
 8360 057376                                  CALL    REMAP   #OLLOG          ; allow access to listen log
 8361 057410  023727  001232' 100000          CMP     LISNXT,#LISLOG          ; Are there any entries in the log
 8362 057416  001021                          BNE     5$                      ; yes, go print there contents
 8363 057420                                  PRINTF  #LEMSG                  ; NO, print log empty message
 8364                                 ;
 8365                                 ; Right here we know that the address list must be empty also, so just print
 8366                                 ; "address list empty" message
 8367                                 ;
 8368 057440                                  PRINTF  #ALEMPT                 ; print empty message
 8369 057460  000535                          BR      50$                     ; don't bother going on
 837C
 8371 057462                          5$:     PRINTF  #LISHD1                 ; print listen log header ...
 8372 057502                                  PRINTF  #LISHD2                 ;  ... more header
 8373 057522  012702  100000                  MOV     #LISLOG,R2              ; let R2 point to beginning of listen log
 8374 057526  020237  001232'         10$:    CMP     R2,LISNXT               ; have we finished printing log?
 8375 057532  001424                          BEQ     20$                     ; YES!!
 8376 057534  016203  000016                  MOV     LBCOU(R2),R3            ; put message length in R3
 8377 057540                                  CALL    PRLENT  R2,R3           ; print entry pointed to by R2
 8378 057552                                  PRINTF  #LCOUNT,LISCOU(R2)      ; print the number of times this message
 8379                                                                         ; was received.
 8380 057576  062702  000022                  ADD     #LISENT,R2              ; point R2 to next entry
 8381 057602  000751                          BR      10$                     ;
 8382
 8383 057604  105737  001252'         20$:    TSTB    LISFUL                  ; see if listen log was filled
 8384 057610  001414                          BEQ     30$                     ; NO, IT WEREN'T
 8385 057612                                  PRINTF  #LFMSG,LOGFMN,LOGFSC    ; print log filled message
 8386
 8387 057642                          30$:    PRINTF  #ALHDR                  ; print address list header
 8388 057662  012702  101034                  MOV     #ADRLIS,R2              ; let R2 point to beginning of addr. list
 8389
 8390 057666  020237  001256'         40$:    CMP     R2,ADRNXT               ; done printing list?
 8391 057672  001430                          BEQ     50$                     ; YAA!
 8392 057674                                  CALL    BINHEX  R2,#6,#STRBUF   ; convert address pointed to by R2 to HEX
 8393 057714  0627C2  000006                  ADD     #6,R2                   ; point R2 to "#-of-times"
 8394 057720                                  PRINTF  #AADDR,#STRBUF,(R2)     ; print this info
 8395 057746  062702  000002                  ADD     #2,R2                   ; point R2 to next entry
 8396 057752  000745                          BR      40$                     ;
 8397
 8398                                 ; Now print total listen time and number of listen commands
 8399 057754                          50$:    PRINTF  #LTMSG,LISMIN,LISSEC,LISNUM
 8400
 8401 060010                                  P$POP   R2,R3                   ; restore R2 and R3
 8402 060014  105037  001300'                 CLRB    P$NNUF                  ; clear not enough flag
 8403 060020                          60$:    CALL    RETMEM                  ; restore memory mapping
 8404 060026  000207                          RTS     PC                      ; RETURN
```

```
 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                                   ; Name - PRLENT
 8426                                   ;
 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                                   ;
 8432                                   ; Inputs -      P1 - A pointer to an entry in the listen log or to a message
 8433                                   ;                    buffer.
 8434                                   ;               P2 - The length of the entry or message
 8435                                   ;
 8436                                   ; Outputs - none
 8437                                   ;
 8438                                   ; Calling procedure - CALL PRLENT P1,P2
 8439                                   ;
 8440                                   ; Side effects - Information about the frame/listen log entry is printed at
 8441                                   ;                the user's terminal.
 8442                                   ;
 8443                                   ; Subordinate Routines -
 8444                                   ;               BINHEX - convert binary to an ASCII HEX string
 8445                                   ; Register Usage -
 8446                                   ;               R2 - pointer to buffer that contains dest., source, and protocol
 8447                                   ;                    type
 8448                                   ;               R3 - contains the length of the message
 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                                ; Name - MEMMAP
 8466                                ;
 8467                                ; Functional Description
 8468                                ;               All the CPUs that this diagnostic runs on have at
 8469                                ;               least an 18-bit bus providing for at least 128kW of
 8470                                ;               physical memory.  Of this memory, only 32kW are strictly
 8471                                ;               allocated for the diagnostic.  But, there is another 32kW
 8472                                ;               block that is available to the diagnostic by requesting
 8473                                ;               its use from the DRS.  The management of the memory is
 8474                                ;               supposed to be done by the DRS.  With the nature of this
 8475                                ;               diagnostic, speed being of the essence, it has become
 8476                                ;               necessary for me to skirt the DRS and handle the management
 8477                                ;               of this extended memory.
 8478                                ;                       This routine will check with the DRS first to make
 8479                                ;               sure that the extended memory exists.  It then will format
 8480                                ;               the extended memory in the following manner.
 8481                                ;
 8482                                ;                +-----------------------------------+
 8483                                ;                )  FUTURE USE                       ) 377776
 8484                                ;                )                                   ) 360000
 8485                                ;                +-----------------------------------+
 8486                                ;                )  LISTEN LOG AND ADDRESS LIST FOR  ) 357776
 8487                                ;                )  LISTEN COMMAND                   ) 340000
 8488                                ;                +-----------------------------------+
 8489                                ;                )                                   ) 337776
 8490                                ;                )                                   )
 8491                                ;                .  SUMMARY TABLE                    .
 8492                                ;                )                                   )
 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 KPAR5 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 KPAR5 which are pointing to extended memory.
 8515                                ;               (which, for obvious reasons, would completely screw everything
 8516                                ;               up).
 8517                                ;
 8518                                ; Inputs - none
```

```
 8519                                   ;
 8520                                   ; Outputs - none
 8521                                   ;
 8522                                   ; Calling Procedure: CALL MEMMAP
 8523                                   ;
 8524                                   ; Side Effects -
 8525                                   ;               1.) If the call to the DRS returns successfully, then
 8526                                   ;                   extended memory will be formatted as above
 8527                                   ;
 8528                                   ;               2.) If the call to the DRS fails, indicating that there
 8529                                   ;                   is no extended memory, then the diagnostic will be
 8530                                   ;                   aborted.
 8531                                   ;
 8532                                   ; Subordinate routines -
 8533                                   ;               REMAP  - used to remap memory so that the transmit ring may be
 8534                                   ;                        accessed
 8535                                   ;               RETMEM - used to return the mapping of memory to its original
 8536                                   ;                        state
 8537                                   ;
 8538                                   ; Register Usage -
 8539                                   ;
 8540                                   ;
 8541                                   ;--+
 8542 060336                           MEMMAP::
 8543 060336                                   MMU     OFF                     ; let diagnostic control MMU
 8544
 8545
 8546                                   ;--+
 8547                                   ;       This diagram shows the structure of the transmit and receive rings
 8548                                   ;       note RING_BASE+10 is defined by this program.  It is the virtual address
 8549                                   ;       of the buffer associated with the particular entry.  In the DELUA/DEUNA
 8550                                   ;       documentation it is reserved for the port driver.
 8551                                   ;
 8552                                   ;       +----------------------+
 8553                                   ;       )  Segment length      )
 8554                                   ;       )                      ) RING_BASE+0
 8555                                   ;       +----------------------+
 8556                                   ;       )  Segment physical    )
 8557                                   ;       )  address             ) RING_BASE+2
 8558                                   ;       +----------------------+
 8559                                   ;       )  Status              )
 8560                                   ;       )                      ) RING_BASE+4
 8561                                   ;       +----------------------+
 8562                                   ;       )  Status & TDR/MLEN   )
 8563                                   ;       )                      ) RING_BASE+6
 8564                                   ;       +----------------------+
 8565                                   ;       )  Segment virtual     )
 8566                                   ;       )  address             ) RING_BASE+10
 8567                                   ;       +----------------------+
 8568                                   ;--+
 8569
 8570                                   ;--+
 8571                                   ;       Now build the receive ring.  There will be eight entries in
 8572                                   ;       the ring.  The receive buffers follow directly after the receive
 8573                                   ;       ring or 120(0) away from the start of this segment of memory.
 8574                                   ;--+
 8575 060344                                   CALL    REMAP   @ORRING         ; enable access to portion of memory
```

```
                                                                        ; that has receive ring and buffers
                                                                        ;
                                                                        ;
060356  012701  100120                MOV     #RBUFV1,R1              ; R1 has virt. addr. of first buffer
060362  012702  100000                MOV     #RRING,R2               ; R2 has base address of receive ring
060366  012703  000120                MOV     #R11501,R3              ; R3 points to the first receive buffer
060372  012704  000010                MOV     #NO.NRR,R4              ; R4 has count of receive ring entries
                                                                        ;
060376  012722  002756        20$:    MOV     #RPKLEN,(R2)+           ; Set up length of segment (1518(D))
060402  010322                        MOV     R3,(R2)+                ; store address <15:01> of SEGB
060404  012722  000001                MOV     #R11716,(R2)+           ; store address <17:16> of SEGB
060410  005722                        TST     (R2)+                   ; leave room for buffer length
060412  010122                        MOV     R1,(R2)+                ; store virtual addr. of SEGB
060414  062701  002756                ADD     #RPKLEN,R1              ; point R1 to next receive buffer
060420  062703  002756                ADD     #RPKLEN,R3              ; point R3 to next receive buffer
060424  005304                        DEC     R4                      ; decrement loop control
060426  001363                        BNE     20$                     ; keep going if more to do
                                                                        ;
                                                                        ;
                                ;--+                                    ;
                                ;                                       ; Now build transmit ring and buffers.  There will be two entries
                                ;                                       ; in the transmit ring.  The transmit buffers follow the transmit
                                ;                                       ; ring directly or start at address 20(0)
                                ;--+                                    ;
                                                                        ;
060430                                  CALL    REMAP   #OTRING         ; enable access to portion of memory
                                                                        ; that has transmit ring and buffers
                                                                        ;
060442  012701  100050                MOV     #XBUFV1,R1              ; R1 has virt addr. of first buffer
060446  012702  100000                MOV     #XRING,R2               ; R2 has base address of transmit ring
060452  012703  040050                MOV     #X11501,R3              ; R3 points to the first transmit buffer
060456  012704  000004                MOV     #NO.NTR,R4              ; R4 has count of transmit ring entries
                                                                        ;
060462  012722  002756        30$:    MOV     #RPKLEN,(R2)+           ; setup segment length
060466  010322                        MOV     R3,(R2)+                ; store address <15:01> of SEGB
060470  012722  000001                MOV     #X11716,(R2)+           ; store address <17:16> of SEGB
060474  005722                        TST     (R2)+                   ; leave room for buffer length
060476  010122                        MOV     R1,(R2)+                ; store virt. addr. of SEGB
060500  062701  002756                ADD     #RPKLEN,R1              ; point R1 to next transmit buffer
060504  062703  002756                ADD     #RPKLEN,R3              ; point R3 to next transmit buffer
060510  005304                        DEC     R4                      ; decrement loop control
060512  001363                        BNE     30$                     ; non-zero means more to do
                                                                        ;
                                ;--+                                    ;
                                ;                                       ; The node table needs to be cleared.
                                ;--+                                    ;
060514                                  CALL    REMAP   #ONTAB          ; allow access to node table
060526  012702  100000                MOV     #NODTBL,R2              ; let R2 point to the node table
060532  005022                40$:    CLR     (R2)+                   ; DO clear the node location WHILE
060534  020227  110000                CMP     R2,#NODEND              ;     there are more locations to clear
060540  001374                        BNE     40$                     ; ENDDO
                                                                        ;
                                ;--+                                    ;
                                ;                                       ; The summary table must be cleared also
                                ;--+                                    ;
060542                                  CALL    REMAP   #OSTAB          ; allow access to summary table
060554  012702  100000                MOV     #STATBL,R2              ; let R2 point to the summary table
```

```
 8633 060560  005022                    50$:    CLR     (R2)+                   ; clear a word of summary table
 8634 060562  020227  126000                    CMP     R2,@STAEND              ; Are there more locations to clear?
 8635 060566  001374                            BNE     50$                     ; YES, keep going
 8636
 8637 060570                                    CALL    RETMEM                  ; restore mapping of upper memory
 8638
 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,@#MMCSR0        ; 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,@#MMCSR0        ; 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 adata structure.
 8745                                   ;                          Do this by adding 200(0) to KPAR4 *)
 8746                                   ;                       KPAR4 <-- KPAR4 + 200(0)
```

```
                                ;
                                ;                       (* need to clear bit 13 of virtual address so it will
                                ;                          map through KPAR4 *)
                                ;                       CLEAR BIT 13 of first virtual address
                                ;
                                ;               ENDIF
                                ;
                                ;               (* ELSE no change on first pair *)
                                ;
                                ;               KPAR5 <-- second base address
                                ;
                                ;               TEST BIT 13 of second virtual address
                                ;
                                ;               IF SET THEN
                                ;                       (* want to access the second page of a data structure.
                                ;                          Do this by adding 200(0) to KPAR5 *)
                                ;                       KPAR5 <-- KPAR5 + 200(0)
                                ;
                                ;               ELSE
                                ;                       (* KPAR5 was correct, but need to set bit 13 of virtual
                                ;                          address to map through KPAR5 *)
                                ;                       SET BIT 13 of second virtual address
                                ;
                                ;               ENDIF
                                ;
                                ;               After the base and virtual addresses are derived, KPAR4 and
                                ;               KPAR5 are written and MMU is enabled.
                                ;
                                ; Inputs - Implicit - NOTE: because of speed considerations registers
                                ;                           one through four must be set up before routine
                                ;                           is called
                                ;               R1 - first base value
                                ;               R2 - first virtual address
                                ;               R3 - second base value
                                ;               R4 - second virtual address
                                ;
                                ; Outputs - none
                                ;
                                ; Calling Procedure: SET UP R1 - R4
                                ;                    JSR PC,PARVIR
                                ;
                                ; Side Effects -
                                ;               1.) KPAR4 and KPAR5 are remapped
                                ;               2.) the memory management unit is enabled
                                ;               3.) R1 - R4 may be modified
                                ;
                                ; Subordinate Routines - none
                                ;
                                ; Register Usage - as above
                                ;
                                ;--+
060660                          PARVIRT::
060660  012737  000000  177572          MOV     @#MMUDIS,@#MMCSR0       ; disable memory management
                                ;--+
                                ;       Test bit 13 of the source virtual address.  If it is set, clear
                                ;       it and point KPAR4 to next page in memory
```

```
 8804                                   ;--+
 8805 060666  032702  020000            BIT     @BIT13,R2               ; Test bit 13 of source virtual addr.
 8806 060672  001404                    BEQ     10$                     ; branch if clear
 8807 060674  042702  020000            BIC     @BIT13,R2               ; clear bit 13 to map through KPAR4
 8808 060700  062701  000200            ADD     @200,R1                 ; point KPAR4 to next page in memory
 8809
 8810                                   ;--+
 8811                                   ;       Test bit 13 of the destination virtual address.  If it was set then
 8812                                   ;       point KPAR5 to next page in memory.  If it was clear, then set it
 8813                                   ;       to map through KPAR5 as is.
 8814                                   ;--+
 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$:    BIS     @BIT13,R4               ; set bit 13 to map through KPAR5
 8822
 8823 060724  010137  172350            30$:    MOV     R1,@#KPAR4              ; remap KPAR4 ...
 8824 060730  010337  172352                    MOV     R3,@#KPAR5              ; ... and KPAR5
 8825
 8826 060734  012737  000001  177572            MOV     #MMUENA,@#MMCSR0        ; 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$POP   R1,R2,R3,R4             ; Set up registers for call to PARVIR
 8870
 8871 060754  004737  060660'                   JSR     PC,PARVIR               ; adjust base and virtual addresses
 8872
 8873 060760                                    P$POP   R3                      ; R3 gets number of bytes to compare
 8874 060762                                    CALL    CMPTWO  R2,R4,R3        ; 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 cal' to PARVIR
 8915
 8916 061022  004737  060660'                   JSR     PC,PARVIR               ; adjust base and virtual addresses
 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
 8942
 8944                           ;*************************************************************************
 8945                           ;       BEGIN THE REMAINING TESTS ON NEW PAGES.
 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
 8964                                   ;*****************************************************************************
 8965                                   ;       INSERT HARDWARE PARAMETER INTERPRETIVE CODE HERE.  THIS CODE
 8966                                   ;       IS USED BY THE SUPERVISOR TO INTERROGATE THE OPERATOR FOR
 8967                                   ;       DEVICE INFORMATION TO PUT IN THE P-TABLE.  THIS CODE IS USED
 8968                                   ;       IN CONJUNCTION WITH THE DEFAULT P-TABLE TEMPLATE.  THE MACROS
 8969                                   ;       USED IN THIS SECTION ARE "GPRMD", "GPRMA" AND "GPRML".
 8970                                   ;*****************************************************************************
 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
 8980                                   ;*****************************************************************************
 8981                                   ;     INSERT MESSAGES THAT ARE USED ONLY
 8982                                   ;     DURING THE HARDWARE PARAMETER CODING SECTION.
 8983                                   ;*****************************************************************************
 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
```

HARDWARE PARAMETER CODING SECTION

```
        061212      117     122     111
        061215      124     131     040
        061220      114     105     126
        061223      105     114     077
        061226      000
 8989                                           .EVEN
 8990
```

```
 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                                BGNSFT
 9004
 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                                ;*****************************************************************************************
 9013
 9014                                        .EVEN
 9015
 9016 061232                                ENDSFT
 9017
 9018
 9020                                ;*****************************************************************************************
 9021                                ;     INSERT MESSAGES THAT ARE USED ONLY
 9022                                ;     DURING THE SOFTWARE PARAMETER CODING SECTION.
 9023                                ;*****************************************************************************************
 9025
 9026 061232                        $PATCH::
 9027 061232                                .BLKW   10
 9028
 9030                                ;*****************************************************************************************
 9031                                ;       THIS IS A PATCH AREA THAT SHOULD BE INCLUDED IN ALL DIAGNOSTICS.
 9032                                ;       ADJUST THE SIZE TO FIT YOUR OWN PREFERENCES.
 9033                                ;*****************************************************************************************
 9035
 9036 061252                                LASTAD
      061256                        L$LAST::
```

```
9038
9039
9041                            ;**********************************************************************
9042                            ;       HARDCODED P-TABLES MAY BE PLACED HERE BY USING THE SETUP MACROS
9043                            ;       THIS SECTION IS OPTIONAL AND SHOULD BE REMOVED IF IT IS NOT BEING
9044                            ;       USED.  CHANGE THE POINTER MACRO ARGUMENT TO REFLECT THE REMOVAL.
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     CPYLMT  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
ACTLLI  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      CTARGT= 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      CBOADF  001166R      CMDTY9  017447R      CUNSVF= 000045
ACTDFT  051544R     ARGTY6  017527R      CBOBUF  001042R      CMPBUF  003130RG    CZEROS= 000020
ACTDIR  045750R     ARGTY7  017536R      CCCITT= 000023       CMPERH  025774R      C$AU   = 000052
ACTEXT  044134R     ASKCSR  061104R      CCITT  = 000005 G    CMPER1  026042R      C$AUTO= 000061
ACTFBB  042224RG    ASKPRI  061173R      CCLIST= 000061       CMPER2  026115R      C$BRK  = 000022
ACTFCT  051410R     ASKVEC  061137R      CCLMSG= 000035       CMPER3  026242R      C$BSEG= 000004
ACTHLP  040240R     ASSEMB= 000010       CCLNAD= 000004 G     CMPEXT  060044RG     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$CLCK= 000062
ACTLIS  053206RG    BINDEC  034122RG     CCNTR  = 000036       CNA     012776R      C$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       CNODAL= 000031       C$CVEC= 000036
ACTNOD  040312R     BIT01  = 000002 G    CEXADR= 000013       CNODE  = 000015       C$DCLN= 000044
ACTNUF  040174R     BIT02  = 000004 G    CEXIT  = 000020 G    CNTR00  017546R      C$DODU= 000051
ACTNUL  040202R     BIT03  = 000010 G    CEXPRO= 000056       CNTR01  017626R      C$DRPT= 000024
ACTONE  044260R     BIT04  = 000020 G    CFLAG   002024R      CNTR02  017675R      C$DU   = 000053
ACTOPR  044330R     BIT05  = 000040 G    CFUNCT= 000040       CNTR03  017730R      C$EDIT= 000003
ACTPAT  047352R     BIT06  = 000100 G    CHARAC  013445R      CNTR04  017775R      C$ERDF= 000055
ACTPRO  043222R     BIT07  = 000200 G    CLIACT  040012R      CNTR05  020052R      C$ERHR= 000056
ACTRNA  045302R     BIT08  = 000400 G    CLIALP= 000006       CNTR06  020121R      C$ERRO= 000060
ACTRNL  046414R     BIT09  = 001000 G    CLIBIF= 000003       CNTR07  020160R      C$ERSF= 000054
ACTRUN  045146R     BIT1   = 000002 G    CLIBR  = 000002       CNTR08  020230R      C$ERSO= 000057
ACTSAV  051612R     BIT10  = 002000 G    CLIBRX  011732R      CNTR09  020302R      C$ESCA= 000010
ACTSBB  042042RG    BIT11  = 004000 G    CLIDEC= 000011       CNTR10  020352R      C$ESEG= 000005
ACTSLI  057372R     BIT12  = 010000 G    CLIERM  011623R      CNTR11  020410R      C$ESUB= 000003
ACTSMS  047434R     BIT13  = 020000 G    CLIERR= 000000       CNTR12  020457R      C$ETST= 000001
ACTSND  051006R     BIT14  = 040000 G    CLIEXI= 000001       CNTR13  020524R      C$EXIT= 000032
ACTSOU  053112R     BIT15  = 100000 G    CLINBG  011705R      CNTR14  020571R      C$GETB= 000026
ACTSQK  040204R     BIT2   = 000004 G    CLINUF  011654R      CNTR15  020624R      C$GETW= 000027
ACTSR4  044242R     BIT3   = 000010 G    CLINUM= 000005       CNTR16  020666R      C$GMAN= 000043
ACTSUM  042674R     BIT4   = 000020 G    CLIOCT= 000010       CNTR17  020734R      C$GPHR= 000042
ACTSZE  044516R     BIT5   = 000040 G    CLISPA= 000004       CNTR18  021006R      C$GPLO= 000030
ACTTYP  044510R     BIT6   = 000100 G    CLISTR= 000012       CNTR19  021052R      C$GPRI= 000040
ACTUSF  052270R     BIT7   = 000200 G    CLITRE  003430R      CNTR20  021123R      C$INIT= 000011
ACTXAD  044144R     BIT8   = 000400 G    CLI$PM  011614R      CNTR21  021162R      C$INLP= 000020
ACTZRO  044270R     BIT9   = 001000 G    CLKBR   002030R      COMAND  030330RG     C$MANI= 000050
ACTOAL  044310R     BLDBUF  033120RG     CLKCSR  002026R      COMPAR  017302R      C$MEM  = 000031
ACT1AL  044300R     BLDDON  012244R      CLKEN   002036R      CONES  = 000017       C$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$PNTF= 000017
ADRDEL  014621R     BLDREQ  032670RG     CLKVEC  002032R      CPROER  012553R      C$PNTS= 000016
ADREND= 101414 G    BNCBUF  002062R      CLRCNT= 000013 G     CPYCNT  003122RG     C$PNTX= 000015
```

Symbol table

```
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$REFG= 000047          EF.CON= 000036 G        ERRLK   005730RG        F$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$RPT  = 000025         EF.STA= 000040 G        ERROR   027316RG        F$SW   = 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$TPRI= 000013          EMSG02  021254R         ERR2    026654RG        GETFNT= 000002 G        HELP29 011105R
C.COLL= 000074 G        EMSG03  021304R         ERR3    026742RG        GETIDA  056020R         HELP3  006126R
C.MREC= 000010 G        EMSG04  021346R         EVL    = 000004 G       GETPCB= 000001 G        HELP30 011207R
C.MXMT= 000040 G        EMSG05  021400R         EXEBLD  040644R         GETRNX  033056RG        HELP31 011326R
C.PREC= 000004 G        EMSG06  021443R         EXEBNC  042354R         GETTYP  056134R         HELP32 011376R
C.PXMD= 000054 G        EMSG07  021503R         EXEHLP  040250RG        GETXNX  033070RG        HELP33 011505R
C.PXMT= 000034 G        EMSG08  021556R         EXELIS  056272RG        G$CNTO= 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$EXCP= 000400          HELP7  006463R
C.RERB= 000014 G        EMSG14  021706R         FAADR1= 000022 G        G$HILI= 000002          HELP8  006574R
C.RERR= 000016 G        EMSG15  021761R         FAADR2= 000032 G        G$LOLI= 000001          HELP9  006664R
C.RLEX= 000032 G        EMSG16  022014R         FAADR3= 000042 G        G$NO   = 000000         HEXBIN 031632RG
C.RLIN= 000030 G        EMSG18  022067R         FAADR4= 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         FIXDSL  053714R         HDMSG1  015710R         IDTCSA= 000013 G
DEF    = 002000 G       EMSG34  022654R         FORLOG  052160R         HDMSG2  015761R         IDTLUA= 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
DMXFLG  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$INIT= 000041        LOGNM   045126RG      L$SPTP  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      NCMN  = 020000 G      NOD14   003540R
I$PTAB= 000041        LPACNM  001236R       L$TIML  000014RG      NCMPAR= 000050        NOD140  004536R
I$PWR = 000041        LST     031750R       L$UNIT  000012RG      NETADR  012726R       NOD141  004542R
I$RPT = 000041        LTMSG   013736R       L10000  000212R       NEW     037250R       NOD142  004546R
I$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
KPAR5 = 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$DISP  000200RG      L10020  061232R       NODTYP  012720R       NOD16   003560R
LCLKEN= 000100 G      L$DLY   000116RG      MEMMAP  060336RG      NOD0    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$DU    037510RG      MMCSR0= 177572 G      NOD100  004270R       NOD163  004662R
LDADR2= 000032 G      L$DUT   000072RG      MMUDIS= 000000 G      NOD101  004274R       NOD164  004666R
LDATA = 000022 G      L$DVTY  000122RG      MMUENA= 000001 G      NOD102  004300R       NOD165  004672R
LDFCT1= 000020 G      L$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$INIT  035662RG      MSG0C   001432R      NOD117  004400R       NOD200  004764R
LISFUL  001252R       L$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$NAME  000000RG      MSG05   001622RG      NOD124  004424R       NOD206  005030R
LISNXT  001232R       L$PRIO  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
LOCDST  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$   004644R     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
PCBB0   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    SVCGBL= 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$GMAN= 000000
PCMSG   025734RG    P$MERR  001304R    RUNPAT  047362R    SVCTAG= 177777       T$HILI= 000007
PCSR0   002106RG    P$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$TEXT  001305R    SELMSG  053464R    S.REC   002766RG     T$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$TRS   034352R    SFPTBL  000214RG    TABEMT  014113R     T$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$TEST= 000001
POSDS   025221R     RBUFV1= 100120 G   SIMSG3  024261R    TEMP1   003112RG     T$TSTM= 177777
POSDS0  025250R     RCBI  = 002000 G   SIMSG4  024334R    TEMP2   003114RG     T$TSTS= 000001
POSDS1  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$$HAR= 010017
POSRVN  025470R     RDMODE= 000014 G   SIRCPT= 000022 G    TIMMIN  002040R     T$$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    TRVBRC  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     RETRYS  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        XRGLST  002102RG        ZEROS = 000002 G        $RDMO   002556RG
UNPISR  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        $WDMO   002566RG
UNKNWN  013032R         XFLAG   003010RG        X$FALS= 000040          $PNOP   002230RG        $WDPH   002252RG
UNSMSG  015133R         XMIT    030414RG        X$OFFS= 000400          $RDCN   002432RG        $WDRN   002406RG
USCI  = 000400 G        XPKLEN= 001756 G        X$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
```