RQDX1 RQDX1 FORMATTER
RD51 RD52 CZRQBC0

AH-T567C-MC
FICHE 01 OF 02

JUL 1984
digital
Made In USA

RQDX1 RQDX1 FORMATTER
RD51 RD52 CZRQBC0

AH-T567C-MC
FICHE 02 OF 02

JUL 1984
digital
Made In USA

```
;    0001  0     MODULE ZRQB1 (%TITLE 'RDRx DISK FORMATTER'
;    0002  0                   IDENT = 'REV C PATCH 0',
;    0003  0                   ADDRESSING_MODE (ABSOLUTE) ,
;    0004  0                   ENVIRONMENT (NOEIS)
;    0005  0                   ) ;
;    0006  0
;    0007  1     BEGIN
;    0008  1
; C  0009  1     %(
; C  0010  1
; C  0011  1                              IDENTIFICATION
; C  0012  1                              --------------
; C  0013  1
; C  0014  1             PRODUCT CODE:            AC-I566C-MC
; C  0015  1
; C  0016  1             PRODUCT NAME:            CZRQBC0 RQDx1 DISK FORMATTER
; C  0017  1
; C  0018  1             PRODUCT DATE:            9 APRIL 1984
; C  0019  1
; C  0020  1             MAINTAINER:              DIAGNOSTIC ENGINEERING
; C  0021  1
; C  0022  1             AUTHOR:                  Doug Neale
; C  0023  1
; C  0024  1
; C  0025  1             THE INFORMATION IN THIS DOCUMENT IS  SUBJECT  TO  CHANGE WITHOUT
; C  0026  1             NOTICE  AND  SHOULD  NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
; C  0027  1             EQUIPMENT CORPORATION.  DIGITAL EQUIPMENT CORPORATION ASSUMES NO
; C  0028  1             RESPONSIBILITY FOR ANY ERRORS THAT MAY APPEAR IN THIS DOCUMENT.
; C  0029  1             NO RESPONSIBILITY IS ASSUMED  FOR  THE  USE  OR  RELIABILITY  OF
; C  0030  1             SOFTWARE ON EQUIPMENT THAT IS NOT  SUPPLIED  BY  DIGITAL  OR ITS
; C  0031  1             AFFILIATED COMPANIES.
; C  0032  1
; C  0033  1             COPYRIGHT (C) 1983/84 BY DIGITAL EQUIPMENT CORPORATION
; C  0034  1             THE FOLLOWING ARE TRADEMARKS OF DIGITAL EQUIPMENT CORPORATION:
; C  0035  1
; C  0036  1             DIGITAL          PDP              UNIBUS           MASSBUS
; C  0037  1             DEC              DECUS            DECTAPE
```

```
;C 0038  1                    REVISION HISTORY
;C 0039  1
;C 0040  1
;C 0041  1               February 1983   Jay Nevin
;C 0042  1                  -Adapted existing AZTEC formatter for use with RQDX RD51
;C 0043  1
;C 0044  1               March 1983      Russell Young
;C 0045  1                  -Corrected minor incompatabilities in expectations between controller and host
;C 0046  1
;C 0047  1               February 1984   Russell Young
;C 0048  1                  -Updated the main routine to work on either version 8 or 9 of the RQDX microcode.
;C 0049  1                   In version 8 the ASCII strings are stored in the host code because of space
;C 0050  1                   limitations in the controller. Version 9, for increased generality, has the
;C 0051  1                   strings held in the controller and transmitted to the host. This still contains
;C 0052  1                   the original code section to maintain compatability with version 8.
;C 0053  1
;C 0054  1               March 1984      Russell Young
;C 0055  1                  -Removed checks on incoming error and informational messages to allow controller
;C 0056  1                   modification without reissuing this every time. Also, the routine SETCPU was
;C 0057  1                   added to allow this to run on an ORION system.
```

```
;  C 0058  1
;  C 0059  1                        TABLE OF CONTENTS
;  C 0060  1
;  C 0061  1
;  C 0062  1
;  C 0063  1                1.0     GENERAL INFORMATION
;  C 0064  1
;  C 0065  1                1.1     PROGRAM ABSTRACT
;  C 0066  1                1.1.1   HOST RESIDENT PROGRAM
;  C 0067  1                1.1.2   CONTROLLER RESIDENT PROGRAM
;  C 0068  1
;  C 0069  1                1.2     SYSTEM REQUIREMENTS
;  C 0070  1                1.2.1   HARDWARE REQUIREMENTS
;  C 0071  1                1.2.2   SOFTWARE REQUIREMENTS
;  C 0072  1
;  C 0073  1                1.3     RELATED DOCUMENTS AND STANDARDS
;  C 0074  1
;  C 0075  1                2.0     OPERATING INSTRUCTIONS
;  C 0076  1                2.1     HARDWARE QUESTIONS
;  C 0077  1                2.2     SOFTWARE QUESTIONS
;  C 0078  1                2.3     FORMATTER QUESTIONS
;  C 0079  1
;  C 0080  1                3.0     RUNNING THE FORMATTER
;  C 0081  1                3.1     ERRORS
;  C 0082  1                3.2     SUCCESS
```

```
; C 0083  1
; C 0084  1
; C 0085  1
; C 0086  1               1.0   GENERAL INFORMATION
; C 0087  1                     -------------------
; C 0088  1
; C 0089  1
; C 0090  1               1.1   PROGRAM ABSTRACT
; C 0091  1                     ---------------
; C 0092  1
; C 0093  1               1.1.1 HOST RESIDENT PROGRAM
; C 0094  1
; C 0095  1                     This program is the front end which invokes the formatter
; C 0096  1                     for the RD51/52 disk connected to the RQDX1 controller. It
; C 0097  1                     interfaces with the actual formatter which is in the controller.
; C 0098  1                     This involves initialization of the port, invoking the actual
; C 0099  1                     formatter via the DUP protocol, getting needed data from the
; C 0100  1                     user and sending it to the controller, and finally informing
; C 0101  1                     the user of the final outcome. The last two steps depend on
; C 0102  1                     the version of the controller present. The host resident program
; C 0103  1                     is designed to run both version 8 and version 9 of the RQDX
; C 0104  1                     controller.
; C 0105  1
; C 0106  1
; C 0107  1
; C 0108  1               1.1.2 CONTROLLER RESIDENT PROGRAM
; C 0109  1
; C 0110  1                     When invoked by the host resident portion, this will prompt
; C 0111  1                     for any information it needs, and then begin running. A run
; C 0112  1                     consists of marking the disk as unformatted, formatting it,
; C 0113  1                     running three passes of a surface analysis, saving the FCT
; C 0114  1                     and RCT, and marking the disk as formatted.
; C 0115  1
; C 0116  1
; C 0117  1
; C 0118  1               1.2   SYSTEM REQUIREMENTS
; C 0119  1                     -------------------
; C 0120  1
; C 0121  1               1.2.1 HARDWARE REQUIREMENTS
; C 0122  1                     ---------------------
; C 0123  1
; C 0124  1                     LSI - 11/23 processor with 28K or more of memory, console
; C 0125  1                     device (EX. VT100) and RQDX1 CONTROLLER board and attached
; C 0126  1                     RD51/52 WINCHESTER drive(s).
; C 0127  1
; C 0128  1                     As of rev C of this program, it will also run on an Orion
; C 0129  1                     J-11 processor.
; C 0130  1
; C 0131  1               1.2.2 SOFTWARE REQUIREMENTS
; C 0132  1                     ---------------------
; C 0133  1
; C 0134  1                     THIS DIAGNOSTIC IS DESIGNED TO RUN WITH THE DIAGNOSTIC SUPERVISOR
; C 0135  1                     AS DESCRIBED IN PARAGRAPH 2.0.
```

```
; C 0136  1
; C 0137  1
; C 0138  1
; C 0139  1
; C 0140  1
; C 0141  1
; C 0142  1
```

1.3 RELATED DOCUMENTS AND STANDARDS

XXDP+ SUPERVISOR/USERS MANUAL CHQUS
DSDFV12 DEC DISK FORMATTING STANDARDS

```
; C 0143  1
; C 0144  1
; C 0145  1
; C 0146  1
; C 0147  1
; C 0148  1
; C 0149  1
; C 0150  1
; C 0151  1
; C 0152  1
; C 0153  1
; C 0154  1
; C 0155  1
; C 0156  1
; C 0157  1
; C 0158  1
; C 0159  1
; C 0160  1
; C 0161  1
; C 0162  1
; C 0163  1
; C 0164  1
; C 0165  1
; C 0166  1
; C 0167  1
; C 0168  1
; C 0169  1
; C 0170  1
; C 0171  1
; C 0172  1
; C 0173  1
; C 0174  1
; C 0175  1
; C 0176  1
; C 0177  1
; C 0178  1
; C 0179  1
; C 0180  1
; C 0181  1
; C 0182  1
; C 0183  1
; C 0184  1
; C 0185  1
; C 0186  1
; C 0187  1
; C 0188  1
; C 0189  1
; C 0190  1
; C 0191  1
; C 0192  1
; C 0193  1
; C 0194  1
; C 0195  1
```

2.0 OPERATING INSTRUCTIONS

This is a rev C supervisor diagnostic: for operating instructions, please see chapter 5 of xxdp+ operator's manual. they are no longer included in the diagnostic listing because it is desired that a change in those instructions not require a reassembly of all supervisor diagnostics.

2.1 HARDWARE QUESTIONS

The following series of questions comprise the parameters necessary to initialize the controller.

Hardware Configuration Questions

The program will ask the following questions in response to a START command (non-script). No default will be accepted for the CHANGE HARDWARE and SOFTWARE questions.

1. CHANGE HW (L) ?

Answer NO to use the pre-built answers for all hardware questions. This program will be released pre-built to format unit 0 with default answers shown below. The pre-built answers may be changed at any time with the setup utility. Answer YES to be asked all the hardware questions.

2. IP ADDRESS (O) 172150 ?

Enter the address of the IP register of one RDQX1 as addressed by the processor with memory management turned off. The program expects an even 16-bit address in the range of 160000 to 177774. 172150 is the default.

3. VECTOR ADDRESS (O) 154 ?

Answer with the interrupt vector of same RDQX1 in the above question. A vector address in the range of 4 to 774 may be specified. 154 is the default.

```
; C 0196  1
; C 0197  1
; C 0198  1
; C 0199  1
; C 0200  1
```

4. BR LEVEL (D) 4 ?

Answer with the bus request interrupt level used by the above RQDX1 Levels 4 through 7 are acceptable. 4 is the default.

```
; C 0201  1
; C 0202  1
; C 0203  1
; C 0204  1
; C 0205  1
; C 0206  1
; C 0207  1
; C 0208  1
; C 0209  1
; C 0210  1
; C 0211  1
; C 0212  1
; C 0213  1
; C 0214  1
; C 0215  1
; C 0216  1
; C 0217  1
; C 0218  1
; C 0219  1
; C 0220  1
; C 0221  1
; C 0222  1
; C 0223  1
; C 0224  1
; C 0225  1
; C 0226  1
; C 0227  1
; C 0228  1
; C 0229  1
; C 0230  1
; C 0231  1
; C 0232  1
; C 0233  1
; C 0234  1
; C 0235  1
; C 0236  1
; C 0237  1
; C 0238  1
; C 0239  1
; C 0240  1
; C 0241  1
; C 0242  1
; C 0243  1
; C 0244  1
; C 0245  1
; C 0246  1
; C 0247  1
; C 0248  1
; C 0249  1
; C 0250  1
; C 0251  1
; C 0252  1
; C 0253  1
```

2.2 SOFTWARE QUESTIONS

Software Parameter Questions

The program will ask the following questions in response to a START or RESTART command (non-script). No default will be accepted.

1. CHANGE SW (L) ?

   Answer either Y or N to this question. A Yes answer will allow the formatter to be set up for the APT environment by asking only questions 2. and 3. below. A No answer will cause the formatter to proceed to ask questions as explained in section 2.3 below.

2. SOFTWARE QUESTIONS ONLY APPLY UNDER APT. ENTER UNIT NUMBER (0) ?

   This will accept any answer in the range of 0 to 3. It will default to 0 if a <CR> is struck.

3. ENTER MODE [1 = REFORMAT, 2 = RESTORE, 3 = RECONSTRUCT] (0) ?

   Answering this question with one of the above numbers will cause the formatter to try one of the three formatting modes as explained in 2a, 2b or 2c below.

2.3 FORMATTER QUESTIONS

The questions asked depend on the version of the controller microcode in the RQDX. When the controller is first initialized, the host program determines which version is running. If it is version 8, the host program, driven by prompts from the controller, displays questions stored as ASCII text on the host. If version 9 is running, the prompts are transmitted from the controller. The actual questions asked and the order in which they are asked differs in the two versions.

2.3.1 VERSION 8 QUESTIONS

After these DRS set up questions, the version 8 formatter will ask the following manual intervention questions needed to proceed. These questions will not be asked under APT. The default answers contained in the formatter will be used. These answers interact directly with the RQDX1 Controller.

```
; C 0254  1               1.  ENTER UNIT NUMBER TO FORMAT  (0)
; C 0255  1
; C 0256  1                       The answer should be in the range of 0 to 3. The default answer
; C 0257  1                       is 0.
```

```
; C 0258  1
; C 0259  1
; C 0260  1                2.    The next three questions select the type of format which will
; C 0261  1                      be done. The three modes are explained in Unibus Disk Adapter
; C 0262  1                      Functional Specication REV: 2.8.  Since they are mutually
; C 0263  1                      exclusive, answering Y to any one of them will cause the formatter
; C 0264  1                      to skip the remaining ones and go on to the next question.
; C 0265  1                      Answering N to all three will cause it to default to REFORMAT
; C 0266  1                      mode, the same as answering Y to question 2a. In this case,
; C 0267  1                      the following message will be printed.
; C 0268  1
; C 0269  1                              EXISTING BAD BLOCK INFORMATION USED
; C 0270  1
; C 0271  1
; C 0272  1                2a. USE EXISTING BAD BLOCK INFORMATION  (N)
; C 0273  1
; C 0274  1                      Answering Y to this will cause the formatter to try a REFORMAT
; C 0275  1                      mode format. This means it will try to read its own FCT to get
; C 0276  1                      its serial number, and try reading the manufacturer's bad spot
; C 0277  1                      record on the inner cylinder to initialize the RCT. If it fails
; C 0278  1                      in either attempt, it will give up and return an error.
; C 0279  1
; C 0280  1
; C 0281  1                2b. USE DOWN LINE LOAD  (N)
; C 0282  1
; C 0283  1                      This mode is known as RESTORE, it in not currently
; C 0284  1                      supported, but is included for possible future improvement.
; C 0285  1                      Answering Y to it will have the same result as answering Y
; C 0286  1                      to question 2c.
; C 0287  1
; C 0288  1                2c. CONTINUE IF BAD BLOCK INFORMATION IS INACCURATE (N)
; C 0289  1
; C 0290  1                      Answering Y to this will cause a RECONSTRUCT mode format
; C 0291  1                      to be done. Nothing will be assumed about the disk, and
; C 0292  1                      the manufacturer's bad spot data, even if present, will be
; C 0293  1                      ignored.
; C 0294  1
; C 0295  1
; C 0296  1                3.  ENTER 8 CHARACTER SERIAL NUMBER
; C 0297  1
; C 0298  1                      If REFORMAT mode is selected, this question will not be asked.
; C 0299  1                      Otherwise, it needs 8 characters to be entered. The number is not
; C 0300  1                      important, and must only be unique on the controller. Thus, on
; C 0301  1                      a one RD51/52 system, any eight characters      will suffice, and on a
; C 0302  1                      two RD51/52 system each one must be different - this is not difficult
; C 0303  1                      to achieve, since each character may be any printable ASCII symbol.
; C 0304  1
; C 0305  1                4.  ENTER DATE IN MM-DD-YY FORMAT
; C 0306  1
; C 0307  1                      Type in the date of formatting. It needs exactly eight
; C 0308  1                      characters, so January 1, 1984 must be entered as 01-01-84
; C 0309  1                      This question will be asked in all modes of formatting.
```

```
; C 0310  1
; C 0311  1          2.3.2   VERSION 9 QUESTIONS
; C 0312  1                  ------------------
; C 0313  1
; C 0314  1                  1.  Enter date <MM-DD-YYYY>
; C 0315  1                      Date should be entered in the form requested. For example,
; C 0316  1                              1-29-1958  , or
; C 0317  1                              4-25-1980
; C 0318  1
; C 0319  1                  2.  Enter unit number to format <0>:
; C 0320  1                      Unit number is entered in decimal, 0 <= x < 16. Default is 0
; C 0321  1
; C 0322  1                  3.  Use existing bad block information <N>:
; C 0323  1                      If yes, Execute a REFORMAT mode formatting. That is, try to
; C 0324  1                      use the RCT which already exists on the disk.
; C 0325  1
; C 0326  1                  4.  Use down-line load <N>:
; C 0327  1                      A Y here will cause question 5 to be skipped, and
; C 0328  1                      the controller to prompt for the head, cylindar, and
; C 0329  1                      byte offset of the bad blocks, as reported by the
; C 0330  1                      manufacturer.
; C 0331  1
; C 0332  1                  5.  Continue if bad block information is inaccessable <N>:
; C 0333  1                      If the formatter tries to read the Factory Control Table and
; C 0334  1                      fails, it will abort the format if this is not answered yes.
; C 0335  1
; C 0336  1                  6.  Enter non-zero serial number:
; C 0337  1                      Enter one to ten digits. The controller requires an even number,
; C 0338  1                      so if an odd number are typed a "." is appended to the end.
; C 0339  1
; C 0340  1
; C 0341  1          3.0     RUNNING
; C 0342  1                  -------
; C 0343  1
; C 0344  1                     After asking the date, the actual formatting will begin.
; C 0345  1                  If all goes well, in just under 11 minutes it will return
; C 0346  1                  a successful completion message. Otherwise, it will print
; C 0347  1                  an error message, probably much sooner.
; C 0348  1
; C 0349  1
; C 0350  1          3.1.1   Version 8 Errors
; C 0351  1
; C 0352  1                     The following are the error messages generated by the formatter.
; C 0353  1                  If any other message appears it has been printed by DRS or XXDP+,
; C 0354  1                  so refer to the pertinent documentation for explanation.  Errors
; C 0355  1                  1, 2, and 3 will occur almost immediately, 4 can appear up to
; C 0356  1                  about a minute after starting, 5 from about 1 minute to 10 minutes,
; C 0357  1                  and 6 and 7 after 10 minutes.
; C 0358  1
; C 0359  1          1.      UNIT IS NOT WINCHESTER OR CAN NOT BE SELECTED
; C 0360  1                     The unit selected is either unavailable or is not
; C 0361  1                     an RD51/52. Check to assure it is not write protected.
; C 0362  1
```

```
; C 0363  1              2.      INITIAL FAILURE ACCESSING FCT
; C 0364  1                         The Format Control Table cannot be read. If you are trying
; C 0365  1                         REFORMAT mode, try RECONSTRUCT. If that fails also, the disk
; C 0366  1                         may be bad.
; C 0367  1
; C 0368  1              3.      FACTORY BAD BLOCK INFORMATION IS INACCESSABLE
; C 0369  1                         This will only occur if a REFORMAT is attempted and the factory
; C 0370  1                         bad spot data is not accessible. Run in RECONSTRUCT mode.
; C 0371  1
; C 0372  1              4.      SEEK FAILURE DURING ACTUAL FORMATTING
; C 0373  1                         There has been a hardware error during the actual formatting.
; C 0374  1                         If this error persists, check for hardware problems.
; C 0375  1
; C 0376  1              5.      REVECTOR LIMIT EXCEEDED
; C 0377  1                         The disk can only handle 144 bad blocks, and more than
; C 0378  1                         that have been found. If this persists the disk is bad.
; C 0379  1
; C 0380  1              6.      RCT WRITE FAILURE
; C 0381  1                         The formatting and surface analysis were completed successfully,
; C 0382  1                         but a write to the disk afterwards failed.
; C 0383  1
; C 0384  1              7.      FAILURE CLOSING FCTS
; C 0385  1                         Everything has been completed, but the disk is still
; C 0386  1                         marked as being unformatted.
; C 0387  1
; C 0388  1              3.1.2   Version 9 error messages
; C 0389  1                      1.      GET STATUS failure
; C 0390  1                      2.      Q-PORT send error
; C 0391  1                      3.      Unsuccessful command
; C 0392  1                      4.      Q-PORT receive error
; C 0393  1                      5.      Q-Bus I/O error
; C 0394  1                      6.      Formatter initialization error
; C 0395  1                      7.      Nonexistent unit number
; C 0396  1                      8.      DBN/X8N format error (drive FORMAT command failed)
; C 0397  1                      9.      FCT does not have enough good copies of each block
; C 0398  1                      10.     SEEK error
; C 0399  1                      11.     RCT does not have enough good copies of each block
; C 0400  1                      12.     LBN format error (drive FORMAT command failed)"
; C 0401  1                      13.     FCT write error
; C 0402  1                      14.     RCT read error
; C 0403  1                      15.     RCT write error
; C 0404  1                      16.     RCT full
; C 0405  1                      17.     FCT read error
; C 0406  1                      18.     FCT nonexistant
; C 0407  1                      19.     FCT Down line-load error
; C 0408  1                      20.     Drive init timeout
; C 0409  1                      21.     Illegal response to start-up question
; C 0410  1                      22.     WARNING - possible head addressing problem - run diagnostics
; C 0411  1                      23.     INPUT Error
; C 0412  1                      24.     Media degraded
; C 0413  1
; C 0414  1
; C 0415  1          3.2     SUCCESS
```

```
; C 0416  1                   -------
; C 0417  1
; C 0418  1                      If all goes well, in about 11 minutes the format will be
; C 0419  1                   complete.
; C 0420  1           3.2.1   VERSION 8 SUCCESS
; C 0421  1                   -----------------
; C 0422  1                   Successful completion is signaled by a message
; C 0423  1
; C 0424  1                           FORMAT COMPLETED, xxx REVECTORED LBNS
; C 0425  1
; C 0426  1                   where xxx is a decimal number. This should be a small
; C 0427  1                   number - performance will deteriorate if it is too large.
; C 0428  1
; C 0429  1           3.2.2   VERSION 9 SUCCESS
; C 0430  1                   -----------------
; C 0431  1
; C 0432  1                      The rev 9 controller will provide additional information
; C 0433  1                   and statistics at the end of each run.
; C 0434  1
; C 0435  1           1. Format complete
; C 0436  1                      Signals the end of the formatting process
; C 0437  1           2. FCT used successfully
; C 0438  1                      or
; C 0439  1              FCT was not used
; C 0440  1                      Reports if the Factory Control Table was accessable
; C 0441  1           3. xxx  Revectored LBN's
; C 0442  1                      The total number of bad blocks found
; C 0443  1           4. xxx  Primary revectored LBN's
; C 0444  1                      The number of primary revectored LBNs. Currently the controller
; C 0445  1                      does not support primary revectoring.
; C 0446  1           5. xxx  Secondary/tertiary revectored LBN's
; C 0447  1                      This number should be equal to the total LBNs reported in
; C 0448  1                      message 3.
; C 0449  1           6. xxx Bad blocks in the RCT area due to data errors
; C 0450  1                       bad blocks found in the Replacement and Caching Table area
; C 0451  1           7. xxx Bad blocks in the DBN area due to data errors
; C 0452  1                       Bad blocks found in the Diagnostic Block Number area
; C 0453  1           8. xxx Bad blocks in the XBN area due to data errors
; C 0454  1                       Bad blocks in the eXtendedBlock Number area
; C 0455  1           9. xxx Bad RBN's
; C 0456  1                       Bad blocks found in the Replacement Block Number area
; C 0457  1           10. xxx Blocks retried on the check pass
; C 0458  1                       The number of revectorings
;   0459  1       )%
```

```
;    0460  1     %sbttl 'PROGRAM HEADER'
;    0461  1
;    0462  1     library 'ZRQBB0.L16';                   !Define RDRX Formatter Library
;    0463  1
;    0464  1     require 'BLSMAC.REQ';                   !Define Bliss Macro Require file
;    2000  1
;    2001  1     !+
;    2002  1     ! The  psect  named  "code  or  $code$" is redefinded here
;    2003  1     ! to  be  called "aa$code".  This is done to force the TKB
;    2004  1     ! linker to place the programs header information starting
;    2005  1     ! at absolute address 2000.
;    2006  1     !-
;    2007  1     !psect
;    2008  1     !    code = aa$code;
;    2009  1
;    2010  1     literal
;    2011  1         DS$NBR_OF_TESTS = 1;                !Indicates number of test in Diag
;    2012  1
;    2013  1     !+
;    2014  1     ! The structure of a diagnostic program may contain any or  all  of  the
;    2015  1     ! ten  optional  sections.  But five of the optional sections require a
;    2016  1     ! pointer that is derived by and for the supervisor, and is  located  in
;    2017  1     ! the  header  block.  Therefore,  in  relation to the effective use of
;    2018  1     ! these five pointers, the optional sections call must be coded  to  re-
;    2019  1     ! flect usage (i.e., any,all,or none).
;    2020  1     !
;    2021  1     ! The following coding possibilities exist:
;    2022  1     !
;    2023  1     !  POINTER (BGNRPT,BGNSW,BGNSFT,BGNAU,BGNDU,ERRTBL,BGNSETUP)
;    2024  1     !
;    2025  1     !  (or any subset of the args)
;    2026  1     !
;    2027  1     !  POINTER (ALL)                    ; All provides pointers for all five
;    2028  1     !                                   ; sections
;    2029  1     !  POINTER (NONE)                   ; None indicates to supervisor that no
;    2030  1     !                                   ; pointers are required.
;    2031  1     !                                   ; this is the default
;    2032  1     !
;    2033  1     !       No pointers are optional using bliss. Make sure the following
;    2034  1     !       sections of  code are in place (in the correct skels),even if
;    2035  1     !       the sections are blank.
;    2036  1     !
;    2037  1     !       ARGUMENT        FUNCTION
;    2038  1     !       --------        ------
;    2039  1     !       RPT             REPORT CODE
;    2040  1     !       SW              SOFTWARE TABLE
;    2041  1     !       SFT             SOFTWARE TABLE QUESTIONS
;    2042  1     !       AU              ADD CODE
;    2043  1     !       DU              DROP CODE
;    2044  1     !       TBL             ERROR TABLE
;    2045  1     !       SETUP           ASSEMBLED P-TABLES
;    2046  1     !-
;    2047  1     POINTER (ALL);
```

```
;    2048  1     !+
;    2049  1     ! The program  header section contains  general  information  which  des-
;    2050  1     ! cribes  the  major characteristics of the diagnostic program.  This in-
;    2051  1     ! cludes, the  program name, and revision and  patch-order  levels.   The
;    2052  1     ! header  also  provides  space   for an event flag register, and for the
;    2053  1     ! storage of  pointers, through which the supervisor may find  access  to
;    2054  1     ! other  key sections of the program(e.g., dispatch table, initialize and
;    2055  1     ! clean-up code, etc.).  An argument on the header gives the device type
;    2056  1     ! if it is an XXDP+ bootable device.  This enables the supervisor to pro-
;    2057  1     ! vide load medium protection when necessary.
;    2058  1     !-
;    2059  1     HEADER (%ascii'ZRQB', %ascii'C', %ascii'0', 1200, 0, PRI00);
```

```
;    2060  1     %sbttl 'DISPATCH TABLE'
;    2061  1     !+
;    2062  1     ! The dispatch table section contains address pointers  to  the  various
;    2063  1     ! tests  contained within the diagnostic program.  This section requires
;    2064  1     ! the coding of only the dispatch macro.
;    2065  1     !-
;    2066  1     DISPATCH (DS$NBR_OF_TESTS);
;    2067  1     ERRTBL;                                 !Define Supervisor Error table storage
```

```
;    2068  1     %sbttl 'DEFAULT HARDWARE P-TABLE'
;    2069  1     !+
;    2070  1     ! The  default hardware P-Table contains default values of
;    2071  1     ! the test-device parameters.  The structure of this table
;    2072  1     ! is  identical to  the structure of the hardware P-Tables,
;    2073  1     ! and  is used  as a  "template" for building the P-Tables.
;    2074  1     !-
;    2075  1     BGNHW (DFPTBL);
;    2076  1
;    2077  1     global
;    2078  1         HW_IP_ADRS : word initial (%o'172150'),      !Define RDRX Controler IP reg
;    2079  1         HW_VECTOR : word initial (%o'154'),          !Define RDRX interrupt vector adddrs
;    2080  1         HW_BR_LEVEL : word initial (4),              !Define RDRX bus request level
;    2081  1         HW_UNIT_NO : word initial (0);               !Define RDRX unit no. to format
;    2082  1
;    2083  1     ENDHW;
```

```
;    2084  1     %sbttl 'SOFTWARE P-TABLE'
;    2085  1     !+
;    2086  1     ! The  software table  contains various  data used  by  the
;    2087  1     ! program as  operational parameters.  These parameters are
;    2088  1     ! set up at assembly time and may be varied by the operator
;    2089  1     ! at run time.
;    2090  1     !-
;    2091  1     BGNSW (SFPTBL);
;    2092  1     !+
;    2093  1     ! All software parameter coding is done within the RDRX
;    2094  1     ! DM code.  This is  per DUP functional spec compliance.
;    2095  1     !-
;    2096  1     global
;    2097  1         SW_UNIT_NO :        word initial (0),           !default unit to 0
;    2098  1         SW_MODE :           word initial (3);           !default mode to reconstruct
;    2099  1     ENDSW;
```

```
;    2100  1     %sbttl 'PROTECTION TABLE'
;    2101  1     !+
;    2102  1     ! This  table is used by the runtime
;    2103  1     ! services to protect the load media.
;    2104  1     !
;    2105  1     ! 1st arg =     Offset into P_Table for csr address
;    2106  1     ! 2nd arg =     Offset into P_Table for massbus address
;    2107  1     ! 3rd arg =     Offset into P_Table for drive number
;    2108  1     !-
;    2109  1     BGNPROT (-1, -1, -1);
;    2110  1     ENDPROT;
```

```
;    2111  1     %sbttl 'MODULE DECLARATIONS'
;    2112  1     !+
;    2113  1     ! Within  BLSMAC.REQ  the  psect  names, plit global and own,
;    2114  1     ! are  redefined  to  be aa$code.  This is done to force the
;    2115  1     ! tkb  linker  to  link  the  header information starting at
;    2116  1     ! at absolute  address  2000.  Redefine  these  psect  names
;    2117  1     ! back  to  their  original names for house keeping purposes.
;    2118  1     !
;    2119  1     ! Also  change  the attributes for the psect "global" so that
;    2120  1     ! global data will not be linked starting at absolute address
;    2121  1     ! 2000.
;    2122  1     !-
;    2123  1     psect
;    2124  1         plit = $plit$( global),
;    2125  1         global = $glob$(nowrite, noexecute, global, concatenate),
;    2126  1         own = $own$;
;    2127  1
;    2128  1     !+
;    2129  1     ! Structure declarations used within this
;    2130  1     ! module.
;    2131  1     !-
;    2132  1
;    2133  1     structure
;    2134  1
;    2135  1         !+
;    2136  1         ! RDRX  register  accessing structure.  This
;    2137  1         ! structure  allows  RDRX register accessing
;    2138  1         ! to  be transportable between the PDP-11 and
;    2139  1         ! VAX Diagnostic Supervisors.
;    2140  1         !
;    2141  1         ! This  also  defines an access algorithm for
;    2142  1         ! VAX to allow field reference to MBA address
;    2143  1         ! space  without  generating  machine  checks.
;    2144  1         !-
;    2145  1
;    2146  1         RDRX [0, P, S, E] =
;    2147  2             begin
;    2148  2
;    2149  2             local
;    2150  2                 RC$S_REG;
;    2151  2
;    2152  2             RC$S_REG = .(RDRX + %upval*0)<0, %bpval, 0>;
;    2153  2             RC$S_REG
;    2154  2             end
;    2155  1             <P, S, E>;
;    2156  1
```

```
;    2157  1     %sbttl 'GLOBAL DATA SECTION'
;    2158  1     !+
;    2159  1     ! The global data section contains data that are used
;    2160  1     ! in more than one test or module.
;    2161  1     !-
;    2162  1
;    2163  1     global
;    2164  1         !
;    2165  1         ! Communication area Declarations
;    2166  1         !
;    2167  1         COM_AREA : blockvector [REC_ALLOCATE + SND_ALLOCATE + HDR_SIZ, 2, word],
;    2168  1         HEAD_AREA : ref block [4, word] field (HDR_FIELD),
;    2169  1         RECEIVE_RING : ref blockvector [REC_ALLOCATE, 2, word] field (DSC_FIELD),
;    2170  1         SEND_RING : ref blockvector [SND_ALLOCATE, 2, word] field (DSC_FIELD),
;    2171  1         REC_ENVELOPE : blockvector [REC_ALLOCATE, RB_SIZE + 2, word] field (ENV_FIELD),
;    2172  1         SND_ENVELOPE : blockvector [SND_ALLOCATE, SB_SIZE + 2, word] field (ENV_FIELD),
;    2173  1         REC_BUF : block [RECB_SIZE, word] field (RECB_FIELD),
;    2174  1         SND_BUF : vector [SNDB_SIZE, word],
;    2175  1         OUT$STD_BUF : BLOCKVECTOR [REC_ALLOCATE, 2, WORD] FIELD (OUT$FIELD),
;    2176  1         RET_EN$AD : ref block [RB_SIZE + 2, word] field (ENV_FIELD);
;    2177  1
;    2178  1     global bind
;    2179  1         !
;    2180  1         ! Diagnostic supervisor printing ascii format strings.
;    2181  1         !
;    2182  1         FMT1 = uplit (%asciz'%T'),                    !Print one ascii string pointer
;    2183  1         FMT2 = uplit (%asciz'%N%AFORMATTING PHYSICAL UNIT %D3'),
;    2184  1         FMT3 = uplit (%asciz'%N%ALOGICAL UNIT %D2%S%APHYSICAL UNIT %D2%S%AFORMAT ABORTED'),
;    2185  1         FMT4 = uplit (%asciz'%N%AFORMAT COMPLETED, %D2%S%A REVECTORED LBNS'),
;    2186  1     !    FMT5 = uplit (%asciz'%N%AFORMAT ABORTED, ERROR NUMBER %D2'),
;    2187  1         CRLF =  uplit (%asciz'%N'),
;    2188  1         !
;    2189  1         ! Formatter messages (so formatter can conserve space).
;    2190  1         !
;    2191  1         UNIT$_MSG    = uplit (%asciz'ENTER UNIT TO BE FORMATTED'),
;    2192  1         EXIST$_MSG   = uplit (%asciz'USE EXISTING BAD BLOCK INFORMATION'),
;    2193  1         DOWN$_MSG    = uplit (%asciz'USE DOWN LINE LOAD'),
;    2194  1         INACC$_MSG   = uplit (%asciz'CONTINUE IF BAD BLOCK INFORMATION IS INACCURATE'),
;    2195  1         DFLT$_MSG    = uplit (%asciz'%N%AEXISTING BAD BLOCK INFORMATION USED'),
;    2196  1         SERIAL$_MSG = uplit (%asciz'ENTER 8 CHARACTER SERIAL NUMBER'),
;    2197  1         DATE$_MSG    = uplit (%asciz'ENTER DATE IN MM-DD-YY FORMAT'),
;    2198  1         DATMSG       = uplit (%asciz'ENTER DATE  <MM-DD-YYYY>'),     !rev 9 date string
;    2199  1         !
;    2200  1         ! default strings
;    2201  1            DEF_DATE = uplit (%asciz'01-29-58'),
;    2202  1            DEF_SERIAL =  uplit (%asciz'40502179'),
;    2203  1         ! Ring base address declaration
;    2204  1         !
;    2205  1         RINGBASE = COM_AREA [REC_BASE],
;    2206  1         !
;    2207  1         !
;    2208  1         !
;    2209  1         MSGADR = REC_BUF [MSG_TXT];
```

```
;  2210  1
;  2211  1     global
;  2212  1         !
;  2213  1         ! Miscellaneous data declarations
;  2214  1         !
;  2215  1     !    OVSA : WORD,                                  !Overlay section starting adrs
;  2216  1         NXT_CRN : BYTE,                               !Stores next cmd ref number
;  2217  1         RET_STATUS : word initial (%o'000000'),       !Saves various return status codes
;  2218  1         LUN : word,                                   !Stores logical unit number being formatted
;  2219  1         PID_SAVE : word,                              !Saves proces indicator word
;  2220  1         UC_VER : byte,                                !Stores ucode version number
;  2221  1         NSD_SLOT : word,                              !Next send Descriptor slot
;  2222  1         NRD_SLOT : word,                              !Next receive Descriptor slot
;  2223  1         !
;  2224  1         ! Hardware P_Table storage declarations
;  2225  1         !
;  2226  1         RDRX_ADDR : ref RDRX field (ISD_FIELD),       !Controller register access structure
;  2227  1         VEC_ADDR : word,                              !Interrupt vector address storage
;  2228  1         BR_LEVEL : word,                              !Bus request level storage
;  2229  1         UNIT_NO : word,                               !Unit number to format storage
;  2230  1         PTBL_PTR : ref vector [4, word],              !Stores P_Table base address
;  2231  1     !
;  2232  1     ! Dup Protocol data structures
;  2233  1     !
;  2234  1     !+
;  2235  1     !Reserved field mask structure declaration
;  2236  1     !-
;  2237  1         RSVD_STRUCT : vector [4, word] preset (       !Reserved SA reg fields definitions
;  2238  1             [0] = %x'00FF',                           !Step one rsvd field
;  2239  1             [1] = %x'0000',                           !Step two rsvd field
;  2240  1             [2] = %x'0700',                           !Step three rsvd field
;  2241  1             [3] = %x'07FF'),                          !Step four rsvd & ucode field
;  2242  1     !+
;  2243  1     ! Init Sequence Data_Structure declaration
;  2244  1     !-
;  2245  1
;  2246  1         ISD_STRUCT : blockvector [4, 2, word] field (ISD_FIELD) preset (
;  2247  1             !
;  2248  1             ! Step one read SA register field declaration
;  2249  1             !
;  2250  1             [BLK0, WRD0, ERR_BIT] = 0,                        !Error bit
;  2251  1             [BLK0, WRD0, STP_FIELD] = %b'0001',               !All step bit fields
;  2252  1             [BLK0, WRD0, S1R_NV] = 0,                         !No host inter vec settable adrs
;  2253  1             [BLK0, WRD0, S1R_QB] = 1,                         !22-bit addressing support
;  2254  1             [BLK0, WRD0, S1R_DI] = 1,                         !Enhanced diag implementation
;  2255  1             [BLK0, WRD0, S1R_RSVD]= %o'377',                  !Reserved field
;  2256  1             !
;  2257  1             ! Step one write SA register field declaration
;  2258  1             !
;  2259  1             [BLK0, WRD1, ERR_BIT] = 1,                        !Error bit
;  2260  1             [BLK0, WRD1, S1W_WR]= 0,                          !Diag wrap around
;  2261  1             [BLK0, WRD1, S1W_CRING]= SND_SIZ,                 !Number of Send-ring slots 'pwrs of 2'
;  2262  1             [BLK0, WRD1, S1W_RRING]= REC_SIZ,                 !Number of Receive-ring slots 'pwrs of 2'
```

```
;    2263  1              [BLK0, WRD1, S1W_IE]= 0,                        !Init Sequence interrupt request
;    2264  1              [BLK0, WRD1, S1W_VADR]= %o'33',                 !Interrupt vector address
;    2265  1              !
;    2266  1              ! Step two read SA register field declaration
;    2267  1              !
;    2268  1              [BLK1, WRD0, ERR_BIT] = 0,                      !Error bit
;    2269  1              [BLK1, WRD0, STP_FIELD] = %b'0010',             !All step bit fields
;    2270  1              [BLK1, WRD0, S2R_PTYP]= 0,                      !Port type number
;    2271  1              [BLK1, WRD0, S2R_BIT7]= 1,                      !Echoed IE bit from step one write
;    2272  1              [BLK1, WRD0, S2R_WR]= 0,                        !Echoed bit 14 from step one write
;    2273  1              [BLK1, WRD0, S2R_CRING]= SND_SIZ,               !Echoed bits 3-5 from step one write
;    2274  1              [BLK1, WRD0, S2R_RRING]= REC_SIZ,               !Echoed bits 0-2 from step one write
;    2275  1              !
;    2276  1              ! Step two write SA register field declaration
;    2277  1              !
;    2278  1              [BLK1, WRD1, S2W_LRBASE]= RINGBASE,             !Ring base lower address
;    2279  1              !
;    2280  1              ! NOTE:
;    2281  1              ! The adapter purge interrupt is loaded within
;    2282  1              ! the bgninit code due to the inability to field
;    2283  1              ! select bits <1, 15, 0> from the ringbase adrs.
;    2284  1              ! [BLK1, WRD1, S2W_PI]= 0,                       !Adapter purge interrupt request
;    2285  1              !
;    2286  1              ! Step three read SA register field declaration
;    2287  1              !
;    2288  1              [BLK2, WRD0, ERR_BIT] = 0,                      !Error bit
;    2289  1              [BLK2, WRD0, STP_FIELD] = %b'0100',             !All step bit fields
;    2290  1              [BLK2, WRD0, S3R_RSVD]= %o'7',                  !Reserved
;    2291  1              [BLK2, WRD0, S3R_IE]= 0,                        !Echoed IE bit from step one write
;    2292  1              [BLK2, WRD0, S3R_VADR]= %o'33',                 !Echoed VADR from step one write
;    2293  1              !
;    2294  1              ! Step three write SA register field declaration
;    2295  1              !
;    2296  1              [BLK2, WRD1, S3W_PP]= 0,                        !Purge & Poll test request
;    2297  1              [BLK2, WRD1, S3W_HRBASE]= 0,                    !Ring base high address
;    2298  1              !
;    2299  1              ! Step four read SA register field declaration
;    2300  1              !
;    2301  1              [BLK3, WRD0, ERR_BIT] = 0,                      !Error bit
;    2302  1              [BLK3, WRD0, STP_FIELD] = %b'1000',             !All step bit fields
;    2303  1              [BLK3, WRD0, S4R_RSVD]= %o'7',                  !Reserved
;    2304  1              [BLK3, WRD0, S4R_MOD]= %o'17',          !Controller u-CODE version
;    2305  1              [BLK3, WRD0, S4R_VER]= %o'17',          !Controller u-CODE version
;    2306  1              !
;    2307  1              ! Step four write SA register field declaration
;    2308  1              !
;    2309  1              [BLK3, WRD1, S4W_RSVD]= %o'377',                !Reserved
;    2310  1              [BLK3, WRD1, S4W_BURST]= 0,                     !Max number longwords per NPR xfer
;    2311  1              [BLK3, WRD1, S4W_LF]= 0,                        !Last fail request
;    2312  1              [BLK3, WRD1, S4W_GO]= 0);                       !Go bit
;    2313  1
```

```
;    2314  1     %sbttl 'GLOBAL TEXT SECTION'
;    2315  1     !+
;    2316  1     ! The global text section contains format statements,
;    2317  1     ! messages, and ASCII information that are used in
;    2318  1     ! all modules.
;    2319  1     !-
;    2320  1
;    2321  1     global bind
;    2322  1         !
;    2323  1         ! Self-detected fatal port/controller errors
;    2324  1         !
;    2325  1         PFE_STRUCT = uplit (
;    2326  1                         uplit (%asciz'%N%A$FMTERR- UNRECOGNIZABLE ERROR CODE'),
;    2327  1                         uplit (%asciz'%N%A$FTLERR- ENVELOPE/PACKET READ (PARITY OR TIMEOUT)'),
;    2328  1                         uplit (%asciz'%N%A$FTLERR- ENVELOPE/PACKET WRITE (PARITY OR TIMEOUT)'),
;    2329  1                         uplit (%asciz'%N%A$FTLERR- CONTROLLER ROM AND RAM PARITY'),
;    2330  1                         uplit (%asciz'%N%A$FTLERR- CONTROLLER RAM PARITY'),
;    2331  1                         uplit (%asciz'%N%A$FTLERR- CONTROLLER ROM PARITY'),
;    2332  1                         uplit (%asciz'%N%A$FTLERR- RING READ (PARITY OR TIMEOUT)'),
;    2333  1                         uplit (%asciz'%N%A$FTLERR- RING WRITE (PARITY OR TIMEOUT)'),
;    2334  1                         uplit (%asciz'%N%A$FTLERR- INTERRUPT MASTER'),
;    2335  1                         uplit (%asciz'%N%A$FTLERR- HOST ACCESS TIMEOUT'),
;    2336  1                         uplit (%asciz'%N%A$FTLERR- CREDIT LIMIT EXCEEDED'),
;    2337  1                         uplit (%asciz'%N%A$FTLERR- UNIBUS MASTER ERROR'),
;    2338  1                         uplit (%asciz'%N%A$FTLERR- DIAGNOSTIC CONTROLLER FATAL ERROR'),
;    2339  1                         uplit (%asciz'%N%A$FTLERR- INSTRUCTION LOOP TIMEOUT'),
;    2340  1                         uplit (%asciz'%N%A$FTLERR- INVALID CONNECTION IDENTIFIER'),
;    2341  1                         uplit (%asciz'%N%A$FTLERR- INTERRUPT WRITE'),
;    2342  1                         uplit (%asciz'%N%A$FTLERR- MAINTENANCE READ/WRITE INVALID REGION IDENTIFIER'),
;    2343  1                         uplit (%asciz'%N%A$FTLERR- MAINTENANCE WRITE LOAD TO NON-LOADABLE CONTROLLER'),
;    2344  1                         uplit (%asciz'%N%A$FTLERR- CONTROLLER RAM ERROR (NON-PARITY)'),
;    2345  1                         uplit (%asciz'%N%A$FTLERR- INIT SEQUENCE ERROR'),
;    2346  1                         uplit (%asciz'%N%A$FTLERR- HIGH-LEVEL PROTOCOL INCOMPATIBILITY ERROR'),
;    2347  1                         uplit (%asciz'%N%A$FTLERR- PURGE/POLL HARDWARE FAILURE')) : vector [22],
;    2348  1         !
;    2349  1         ! Init code error and informational
;    2350  1         ! messages
;    2351  1         !
;    2352  1         PWR_MSG = uplit (%asciz'%N%A$FTLERR- INIT CODE RE-ENTERED DUE TO PWR FAIL'),
;    2353  1         ABO_MSG = uplit (%asciz'%N%A$FTLERR- ABORTING HOST AND REMOTE PROGRAMS'),
;    2354  1         TO_MANY_UNITS = uplit (%asciz'%N%A$FTLERR- ILLEGAL NUMBER OF UNITS SELECTED'),
;    2355  1         GOOD_NUM_UNITS = uplit (%asciz'%N%A$FTLERR- LIMIT OF SIXTEEN UNITS PER FORMATING SESSION'),
;    2356  1         BOOT_FAILURE = uplit (%asciz'%N%A$FTLERR- RDRX CONTROLLER INITIALIZATION ERROR'),
;    2357  1         PROTO_VIOLATION = uplit (%asciz'%N%A$FTLERR- PROTOCOL VIOLATION ERROR'),
;    2358  1         PORT_INIT_ERR = uplit (%asciz'%N%A$FTLERR- COMMUNICATION AREA INIT ERROR'),
;    2359  1         !
;    2360  1         ! Local load media DM module file name.ext
;    2361  1         !
;    2362  1     !    DM_FN$EXT = UPLIT (%ASCIZ'AZFMTR.SAV'),
;    2363  1         !
;    2364  1         ! Hardware parameter coding questions
;    2365  1         !
;    2366  1         HW_Q1_IP     = uplit (%asciz'IP REGISTER ADDRESS'),
```

```
;    2367  1          HW_Q2_VECTOR = uplit (%asciz'INTERRUPT VECTOR ADDRESS'),
;    2368  1          HW_Q3_BR =     uplit (%asciz'BUS REQUEST LEVEL'),
;    2369  1
;    2370  1          SW_Q1_UNIT =   uplit (%asciz'SOFTWARE QUESTIONS ONLY APPLY UNDER APT, ENTER UNIT NUMBER'),
;    2371  1          SW_Q2_MODE =   uplit (%asciz'ENTER MODE [1 = REFORMAT, 2 = RESTORE, 3 = RECONSTRUCT]'),
;    2372  1          !
;    2373  1          ! Program flow ascii string messages
;    2374  1          !
;    2375  1          !RP_MSG = uplit (%asciz'%N%AREPORT CODE SECTION'),
;    2376  1          !IN_MSG = uplit (%asciz'%N%AINIT CODE SECTION'),
;    2377  1          !AUTO_MSG = uplit (%asciz'%N%AAUTODROP SECTION'),
;    2378  1          !CLN_MSG = uplit (%asciz'%N%ACLEAN-UP CODE SECTION'),
;    2379  1          !DU_MSG = uplit (%asciz'%N%ADROP-UNIT CODE SECTION'),
;    2380  1          !AU_MSG = uplit (%asciz'%N%AADD UNIT CODE SECTION'),
;    2381  1          !T1_MSG = uplit (%asciz'%N%ATEST CODE SECTION 1'),
;    2382  1          !T2_MSG = uplit (%asciz'%N%ATEST CODE SECTION 2'),
;    2383  1          !T3_MSG = uplit (%asciz'%N%ATEST CODE SECTION 3'),
;    2384  1          !DINT_MSG = uplit (%asciz'%N%ADUP$I_SERVICE ROUTINE'),
;    2385  1          !IINT_MSG = uplit (%asciz'%N%AINT$I_SERVICE ROUTINE'),
;    2386  1          !
;    2387  1
;    2388  1      !       Formatter error returns
;    2389  1          FMT_ERR = uplit (
;    2390  1                  uplit (%asciz'%N%AFAILURE CLOSING FCTS'),
;    2391  1                  uplit (%asciz'%N%ARCT WRITE FAILURE'),
;    2392  1                  uplit (%asciz'%N%AREVECTOR LIMIT EXCEEDED'),
;    2393  1                  uplit (%asciz'%N%ASEEK FAILURE DURING ACTUAL FORMATTING'),
;    2394  1                  uplit (%asciz'%N%AFACTORY BAD BLOCK INFORMATION IS INACCESSABLE'),
;    2395  1                  uplit (%asciz'%N%AINITIAL FAILURE ACCESSING FCT'),
;    2396  1                  uplit (%asciz'%N%AUNIT IS NOT WINCHESTER OR CAN NOT BE SELECTED')) : vector [7],
;    2397  1
;    2398  1          !
;    2399  1          ! Error message structure
;    2400  1          !
;    2401  1          EMSG_STRUCT = uplit (
;    2402  1      uplit (%asciz'%N%A$FTLERR- RESPONCE STATUS ERROR'),
;    2403  1      uplit (%asciz'%N%A$FTLERR- HOST/CONTROLLER OUT OF SEQ'),
;    2404  1      uplit (%asciz'%N%A$FTLERR- REMOTE PROG NOT RUNNING'),
;    2405  1      uplit (%asciz'%N%A$FTLERR- UNKNOWN RETURN STATUS CODE'),
;    2406  1      uplit (%asciz'%N%A$FTLERR- COM AREA INIT ERROR'),
;    2407  1      uplit (%asciz'%N%A$FTLERR- PORT/HOST SYNC ERROR'),
;    2408  1      uplit (%asciz'%N%A$FTLERR- MESSAGE LENGTH ERROR'),
;    2409  1      uplit (%asciz'%N%A$FTLERR- UNKNOWN ENDCODE RECEIVED'),
;    2410  1      uplit (%asciz'%N%A$FTLERR- ADAPTOR PURGE ERROR'),
;    2411  1      uplit (%asciz'%N%A$FTLERR- UNKNOWN INTERRUPT'),
;    2412  1      uplit (%asciz'%N%A$FTLERR- INIT SEQ STEP TIMED OUT'),
;    2413  1      uplit (%asciz'%N%A$FTLERR- INIT SEQ COMPARE ERROR'),
;    2414  1      uplit (%asciz'%N%A$FTLERR- UNEXPECTED ATTENTION END MESSAGE RECEIVED'),
;    2415  1      uplit (%asciz'%N%A$FTLERR- UNEXPECTED COMMAND OPCODE IN END MESSAGE RECEIVED'),
;    2416  1      uplit (%asciz'%N%A$FTLERR- UNEXPECTED SERIOUS EXCEPTION END MESSAGE RECEIVED'),
;    2417  1      uplit (%asciz'%N%A$FTLERR- INVALID COMMAND END MESSAGE RECEIVED'),
;    2418  1      uplit (%asciz'%N%A$FTLERR- UNKNOWN MESSAGE TYPE RECEIVED'),
;    2419  1      uplit (%asciz'%N%A$FTLERR- OUTSTANDING COMMAND BUFFER FULL'),
```

```
;    2420  1     uplit (%asciz'%N%A$FTLERR- OUT STANDING COMMAND BUFFER OUT OF SYNC ERROR'),
;    2421  1     uplit (%asciz'%N%A$FTLERR- UNKNOWN MESSAGE NUMBER RECEIVED'),
;    2422  1     uplit (%asciz'%N%A$FTLERR- FILE READ ERROR'),
;    2423  1     uplit (%asciz'%N%A$FTLERR- PORT/CONTROLLER TIMEOUT ERROR')) : vector [22];
;    2424  1
;    2425  1     end
;    2426  1
;    2427  0     eludom


                                        .TITLE  ZRQB1 RDRX DISK FORMATTER
                                        .IDENT  /REV C /
                                        .ENABL  AMA


000000                                  .PSECT  $CODE$,  RO
000000     132     122     121  L$NAME::.ASCII  /ZRQ/
000003     102                          .ASCII  /B/
000004     000                          .BYTE   0
000005     000                          .BYTE   0
000006     000                          .BYTE   0
000007     000                          .BYTE   0
000010                          L$REV::
000010     103                          .ASCII  /C/
000011     060                          .ASCII  /0/
000012  000000G                 L$UNIT::.WORD   T$PTHV
000014  002260                  L$TIML::.WORD   2260
000016  000000G                 L$HPCP::.WORD   L$HARD
000020  000000G                 L$SPCP::.WORD   L$SOFT
000022  000142'                 L$HPTP::.WORD   L$HW
000024  000156'                 L$SPTP::.WORD   L$SW
000026  000000G                 L$LADP::.WORD   L$LAST
000030  000000                  L$STA:: .WORD   0
000032  000000                  L$CO::  .WORD   0
000034  000000                  L$DTYP::.WORD   0
000036  000000                  L$APT:: .WORD   0
000040  000126'                 L$DTP:: .WORD   L$DISPATCH
000042  000000                  L$PRIO::.WORD   0
000044  000000                  L$ENVI::.WORD   0
000046  000000                  L$EXP1::.WORD   0
000050                          L$MREV::
000050     003                          .BYTE   3
000051     003                          .BYTE   3
000052  000000                  L$EF::  .WORD   0
000054  000000                          .WORD   0
000056  000000                  L$SPC:: .WORD   0
000060  000000G                 L$DEVP::.WORD   L$DVTYP
000062  000000G                 L$REPP::.WORD   L$RPT
000064  000000                  L$EXP4::.WORD   0
000066  000000                  L$EXP5::.WORD   0
000070  000000G                 L$AUT:: .WORD   L$AU
000072  000000G                 L$DUT:: .WORD   L$DU
000074  000000                  L$LUN:: .WORD   0
```

```
000076  000000G                 L$DESP::.WORD   L$DESC
000100  104035                  L$LOAD::.WORD   -73743
000102  000130'                 L$ETP:: .WORD   L$ERRTBL
000104  000000G                 L$ICP:: .WORD   L$INIT
000106  000000G                 L$CCP:: .WORD   L$CLEAN
000110  000000G                 L$ACP:: .WORD   L$AUTO
000112  000164'                 L$PRT:: .WORD   L$PROT
000114  000000                  L$TEST::.WORD   0
000116  000000                  L$DLY:: .WORD   0
000120  000000                  L$HIME::.WORD   0
000122  000001                  L$CPU:: .WORD   1
000124  000001                  D$PCNT::.WORD   1
000126  000000G                 L$DISPATCH::
                                        .WORD   T1
000130                          ERRTYP::.BLKW   1
000132                          ERRNBR::.BLKW   1
000134                          ERRMSG::.BLKW   1
000136                          ERRBLK::.BLKW   1
000140  000000C                 L$HWLEN::
                                        .WORD   <<L$NDHW-L$HWLEN>/2>
000142  172150                  HW.IP.ADRS::
                                        .WORD   -5630
000144  000154                  HW.VECTOR::
                                        .WORD   154
000146  000004                  HW.BR.LEVEL::
                                        .WORD   4
000150  000000                  HW.UNIT.NO::
                                        .WORD   0
000152                          L$NDHW::.BLKW   1
000154  000000C                 L$SWLEN::
                                        .WORD   <<L$NDSW-L$SWLEN>/2>
000156  000000                  SW.UNIT.NO::
                                        .WORD   0
000160  000003                  SW.MODE::
                                        .WORD   3
000162                          L$NDSW::.BLKW   1
000164  177777                  L$PROT::.WORD   -1
000166  177777                                  .WORD   -1
000170  177777                                  .WORD   -1


000000                                  .PSECT  $PLIT$,  RO , D , GBL
000000    045     124     000   P.AAA:  .ASCII  /%T/<00>
000003    000                           .ASCII  <00>
000004    045     116     045   P.AAB:  .ASCII  /%N%/
000007    101     106     117           .ASCII  /AFO/
000012    122     115     101           .ASCII  /RMA/
000015    124     124     111           .ASCII  /TTI/
000020    116     107     040           .ASCII  /NG /
000023    120     110     131           .ASCII  /PHY/
000026    123     111     103           .ASCII  /SIC/
000031    101     114     040           .ASCII  /AL /
```

```
000034  125  116  111                .ASCII  /UNI/
000037  124  040  045                .ASCII  /T %/
000042  104  063  000                .ASCII  /D3/<00>
000045  000                         .ASCII  <00>
000046  045  116  045  P.AAC:       .ASCII  /%N%/
000051  101  114  117                .ASCII  /ALO/
000054  107  111  103                .ASCII  /GIC/
000057  101  114  040                .ASCII  /AL /
000062  125  116  111                .ASCII  /UNI/
000065  124  040  045                .ASCII  /T %/
000070  104  062  045                .ASCII  /D2%/
000073  123  045  101                .ASCII  /S%A/
000076  120  110  131                .ASCII  /PHY/
000101  123  111  103                .ASCII  /SIC/
000104  101  114  040                .ASCII  /AL /
000107  125  116  111                .ASCII  /UNI/
000112  124  040  045                .ASCII  /T %/
000115  104  062  045                .ASCII  /D2%/
000120  123  045  101                .ASCII  /S%A/
000123  106  117  122                .ASCII  /FOR/
000126  115  101  124                .ASCII  /MAT/
000131  040  101  102                .ASCII  / AB/
000134  117  122  124                .ASCII  /ORT/
000137  105  104  000                .ASCII  /ED/<00>
000142  045  116  045  P.AAD:       .ASCII  /%N%/
000145  101  106  117                .ASCII  /AFO/
000150  122  115  101                .ASCII  /RMA/
000153  124  040  103                .ASCII  /T C/
000156  117  115  120                .ASCII  /OMP/
000161  114  105  124                .ASCII  /LET/
000164  105  104  054                .ASCII  /ED,/
000167  040  045  104                .ASCII  / %D/
000172  062  045  123                .ASCII  /2%S/
000175  045  101  040                .ASCII  /%A /
000200  122  105  126                .ASCII  /REV/
000203  105  103  124                .ASCII  /ECT/
000206  117  122  105                .ASCII  /ORE/
000211  104  040  114                .ASCII  /D L/
000214  102  116  123                .ASCII  /BNS/
000217  000                         .ASCII  <00>
000220  045  116  000  P.AAE:       .ASCII  /%N/<00>
000223  000                         .ASCII  <00>
000224  105  116  124  P.AAF:       .ASCII  /ENT/
000227  105  122  040                .ASCII  /ER /
000232  125  116  111                .ASCII  /UNI/
000235  124  040  124                .ASCII  /T T/
000240  117  040  102                .ASCII  /O B/
000243  105  040  106                .ASCII  /E F/
000246  117  122  115                .ASCII  /ORM/
000251  101  124  124                .ASCII  /ATT/
000254  105  104  000                .ASCII  /ED/<00>
000257  000                         .ASCII  <00>
000260  125  123  105  P.AAG:       .ASCII  /USE/
```

```
000263  040  105  130          .ASCII  / EX/
000266  111  123  124          .ASCII  /IST/
000271  111  116  107          .ASCII  /ING/
000274  040  102  101          .ASCII  / BA/
000277  104  040  102          .ASCII  /D B/
000302  114  117  103          .ASCII  /LOC/
000305  113  040  111          .ASCII  /K I/
000310  116  106  117          .ASCII  /NFO/
000313  122  115  101          .ASCII  /RMA/
000316  124  111  117          .ASCII  /TIO/
000321  116  000  000          .ASCII  /N/<00><00>
000324  125  123  105  P.AAH:   .ASCII  /USE/
000327  040  104  117          .ASCII  / DO/
000332  127  116  040          .ASCII  /WN /
000335  114  111  116          .ASCII  /LIN/
000340  105  040  114          .ASCII  /E L/
000343  117  101  104          .ASCII  /OAD/
000346  000  000               .ASCII  <00><00>
000350  103  117  116  P.AAI:   .ASCII  /CON/
000353  124  111  116          .ASCII  /TIN/
000356  125  105  040          .ASCII  /UE /
000361  111  106  040          .ASCII  /IF /
000364  102  101  104          .ASCII  /BAD/
000367  040  102  114          .ASCII  / BL/
000372  117  103  113          .ASCII  /OCK/
000375  040  111  116          .ASCII  / IN/
000400  106  117  122          .ASCII  /FOR/
000403  115  101  124          .ASCII  /MAT/
000406  111  117  116          .ASCII  /ION/
000411  040  111  123          .ASCII  / IS/
000414  040  111  116          .ASCII  / IN/
000417  101  103  103          .ASCII  /ACC/
000422  125  122  101          .ASCII  /URA/
000425  124  105  000          .ASCII  /TE/<00>
000430  045  116  045  P.AAJ:   .ASCII  /%N%/
000433  101  105  130          .ASCII  /AEX/
000436  111  123  124          .ASCII  /IST/
000441  111  116  107          .ASCII  /ING/
000444  040  102  101          .ASCII  / BA/
000447  104  040  102          .ASCII  /D B/
000452  114  117  103          .ASCII  /LOC/
000455  113  040  111          .ASCII  /K I/
000460  116  106  117          .ASCII  /NFO/
000463  122  115  101          .ASCII  /RMA/
000466  124  111  117          .ASCII  /TIO/
000471  116  040  125          .ASCII  /N U/
000474  123  105  104          .ASCII  /SED/
000477  000                    .ASCII  <00>
000500  105  116  124  P.AAK:   .ASCII  /ENT/
000503  105  122  040          .ASCII  /ER /
000506  070  040  103          .ASCII  /8 C/
000511  110  101  122          .ASCII  /HAR/
000514  101  103  124          .ASCII  /ACT/
```

```
000517  105  122  040            .ASCII  /ER /
000522  123  105  122            .ASCII  /SER/
000525  111  101  114            .ASCII  /IAL/
000530  040  116  125            .ASCII  / NU/
000533  115  102  105            .ASCII  /MBE/
000536  122  000                 .ASCII  /R/<00>
000540  105  116  124  P.AAL:    .ASCII  /ENT/
000543  105  122  040            .ASCII  /ER /
000546  104  101  124            .ASCII  /DAT/
000551  105  040  111            .ASCII  /E I/
000554  116  040  115            .ASCII  /N M/
000557  115  055  104            .ASCII  /M-D/
000562  104  055  131            .ASCII  /D-Y/
000565  131  040  106            .ASCII  /Y F/
000570  117  122  115            .ASCII  /ORM/
000573  101  124  000            .ASCII  /AT/<00>
000576  105  116  124  P.AAM:    .ASCII  /ENT/
000601  105  122  040            .ASCII  /ER /
000604  104  101  124            .ASCII  /DAT/
000607  105  040  040            .ASCII  /E  /
000612  074  115  115            .ASCII  /<MM/
000615  055  104  104            .ASCII  /-DD/
000620  055  131  131            .ASCII  /-YY/
000623  131  131  076            .ASCII  /YY>/
000626  000  000                 .ASCII  <00><00>
000630  060  061  055  P.AAN:    .ASCII  /01-/
000633  062  071  055            .ASCII  /29-/
000636  065  070  000            .ASCII  /58/<00>
000641  000                      .ASCII  <00>
000642  064  060  065  P.AAO:    .ASCII  /405/
000645  060  062  061            .ASCII  /021/
000650  067  071  000            .ASCII  /79/<00>
000653  000                      .ASCII  <00>
000654  045  116  045  P.AAQ:    .ASCII  /%N%/
000657  101  044  106            .ASCII  /A$F/
000662  115  124  105            .ASCII  /MTE/
000665  122  122  055            .ASCII  /RR-/
000670  040  125  116            .ASCII  / UN/
000673  122  105  103            .ASCII  /REC/
000676  117  107  116            .ASCII  /OGN/
000701  111  132  101            .ASCII  /IZA/
000704  102  114  105            .ASCII  /BLE/
000707  040  105  122            .ASCII  / ER/
000712  122  117  122            .ASCII  /ROR/
000715  040  103  117            .ASCII  / CO/
000720  104  105  000            .ASCII  /DE/<00>
000723  000                      .ASCII  <00>
000724  045  116  045  P.AAR:    .ASCII  /%N%/
000727  101  044  106            .ASCII  /A$F/
000732  124  114  105            .ASCII  /TLE/
000735  122  122  055            .ASCII  /RR-/
000740  040  105  116            .ASCII  / EN/
000743  126  105  114            .ASCII  /VEL/
```

```
000746  117  120  105                .ASCII  /OPE/
000751  057  120  101                .ASCII  <57>/PA/
000754  103  113  105                .ASCII  /CKE/
000757  124  040  122                .ASCII  /T R/
000762  105  101  104                .ASCII  /EAD/
000765  040  050  120                .ASCII  / (P/
000770  101  122  111                .ASCII  /ARI/
000773  124  131  040                .ASCII  /TY /
000776  117  122  040                .ASCII  /OR /
001001  124  111  115                .ASCII  /TIM/
001004  105  117  125                .ASCII  /EOU/
001007  124  051  000                .ASCII  /T)/<00>
001012  045  116  045        P.AAS:  .ASCII  /%N%/
001015  101  044  106                .ASCII  /A$F/
001020  124  114  105                .ASCII  /TLE/
001023  122  122  055                .ASCII  /RR-/
001026  040  105  116                .ASCII  / EN/
001031  126  105  114                .ASCII  /VEL/
001034  117  120  105                .ASCII  /OPE/
001037  057  120  101                .ASCII  <57>/PA/
001042  103  113  105                .ASCII  /CKE/
001045  124  040  127                .ASCII  /T W/
001050  122  111  124                .ASCII  /RIT/
001053  105  040  050                .ASCII  /E (/
001056  120  101  122                .ASCII  /PAR/
001061  111  124  131                .ASCII  /ITY/
001064  040  117  122                .ASCII  / OR/
001067  040  124  111                .ASCII  / TI/
001072  115  105  117                .ASCII  /MEO/
001075  125  124  051                .ASCII  /UT)/
001100  000  000                    .ASCII  <00><00>
001102  045  116  045        P.AAT:  .ASCII  /%N%/
001105  101  044  106                .ASCII  /A$F/
001110  124  114  105                .ASCII  /TLE/
001113  122  122  055                .ASCII  /RR-/
001116  040  103  117                .ASCII  / CO/
001121  116  124  122                .ASCII  /NTR/
001124  117  114  114                .ASCII  /OLL/
001127  105  122  040                .ASCII  /ER /
001132  122  117  115                .ASCII  /ROM/
001135  040  101  116                .ASCII  / AN/
001140  104  040  122                .ASCII  /D R/
001143  101  115  040                .ASCII  /AM /
001146  120  101  122                .ASCII  /PAR/
001151  111  124  131                .ASCII  /ITY/
001154  000  000                    .ASCII  <00><00>
001156  045  116  045        P.AAU:  .ASCII  /%N%/
001161  101  044  106                .ASCII  /A$F/
001164  124  114  105                .ASCII  /TLE/
001167  122  122  055                .ASCII  /RR-/
001172  040  103  117                .ASCII  / CO/
001175  116  124  122                .ASCII  /NTR/
001200  117  114  114                .ASCII  /OLL/
```

```
001203    105    122    040                .ASCII  /ER /
001206    122    101    115                .ASCII  /RAM/
001211    040    120    101                .ASCII  / PA/
001214    122    111    124                .ASCII  /RIT/
001217    131    000    000                .ASCII  /Y/<00><00>
001222    045    116    045        P.AAV:  .ASCII  /%N%/
001225    101    044    106                .ASCII  /A$F/
001230    124    114    105                .ASCII  /TLE/
001233    122    122    055                .ASCII  /RR-/
001236    040    103    117                .ASCII  / CO/
001241    116    124    122                .ASCII  /NTR/
001244    117    114    114                .ASCII  /OLL/
001247    105    122    040                .ASCII  /ER /
001252    122    117    115                .ASCII  /ROM/
001255    040    120    101                .ASCII  / PA/
001260    122    111    124                .ASCII  /RIT/
001263    131    000    000                .ASCII  /Y/<00><00>
001266    045    116    045        P.AAW:  .ASCII  /%N%/
001271    101    044    106                .ASCII  /A$F/
001274    124    114    105                .ASCII  /TLE/
001277    122    122    055                .ASCII  /RR-/
001302    040    122    111                .ASCII  / RI/
001305    116    107    040                .ASCII  /NG /
001310    122    105    101                .ASCII  /REA/
001313    104    040    050                .ASCII  /D (/
001316    120    101    122                .ASCII  /PAR/
001321    111    124    131                .ASCII  /ITY/
001324    040    117    122                .ASCII  / OR/
001327    040    124    111                .ASCII  / TI/
001332    115    105    117                .ASCII  /MEO/
001335    125    124    051                .ASCII  /UT)/
001340    000    000                       .ASCII  <00><00>
001342    045    116    045        P.AAX:  .ASCII  /%N%/
001345    101    044    106                .ASCII  /A$F/
001350    124    114    105                .ASCII  /TLE/
001353    122    122    055                .ASCII  /RR-/
001356    040    122    111                .ASCII  / RI/
001361    116    107    040                .ASCII  /NG /
001364    127    122    111                .ASCII  /WRI/
001367    124    105    040                .ASCII  /TE /
001372    050    120    101                .ASCII  /(PA/
001375    122    111    124                .ASCII  /RIT/
001400    131    040    117                .ASCII  /Y O/
001403    122    040    124                .ASCII  /R T/
001406    111    115    105                .ASCII  /IME/
001411    117    125    124                .ASCII  /OUT/
001414    051    000                       .ASCII  /)/<00>
001416    045    116    045        P.AAY:  .ASCII  /%N%/
001421    101    044    106                .ASCII  /A$F/
001424    124    114    105                .ASCII  /TLE/
001427    122    122    055                .ASCII  /RR-/
001432    040    111    116                .ASCII  / IN/
001435    124    105    122                .ASCII  /TER/
```

```
001440  122  125  120            .ASCII  /RUP/
001443  124  040  115            .ASCII  /T M/
001446  101  123  124            .ASCII  /AST/
001451  105  122  000            .ASCII  /ER/<00>
001454  045  116  045  P.AAZ:     .ASCII  /%N%/
001457  101  044  106            .ASCII  /A$F/
001462  124  114  105            .ASCII  /TLE/
001465  122  122  055            .ASCII  /RR-/
001470  040  110  117            .ASCII  / HO/
001473  123  124  040            .ASCII  /ST /
001476  101  103  103            .ASCII  /ACC/
001501  105  123  123            .ASCII  /ESS/
001504  040  124  111            .ASCII  / TI/
001507  115  105  117            .ASCII  /MEO/
001512  125  124  000            .ASCII  /UT/<00>
001515  000                      .ASCII  <00>
001516  045  116  045  P.ABA:     .ASCII  /%N%/
001521  101  044  106            .ASCII  /A$F/
001524  124  114  105            .ASCII  /TLE/
001527  122  122  055            .ASCII  /RR-/
001532  040  103  122            .ASCII  / CR/
001535  105  104  111            .ASCII  /EDI/
001540  124  040  114            .ASCII  /T L/
001543  111  115  111            .ASCII  /IMI/
001546  124  040  105            .ASCII  /T E/
001551  130  103  105            .ASCII  /XCE/
001554  105  104  105            .ASCII  /EDE/
001557  104  000  000            .ASCII  /D/<00><00>
001562  045  116  045  P.ABB:     .ASCII  /%N%/
001565  101  044  106            .ASCII  /A$F/
001570  124  114  105            .ASCII  /TLE/
001573  122  122  055            .ASCII  /RR-/
001576  040  125  116            .ASCII  / UN/
001601  111  102  125            .ASCII  /IBU/
001604  123  040  115            .ASCII  /S M/
001607  101  123  124            .ASCII  /AST/
001612  105  122  040            .ASCII  /ER /
001615  105  122  122            .ASCII  /ERR/
001620  117  122  000            .ASCII  /OR/<00>
001623  000                      .ASCII  <00>
001624  045  116  045  P.ABC:     .ASCII  /%N%/
001627  101  044  106            .ASCII  /A$F/
001632  124  114  105            .ASCII  /TLE/
001635  122  122  055            .ASCII  /RR-/
001640  040  104  111            .ASCII  / DI/
001643  101  107  116            .ASCII  /AGN/
001646  117  123  124            .ASCII  /OST/
001651  111  103  040            .ASCII  /IC /
001654  103  117  116            .ASCII  /CON/
001657  124  122  117            .ASCII  /TRO/
001662  114  114  105            .ASCII  /LLE/
001665  122  040  106            .ASCII  /R F/
001670  101  124  101            .ASCII  /ATA/
```

```
001673  114  040  105                .ASCII  /L E/
001676  122  122  117                .ASCII  /RRO/
001701  122  000  000                .ASCII  /R/<00><00>
001704  045  116  045      P.ABD:    .ASCII  /%N%/
001707  101  044  106                .ASCII  /A$F/
001712  124  114  105                .ASCII  /TLE/
001715  122  122  055                .ASCII  /RR-/
001720  040  111  116                .ASCII  / IN/
001723  123  124  122                .ASCII  /STR/
001726  125  103  124                .ASCII  /UCT/
001731  111  117  116                .ASCII  /ION/
001734  040  114  117                .ASCII  / LO/
001737  117  120  040                .ASCII  /OP /
001742  124  111  115                .ASCII  /TIM/
001745  105  117  125                .ASCII  /EOU/
001750  124  000                     .ASCII  /T/<00>
001752  045  116  045      P.ABE:    .ASCII  /%N%/
001755  101  044  106                .ASCII  /A$F/
001760  124  114  105                .ASCII  /TLE/
001763  122  122  055                .ASCII  /RR-/
001766  040  111  116                .ASCII  / IN/
001771  126  101  114                .ASCII  /VAL/
001774  111  104  040                .ASCII  /ID /
001777  103  117  116                .ASCII  /CON/
002002  116  105  103                .ASCII  /NEC/
002005  124  111  117                .ASCII  /TIO/
002010  116  040  111                .ASCII  /N I/
002013  104  105  116                .ASCII  /DEN/
002016  124  111  106                .ASCII  /TIF/
002021  111  105  122                .ASCII  /IER/
002024  000  000                     .ASCII  <00><00>
002026  045  116  045      P.ABF:    .ASCII  /%N%/
002031  101  044  106                .ASCII  /A$F/
002034  124  114  105                .ASCII  /TLE/
002037  122  122  055                .ASCII  /RR-/
002042  040  111  116                .ASCII  / IN/
002045  124  105  122                .ASCII  /TER/
002050  122  125  120                .ASCII  /RUP/
002053  124  040  127                .ASCII  /T W/
002056  122  111  124                .ASCII  /RIT/
002061  105  000  000                .ASCII  /E/<00><00>
002064  045  116  045      P.ABG:    .ASCII  /%N%/
002067  101  044  106                .ASCII  /A$F/
002072  124  114  105                .ASCII  /TLE/
002075  122  122  055                .ASCII  /RR-/
002100  040  115  101                .ASCII  / MA/
002103  111  116  124                .ASCII  /INT/
002106  105  116  101                .ASCII  /ENA/
002111  116  103  105                .ASCII  /NCE/
002114  040  122  105                .ASCII  / RE/
002117  101  104  057                .ASCII  /AD/<57>
002122  127  122  111                .ASCII  /WRI/
002125  124  105  040                .ASCII  /TE /
```

J3


```
002130  111  116  126                .ASCII  /INV/
002133  101  114  111                .ASCII  /ALI/
002136  104  040  122                .ASCII  /D R/
002141  105  107  111                .ASCII  /EGI/
002144  117  116  040                .ASCII  /ON /
002147  111  104  105                .ASCII  /IDE/
002152  116  124  111                .ASCII  /NTI/
002155  106  111  105                .ASCII  /FIE/
002160  122  000                     .ASCII  /R/<00>
002162  045  116  045       P.ABH:   .ASCII  /%N%/
002165  101  044  106                .ASCII  /A$F/
002170  124  114  105                .ASCII  /TLE/
002173  122  122  055                .ASCII  /RR-/
002176  040  115  101                .ASCII  / MA/
002201  111  116  124                .ASCII  /INT/
002204  105  116  101                .ASCII  /ENA/
002207  116  103  105                .ASCII  /NCE/
002212  040  127  122                .ASCII  / WR/
002215  111  124  105                .ASCII  /ITE/
002220  040  114  117                .ASCII  / LO/
002223  101  104  040                .ASCII  /AD /
002226  124  117  040                .ASCII  /TO /
002231  116  117  116                .ASCII  /NON/
002234  055  114  117                .ASCII  /-LO/
002237  101  104  101                .ASCII  /ADA/
002242  102  114  105                .ASCII  /BLE/
002245  040  103  117                .ASCII  / CO/
002250  116  124  122                .ASCII  /NTR/
002253  117  114  114                .ASCII  /OLL/
002256  105  122  000                .ASCII  /ER/<00>
002261  000                         .ASCII  <00>
002262  045  116  045       P.ABI:   .ASCII  /%N%/
002265  101  044  106                .ASCII  /A$F/
002270  124  114  105                .ASCII  /TLE/
002273  122  122  055                .ASCII  /RR-/
002276  040  103  117                .ASCII  / CO/
002301  116  124  122                .ASCII  /NTR/
002304  117  114  114                .ASCII  /OLL/
002307  105  122  040                .ASCII  /ER /
002312  122  101  115                .ASCII  /RAM/
002315  040  105  122                .ASCII  / ER/
002320  122  117  122                .ASCII  /ROR/
002323  040  050  116                .ASCII  / (N/
002326  117  116  055                .ASCII  /ON-/
002331  120  101  122                .ASCII  /PAR/
002334  111  124  131                .ASCII  /ITY/
002337  051  000  000                .ASCII  /)/<00><00>
002342  045  116  045       P.ABJ:   .ASCII  /%N%/
002345  101  044  106                .ASCII  /A$F/
002350  124  114  105                .ASCII  /TLE/
002353  122  122  055                .ASCII  /RR-/
002356  040  111  116                .ASCII  / IN/
002361  111  124  040                .ASCII  /IT /
```

```
002364  123     105     121             .ASCII  /SEQ/
002367  125     105     116             .ASCII  /UEN/
002372  103     105     040             .ASCII  /CE /
002375  105     122     122             .ASCII  /ERR/
002400  117     122     000             .ASCII  /OR/<00>
002403  000                             .ASCII  <00>
002404  045     116     045     P.ABK:  .ASCII  /%N%/
002407  101     044     106             .ASCII  /A$F/
002412  124     114     105             .ASCII  /TLE/
002415  122     122     055             .ASCII  /RR-/
002420  040     110     111             .ASCII  / HI/
002423  107     110     055             .ASCII  /GH-/
002426  114     105     126             .ASCII  /LEV/
002431  105     114     040             .ASCII  /EL /
002434  120     122     117             .ASCII  /PRO/
002437  124     117     103             .ASCII  /TOC/
002442  117     114     040             .ASCII  /OL /
002445  111     116     103             .ASCII  /INC/
002450  117     115     120             .ASCII  /OMP/
002453  101     124     111             .ASCII  /ATI/
002456  102     111     114             .ASCII  /BIL/
002461  111     124     131             .ASCII  /ITY/
002464  040     105     122             .ASCII  / ER/
002467  122     117     122             .ASCII  /ROR/
002472  000     000                     .ASCII  <00><00>
002474  045     116     045     P.ABL:  .ASCII  /%N%/
002477  101     044     106             .ASCII  /A$F/
002502  124     114     105             .ASCII  /TLE/
002505  122     122     055             .ASCII  /RR-/
002510  040     120     125             .ASCII  / PU/
002513  122     107     105             .ASCII  /RGE/
002516  057     120     117             .ASCII  <57>/PO/
002521  114     114     040             .ASCII  /LL /
002524  110     101     122             .ASCII  /HAR/
002527  104     127     101             .ASCII  /DWA/
002532  122     105     040             .ASCII  /RE /
002535  106     101     111             .ASCII  /FAI/
002540  114     125     122             .ASCII  /LUR/
002543  105     000     000             .ASCII  /E/<00><00>
002546  000654'                 P.AAP:  .WORD   P.AAQ
002550  000724'                         .WORD   P.AAR
002552  001012'                         .WORD   P.AAS
002554  001102'                         .WORD   P.AAT
002556  001156'                         .WORD   P.AAU
002560  001222'                         .WORD   P.AAV
002562  001266'                         .WORD   P.AAW
002564  001342'                         .WORD   P.AAX
002566  001416'                         .WORD   P.AAY
002570  001454'                         .WORD   P.AAZ
002572  001516'                         .WORD   P.ABA
002574  001562'                         .WORD   P.ABB
002576  001624'                         .WORD   P.ABC
002600  001704'                         .WORD   P.ABD
```

L3


```
002602  001752'                                 .WORD   P.ABE
002604  002026'                                 .WORD   P.ABF
002606  002064'                                 .WORD   P.ABG
002610  002162'                                 .WORD   P.ABH
002612  002262'                                 .WORD   P.ABI
002614  002342'                                 .WORD   P.ABJ
002616  002404'                                 .WORD   P.ABK
002620  002474'                                 .WORD   P.ABL
002622     045     116     045          P.ABM:  .ASCII  /%N%/
002625     101     044     106                  .ASCII  /A$F/
002630     124     114     105                  .ASCII  /TLE/
002633     122     122     055                  .ASCII  /RR-/
002636     040     111     116                  .ASCII  / IN/
002641     111     124     040                  .ASCII  /IT /
002644     103     117     104                  .ASCII  /COD/
002647     105     040     122                  .ASCII  /E R/
002652     105     055     105                  .ASCII  /E-E/
002655     116     124     105                  .ASCII  /NTE/
002660     122     105     104                  .ASCII  /RED/
002663     040     104     125                  .ASCII  / DU/
002666     105     040     124                  .ASCII  /E T/
002671     117     040     120                  .ASCII  /O P/
002674     127     122     040                  .ASCII  /WR /
002677     106     101     111                  .ASCII  /FAI/
002702     114     000                          .ASCII  /L/<00>
002704     045     116     045          P.ABN:  .ASCII  /%N%/
002707     101     044     106                  .ASCII  /A$F/
002712     124     114     105                  .ASCII  /TLE/
002715     122     122     055                  .ASCII  /RR-/
002720     040     101     102                  .ASCII  / AB/
002723     117     122     124                  .ASCII  /ORT/
002726     111     116     107                  .ASCII  /ING/
002731     040     110     117                  .ASCII  / HO/
002734     123     124     040                  .ASCII  /ST /
002737     101     116     104                  .ASCII  /AND/
002742     040     122     105                  .ASCII  / RE/
002745     115     117     124                  .ASCII  /MOT/
002750     105     040     120                  .ASCII  /E P/
002753     122     117     107                  .ASCII  /ROG/
002756     122     101     115                  .ASCII  /RAM/
002761     123     000     000                  .ASCII  /S/<00><00>
002764     045     116     045          P.ABO:  .ASCII  /%N%/
002767     101     044     106                  .ASCII  /A$F/
002772     124     114     105                  .ASCII  /TLE/
002775     122     122     055                  .ASCII  /RR-/
003000     040     111     114                  .ASCII  / IL/
003003     114     105     107                  .ASCII  /LEG/
003006     101     114     040                  .ASCII  /AL /
003011     116     125     115                  .ASCII  /NUM/
003014     102     105     122                  .ASCII  /BER/
003017     040     117     106                  .ASCII  / OF/
003022     040     125     116                  .ASCII  / UN/
003025     111     124     123                  .ASCII  /ITS/
```

```
003030      040     123     105             .ASCII  / SE/
003033      114     105     103             .ASCII  /LEC/
003036      124     105     104             .ASCII  /TED/
003041      000                             .ASCII  <00>
003042      045     116     045     P.ABP:  .ASCII  /%N%/
003045      101     044     106             .ASCII  /A$F/
003050      124     114     105             .ASCII  /TLE/
003053      122     122     055             .ASCII  /RR-/
003056      040     114     111             .ASCII  / LI/
003061      115     111     124             .ASCII  /MIT/
003064      040     117     106             .ASCII  / OF/
003067      040     123     111             .ASCII  / SI/
003072      130     124     105             .ASCII  /XTE/
003075      105     116     040             .ASCII  /EN /
003100      125     116     111             .ASCII  /UNI/
003103      124     123     040             .ASCII  /TS /
003106      120     105     122             .ASCII  /PER/
003111      040     106     117             .ASCII  / FO/
003114      122     115     101             .ASCII  /RMA/
003117      124     111     116             .ASCII  /TIN/
003122      107     040     123             .ASCII  /G S/
003125      105     123     123             .ASCII  /ESS/
003130      111     117     116             .ASCII  /ION/
003133      000                             .ASCII  <00>
003134      045     116     045     P.ABQ:  .ASCII  /%N%/
003137      101     044     106             .ASCII  /A$F/
003142      124     114     105             .ASCII  /TLE/
003145      122     122     055             .ASCII  /RR-/
003150      040     122     104             .ASCII  / RD/
003153      122     130     040             .ASCII  /RX /
003156      103     117     116             .ASCII  /CON/
003161      124     122     117             .ASCII  /TRO/
003164      114     114     105             .ASCII  /LLE/
003167      122     040     111             .ASCII  /R I/
003172      116     111     124             .ASCII  /NIT/
003175      111     101     114             .ASCII  /IAL/
003200      111     132     101             .ASCII  /IZA/
003203      124     111     117             .ASCII  /TIO/
003206      116     040     105             .ASCII  /N E/
003211      122     122     117             .ASCII  /RRO/
003214      122     000                     .ASCII  /R/<00>
003216      045     116     045     P.ABR:  .ASCII  /%N%/
003221      101     044     106             .ASCII  /A$F/
003224      124     114     105             .ASCII  /TLE/
003227      122     122     055             .ASCII  /RR-/
003232      040     120     122             .ASCII  / PR/
003235      117     124     117             .ASCII  /OTO/
003240      103     117     114             .ASCII  /COL/
003243      040     126     111             .ASCII  / VI/
003246      117     114     101             .ASCII  /OLA/
003251      124     111     117             .ASCII  /TIO/
003254      116     040     105             .ASCII  /N E/
003257      122     122     117             .ASCII  /RRO/
```

```
003262  122  000            .ASCII  /R/<00>
003264  045  116  045  P.ABS: .ASCII  /%N%/
003267  101  044  106       .ASCII  /A$F/
003272  124  114  105       .ASCII  /TLE/
003275  122  122  055       .ASCII  /RR-/
003300  040  103  117       .ASCII  / CO/
003303  115  115  125       .ASCII  /MMU/
003306  116  111  103       .ASCII  /NIC/
003311  101  124  111       .ASCII  /ATI/
003314  117  116  040       .ASCII  /ON /
003317  101  122  105       .ASCII  /ARE/
003322  101  040  111       .ASCII  /A I/
003325  116  111  124       .ASCII  /NIT/
003330  040  105  122       .ASCII  / ER/
003333  122  117  122       .ASCII  /ROR/
003336  000  000            .ASCII  <00><00>
003340  111  120  040  P.ABT: .ASCII  /IP /
003343  122  105  107       .ASCII  /REG/
003346  111  123  124       .ASCII  /IST/
003351  105  122  040       .ASCII  /ER /
003354  101  104  104       .ASCII  /ADD/
003357  122  105  123       .ASCII  /RES/
003362  123  000            .ASCII  /S/<00>
003364  111  116  124  P.ABU: .ASCII  /INT/
003367  105  122  122       .ASCII  /ERR/
003372  125  120  124       .ASCII  /UPT/
003375  040  126  105       .ASCII  / VE/
003400  103  124  117       .ASCII  /CTO/
003403  122  040  101       .ASCII  /R A/
003406  104  104  122       .ASCII  /DDR/
003411  105  123  123       .ASCII  /ESS/
003414  000  000            .ASCII  <00><00>
003416  102  125  123  P.ABV: .ASCII  /BUS/
003421  040  122  105       .ASCII  / RE/
003424  121  125  105       .ASCII  /QUE/
003427  123  124  040       .ASCII  /ST /
003432  114  105  126       .ASCII  /LEV/
003435  105  114  000       .ASCII  /EL/<00>
003440  123  117  106  P.ABW: .ASCII  /SOF/
003443  124  127  101       .ASCII  /TWA/
003446  122  105  040       .ASCII  /RE /
003451  121  125  105       .ASCII  /QUE/
003454  123  124  111       .ASCII  /STI/
003457  117  116  123       .ASCII  /ONS/
003462  040  117  116       .ASCII  / ON/
003465  114  131  040       .ASCII  /LY /
003470  101  120  120       .ASCII  /APP/
003473  114  131  040       .ASCII  /LY /
003476  125  116  104       .ASCII  /UND/
003501  105  122  040       .ASCII  /ER /
003504  101  120  124       .ASCII  /APT/
003507  056  040  105       .ASCII  /. E/
003512  116  124  105       .ASCII  /NTE/
```

```
003515  122  040  125                .ASCII  /R U/
003520  116  111  124                .ASCII  /NIT/
003523  040  116  125                .ASCII  / NU/
003526  115  102  105                .ASCII  /MBE/
003531  122  000  000                .ASCII  /R/<00><00>
003534  105  116  124  P.ABX:        .ASCII  /ENT/
003537  105  122  040                .ASCII  /ER /
003542  115  117  104                .ASCII  /MOD/
003545  105  040  133                .ASCII  /E [/
003550  061  040  075                .ASCII  /1 =/
003553  040  122  105                .ASCII  / RE/
003556  106  117  122                .ASCII  /FOR/
003561  115  101  124                .ASCII  /MAT/
003564  054  040  062                .ASCII  /, 2/
003567  040  075  040                .ASCII  / = /
003572  122  105  123                .ASCII  /RES/
003575  124  117  122                .ASCII  /TOR/
003600  105  054  040                .ASCII1 /E, /
003603  063  040  075                .ASCII  /3 =/
003606  040  122  105                .ASCII  / RE/
003611  103  117  116                .ASCII  /CON/
003614  123  124  122                .ASCII  /STR/
003617  125  103  124                .ASCII  /UCT/
003622  135  000                     .ASCII  /]/<00>
003624  045  116  045  P.ABZ:        .ASCII  /%N%/
003627  101  106  101                .ASCII  /AFA/
003632  111  114  125                .ASCII  /ILU/
003635  122  105  040                .ASCII  /RE /
003640  103  114  117                .ASCII  /CLO/
003643  123  111  116                .ASCII  /SIN/
003646  107  040  106                .ASCII  /G F/
003651  103  124  123                .ASCII  /CTS/
003654  000  000                     .ASCII  <00><00>
003656  045  116  045  P.ACA:        .ASCII  /%N%/
003661  101  122  103                .ASCII  /ARC/
003664  124  040  127                .ASCII  /T W/
003667  122  111  124                .ASCII  /RIT/
003672  105  040  106                .ASCII  /E F/
003675  101  111  114                .ASCII  /AIL/
003700  125  122  105                .ASCII  /URE/
003703  000                         .ASCII  <00>
003704  045  116  045  P.ACB:        .ASCII  /%N%/
003707  101  122  105                .ASCII  /ARE/
003712  126  105  103                .ASCII  /VEC/
003715  124  117  122                .ASCII  /TOR/
003720  040  114  111                .ASCII  / LI/
003723  115  111  124                .ASCII  /MIT/
003726  040  105  130                .ASCII  / EX/
003731  103  105  105                .ASCII  /CEE/
003734  104  105  104                .ASCII  /DED/
003737  000                         .ASCII  <00>
003740  045  116  045  P.ACC:        .ASCII  /%N%/
003743  101  123  105                .ASCII  /ASE/
```

```
003746      105     113     040                 .ASCII  /EK /
003751      106     101     111                 .ASCII  /FAI/
003754      114     125     122                 .ASCII  /LUR/
003757      105     040     104                 .ASCII  /E D/
003762      125     122     111                 .ASCII  /URI/
003765      116     107     040                 .ASCII  /NG /
003770      101     103     124                 .ASCII  /ACT/
003773      125     101     114                 .ASCII  /UAL/
003776      040     106     117                 .ASCII  / FO/
004001      122     115     101                 .ASCII  /RMA/
004004      124     124     111                 .ASCII  /TTI/
004007      116     107     000                 .ASCII  /NG/<00>
004012      045     116     045         P.ACD:  .ASCII  /%N%/
004015      101     106     101                 .ASCII  /AFA/
004020      103     124     117                 .ASCII  /CTO/
004023      122     131     040                 .ASCII  /RY /
004026      102     101     104                 .ASCII  /BAD/
004031      040     102     114                 .ASCII  / BL/
004034      117     103     113                 .ASCII  /OCK/
004037      040     111     116                 .ASCII  / IN/
004042      106     117     122                 .ASCII  /FOR/
004045      115     101     124                 .ASCII  /MAT/
004050      111     117     116                 .ASCII  /ION/
004053      040     111     123                 .ASCII  / IS/
004056      040     111     116                 .ASCII  / IN/
004061      101     103     103                 .ASCII  /ACC/
004064      105     123     123                 .ASCII  /ESS/
004067      101     102     114                 .ASCII  /ABL/
004072      105     000                         .ASCII  /E/<00>
004074      045     116     045         P.ACE:  .ASCII  /%N%/
004077      101     111     116                 .ASCII  /AIN/
004102      111     124     111                 .ASCII  /ITI/
004105      101     114     040                 .ASCII  /AL /
004110      106     101     111                 .ASCII  /FAI/
004113      114     125     122                 .ASCII  /LUR/
004116      105     040     101                 .ASCII  /E A/
004121      103     103     105                 .ASCII  /CCE/
004124      123     123     111                 .ASCII  /SSI/
004127      116     107     040                 .ASCII  /NG /
004132      106     103     124                 .ASCII  /FCT/
004135      000                                 .ASCII  <00>
004136      045     116     045         P.ACF:  .ASCII  /%N%/
004141      101     125     116                 .ASCII  /AUN/
004144      111     124     040                 .ASCII  /IT /
004147      111     123     040                 .ASCII  /IS /
004152      116     117     124                 .ASCII  /NOT/
004155      040     127     111                 .ASCII  / WI/
004160      116     103     110                 .ASCII  /NCH/
004163      105     123     124                 .ASCII  /EST/
004166      105     122     040                 .ASCII  /ER /
004171      117     122     040                 .ASCII  /OR /
004174      103     101     116                 .ASCII  /CAN/
004177      040     116     117                 .ASCII  / NO/
```

```
004202      124     040     102                 .ASCII  /T B/
004205      105     040     123                 .ASCII  /E S/
004210      105     114     105                 .ASCII  /ELE/
004213      103     124     105                 .ASCII  /CTE/
004216      104     000                         .ASCII  /D/<00>
004220  003624'                         P.ABY:  .WORD   P.ABZ
004222  003656'                                 .WORD   P.ACA
004224  003704'                                 .WORD   P.ACB
004226  003740'                                 .WORD   P.ACC
004230  004012'                                 .WORD   P.ACD
004232  004074'                                 .WORD   P.ACE
004234  004136'                                 .WORD   P.ACF
004236      045     116     045         P.ACH:  .ASCII  /%N%/
004241      101     044     106                 .ASCII  /A$F/
004244      124     114     105                 .ASCII  /TLE/
004247      122     122     055                 .ASCII  /RR-/
004252      040     122     105                 .ASCII  / RE/
004255      123     120     117                 .ASCII  /SPO/
004260      116     103     105                 .ASCII  /NCE/
004263      040     123     124                 .ASCII  / ST/
004266      101     124     125                 .ASCII  /ATU/
004271      123     040     105                 .ASCII  /S E/
004274      122     122     117                 .ASCII  /RRO/
004277      122     000     000                 .ASCII  /R/<00><00>
004302      045     116     045         P.ACI:  .ASCII  /%N%/
004305      101     044     106                 .ASCII  /A$F/
004310      124     114     105                 .ASCII  /TLE/
004313      122     122     055                 .ASCII  /RR-/
004316      040     110     117                 .ASCII  / HO/
004321      123     124     057                 .ASCII  /ST/<57>
004324      103     117     116                 .ASCII  /CON/
004327      124     122     117                 .ASCII  /TRO/
004332      114     114     105                 .ASCII  /LLE/
004335      122     040     117                 .ASCII  /R O/
004340      125     124     040                 .ASCII  /UT /
004343      117     106     040                 .ASCII  /OF /
004346      123     105     121                 .ASCII  /SEQ/
004351      000                                 .ASCII  <00>
004352      045     116     045         P.ACJ:  .ASCII  /%N%/
004355      101     044     106                 .ASCII  /A$F/
004360      124     114     105                 .ASCII  /TLE/
004363      122     122     055                 .ASCII  /RR-/
004366      040     122     105                 .ASCII  / RE/
004371      115     117     124                 .ASCII  /MOT/
004374      105     040     120                 .ASCII  /E P/
004377      122     117     107                 .ASCII  /ROG/
004402      040     116     117                 .ASCII  / NO/
004405      124     040     122                 .ASCII  /T R/
004410      125     116     116                 .ASCII  /UNN/
004413      111     116     107                 .ASCII  /ING/
004416      000     000                         .ASCII  <00><00>
004420      045     116     045         P.ACK:  .ASCII  /%N%/
004423      101     044     106                 .ASCII  /A$F/
```

```
004426      124     114     105                 .ASCII  /TLE/
004431      122     122     055                 .ASCII  /RR-/
004434      040     125     116                 .ASCII  / UN/
004437      113     116     117                 .ASCII  /KNO/
004442      127     116     040                 .ASCII  /WN /
004445      122     105     124                 .ASCII  /RET/
004450      125     122     116                 .ASCII  /URN/
004453      040     123     124                 .ASCII  / ST/
004456      101     124     125                 .ASCII  /ATU/
004461      123     040     103                 .ASCII  /S C/
004464      117     104     105                 .ASCII  /ODE/
004467      000                                 .ASCII  <00>
004470      045     116     045         P.ACL:  .ASCII  /%N%/
004473      101     044     106                 .ASCII  /A$F/
004476      124     114     105                 .ASCII  /TLE/
004501      122     122     055                 .ASCII  /RR-/
004504      040     103     117                 .ASCII  / CO/
004507      115     040     101                 .ASCII  /M A/
004512      122     105     101                 .ASCII  /REA/
004515      040     111     116                 .ASCII  / IN/
004520      111     124     040                 .ASCII  /IT /
004523      105     122     122                 .ASCII  /ERR/
004526      117     122     000                 .ASCII  /OR/<00>
004531      000                                 .ASCII  <00>
004532      045     116     045         P.ACM:  .ASCII  /%N%/
004535      101     044     106                 .ASCII  /A$F/
004540      124     114     105                 .ASCII  /TLE/
004543      122     122     055                 .ASCII  /RR-/
004546      040     120     117                 .ASCII  / PO/
004551      122     124     057                 .ASCII  /RT/<57>
004554      110     117     123                 .ASCII  /HOS/
004557      124     040     123                 .ASCII  /T S/
004562      131     116     103                 .ASCII  /YNC/
004565      040     105     122                 .ASCII  / ER/
004570      122     117     122                 .ASCII  /ROR/
004573      000                                 .ASCII  <00>
004574      045     116     045         P.ACN:  .ASCII  /%N%/
004577      101     044     106                 .ASCII  /A$F/
004602      124     114     105                 .ASCII  /TLE/
004605      122     122     055                 .ASCII  /RR-/
004610      040     115     105                 .ASCII  / ME/
004613      123     123     101                 .ASCII  /SSA/
004616      107     105     040                 .ASCII  /GE /
004621      114     105     116                 .ASCII  /LEN/
004624      107     124     110                 .ASCII  /GTH/
004627      040     105     122                 .ASCII  / ER/
004632      122     117     122                 .ASCII  /ROR/
004635      000                                 .ASCII  <00>
004636      045     116     045         P.ACO:  .ASCII  /%N%/
004641      101     044     106                 .ASCII  /A$F/
004644      124     114     105                 .ASCII  /TLE/
004647      122     122     055                 .ASCII  /RR-/
004652      040     125     116                 .ASCII  / UN/
```

```
004655      113     116     117             .ASCII  /KNO/
004660      127     116     040             .ASCII  /WN /
004663      105     116     104             .ASCII  /END/
004666      103     117     104             .ASCII  /COD/
004671      105     040     122             .ASCII  /E R/
004674      105     103     105             .ASCII  /ECE/
004677      111     126     105             .ASCII  /IVE/
004702      104     000                     .ASCII  /D/<00>
004704      045     116     045     P.ACP:  .ASCII  /%N%/
004707      101     044     106             .ASCII  /A$F/
004712      124     114     105             .ASCII  /TLE/
004715      122     122     055             .ASCII  /RR-/
004720      040     101     104             .ASCII  / AD/
004723      101     120     124             .ASCII  /APT/
004726      117     122     040             .ASCII  /OR /
004731      120     125     122             .ASCII  /PUR/
004734      107     105     040             .ASCII  /GE /
004737      105     122     122             .ASCII  /ERR/
004742      117     122     000             .ASCII  /OR/<00>
004745      000                             .ASCII  <00>
004746      045     116     045     P.ACQ:  .ASCII  /%N%/
004751      101     044     106             .ASCII  /A$F/
004754      124     114     105             .ASCII  /TLE/
004757      122     122     055             .ASCII  /RR-/
004762      040     125     116             .ASCII  / UN/
004765      113     116     117             .ASCII  /KNO/
004770      127     116     040             .ASCII  /WN /
004773      111     116     124             .ASCII  /INT/
004776      105     122     122             .ASCII  /ERR/
005001      125     120     124             .ASCII  /UPT/
005004      000     000                     .ASCII  <00><00>
005006      045     116     045     P.ACR:  .ASCII  /%N%/
005011      101     044     106             .ASCII  /A$F/
005014      124     114     105             .ASCII  /TLE/
005017      122     122     055             .ASCII  /RR-/
005022      040     111     116             .ASCII  / IN/
005025      111     124     040             .ASCII  /IT /
005030      123     105     121             .ASCII  /SEQ/
005033      040     123     124             .ASCII  / ST/
005036      105     120     040             .ASCII  /EP /
005041      124     111     115             .ASCII  /TIM/
005044      105     104     040             .ASCII  /ED /
005047      117     125     124             .ASCII  /OUT/
005052      000     000                     .ASCII  <00><00>
005054      045     116     045     P.ACS:  .ASCII  /%N%/
005057      101     044     106             .ASCII  /A$F/
005062      124     114     105             .ASCII  /TLE/
005065      122     122     055             .ASCII  /RR-/
005070      040     111     116             .ASCII  / IN/
005073      111     124     040             .ASCII  /IT /
005076      123     105     121             .ASCII  /SEQ/
005101      040     103     117             .ASCII  / CO/
005104      115     120     101             .ASCII  /MPA/
```

```
005107  122  105  040                .ASCII  /RE /
005112  105  122  122                .ASCII  /ERR/
005115  117  122  000                .ASCII  /OR/<00>
005120  045  116  045        P.ACT:  .ASCII  /%N%/
005123  101  044  106                .ASCII  /A$F/
005126  124  114  105                .ASCII  /TLE/
005131  122  122  055                .ASCII  /RR-/
005134  040  125  116                .ASCII  / UN/
005137  105  130  120                .ASCII  /EXP/
005142  105  103  124                .ASCII  /ECT/
005145  105  104  040                .ASCII  /ED /
005150  101  124  124                .ASCII  /ATT/
005153  105  116  124                .ASCII  /ENT/
005156  111  117  116                .ASCII  /ION/
005161  040  105  116                .ASCII  / EN/
005164  104  040  115                .ASCII  /D M/
005167  105  123  123                .ASCII  /ESS/
005172  101  107  105                .ASCII  /AGE/
005175  040  122  105                .ASCII  / RE/
005200  103  105  111                .ASCII  /CEI/
005203  126  105  104                .ASCII  /VED/
005206  000  000                     .ASCII  <00><00>
005210  045  116  045        P.ACU:  .ASCII  /%N%/
005213  101  044  106                .ASCII  /A$F/
005216  124  114  105                .ASCII  /TLE/
005221  122  122  055                .ASCII  /RR-/
005224  040  125  116                .ASCII  / UN/
005227  105  130  120                .ASCII  /EXP/
005232  105  103  124                .ASCII  /ECT/
005235  105  104  040                .ASCII  /ED /
005240  103  117  115                .ASCII  /COM/
005243  115  101  116                .ASCII  /MAN/
005246  104  040  117                .ASCII  /D O/
005251  120  103  117                .ASCII  /PCO/
005254  104  105  040                .ASCII  /DE /
005257  111  116  040                .ASCII  /IN /
005262  105  116  104                .ASCII  /END/
005265  040  115  105                .ASCII  / ME/
005270  123  123  101                .ASCII  /SSA/
005273  107  105  040                .ASCII  /GE /
005276  122  105  103                .ASCII  /REC/
005301  105  111  126                .ASCII  /EIV/
005304  105  104  000                .ASCII  /ED/<00>
005307  000                         .ASCII  <00>
005310  045  116  045        P.ACV:  .ASCII  /%N%/
005313  101  044  106                .ASCII  /A$F/
005316  124  114  105                .ASCII  /TLE/
005321  122  122  055                .ASCII  /RR-/
005324  040  125  116                .ASCII  / UN/
005327  105  130  120                .ASCII  /EXP/
005332  105  103  124                .ASCII  /ECT/
005335  105  104  040                .ASCII  /ED /
005340  123  105  122                .ASCII  /SER/
```

```
005343      111     117     125                 .ASCII  /IOU/
005346      123     040     105                 .ASCII  /S E/
005351      130     103     105                 .ASCII  /XCE/
005354      120     124     111                 .ASCII  /PTI/
005357      117     116     040                 .ASCII  /ON /
005362      105     116     104                 .ASCII  /END/
005365      040     115     105                 .ASCII  / ME/
005370      123     123     101                 .ASCII  /SSA/
005373      107     105     040                 .ASCII  /GE /
005376      122     105     103                 .ASCII  /REC/
005401      105     111     126                 .ASCII  /EIV/
005404      105     104     000                 .ASCII  /ED/<00>
005407      000                                 .ASCII  <00>
005410      045     116     045         P.ACW:  .ASCII  /%N%/
005413      101     044     106                 .ASCII  /A$F/
005416      124     114     105                 .ASCII  /TLE/
005421      122     122     055                 .ASCII  /RR-/
005424      040     111     116                 .ASCII  / IN/
005427      126     101     114                 .ASCII  /VAL/
005432      111     104     040                 .ASCII  /ID /
005435      103     117     115                 .ASCII  /COM/
005440      115     101     116                 .ASCII  /MAN/
005443      104     040     105                 .ASCII  /D E/
005446      116     104     040                 .ASCII  /ND /
005451      115     105     123                 .ASCII  /MES/
005454      123     101     107                 .ASCII  /SAG/
005457      105     040     122                 .ASCII  /E R/
005462      105     103     105                 .ASCII  /ECE/
005465      111     126     105                 .ASCII  /IVE/
005470      104     000                         .ASCII  /D/<00>
005472      045     116     045         P.ACX:  .ASCII  /%N%/
005475      101     044     106                 .ASCII  /A$F/
005500      124     114     105                 .ASCII  /TLE/
005503      122     122     055                 .ASCII  /RR-/
005506      040     125     116                 .ASCII  / UN/
005511      113     116     117                 .ASCII  /KNO/
005514      127     116     040                 .ASCII  /WN /
005517      115     105     123                 .ASCII  /MES/
005522      123     101     107                 .ASCII  /SAG/
005525      105     040     124                 .ASCII  /E T/
005530      131     120     105                 .ASCII  /YPE/
005533      040     122     105                 .ASCII  / RE/
005536      103     105     111                 .ASCII  /CEI/
005541      126     105     104                 .ASCII  /VED/
005544      000     000                         .ASCII  <00><00>
005546      045     116     045         P.ACY:  .ASCII  /%N%/
005551      101     044     106                 .ASCII  /A$F/
005554      124     114     105                 .ASCII  /TLE/
005557      122     122     055                 .ASCII  /RR-/
005562      040     117     125                 .ASCII  / OU/
005565      124     123     124                 .ASCII  /TST/
005570      101     116     104                 .ASCII  /AND/
005573      111     116     107                 .ASCII  /ING/
```

```
005576      040     103     117                 .ASCII  / CO/
005601      115     115     101                 .ASCII  /MMA/
005604      116     104     040                 .ASCII  /ND /
005607      102     125     106                 .ASCII  /BUF/
005612      106     105     122                 .ASCII  /FER/
005615      040     106     125                 .ASCII  / FU/
005620      114     114     000                 .ASCII  /LL/<00>
005623      000                                 .ASCII  <00>
005624      045     116     045         P.ACZ:  .ASCII  /%N%/
005627      101     044     106                 .ASCII  /A$F/
005632      124     114     105                 .ASCII  /TLE/
005635      122     122     055                 .ASCII  /RR-/
005640      040     117     125                 .ASCII  / OU/
005643      124     040     123                 .ASCII  /T S/
005646      124     101     116                 .ASCII  /TAN/
005651      104     111     116                 .ASCII  /DIN/
005654      107     040     103                 .ASCII  /G C/
005657      117     115     115                 .ASCII  /OMM/
005662      101     116     104                 .ASCII  /AND/
005665      040     102     125                 .ASCII  / BU/
005670      106     106     105                 .ASCII  /FFE/
005673      122     040     117                 .ASCII  /R O/
005676      125     124     040                 .ASCII  /UT /
005701      117     106     040                 .ASCII  /OF /
005704      123     131     116                 .ASCII  /SYN/
005707      103     040     105                 .ASCII  /C E/
005712      122     122     117                 .ASCII  /RRO/
005715      122     000     000                 .ASCII  /R/<00><00>
005720      045     116     045         P.ADA:  .ASCII  /%N%/
005723      101     044     106                 .ASCII  /A$F/
005726      124     114     105                 .ASCII  /TLE/
005731      122     122     055                 .ASCII  /RR-/
005734      040     125     116                 .ASCII  / UN/
005737      113     116     117                 .ASCII  /KNO/
005742      127     116     040                 .ASCII  /WN /
005745      115     105     123                 .ASCII  /MES/
005750      123     101     107                 .ASCII  /SAG/
005753      105     040     116                 .ASCII  /E N/
005756      125     115     102                 .ASCII  /UMB/
005761      105     122     040                 .ASCII  /ER /
005764      122     105     103                 .ASCII  /REC/
005767      105     111     126                 .ASCII  /EIV/
005772      105     104     000                 .ASCII  /ED/<00>
005775      000                                 .ASCII  <00>
005776      045     116     045         P.ADB:  .ASCII  /%N%/
006001      101     044     106                 .ASCII  /A$F/
006004      124     114     105                 .ASCII  /TLE/
006007      122     122     055                 .ASCII  /RR-/
006012      040     106     111                 .ASCII  / FI/
006015      114     105     040                 .ASCII  /LE /
006020      122     105     101                 .ASCII  /REA/
006023      104     040     105                 .ASCII  /D E/
006026      122     122     117                 .ASCII  /RRO/
```

```
006031      122      000      000                    .ASCII  /R/<00><00>
006034      045      116      045          P.ADC:    .ASCII  /%N%/
006037      101      044      106                    .ASCII  /A$F/
006042      124      114      105                    .ASCII  /TLE/
006045      122      122      055                    .ASCII  /RR-/
006050      040      120      117                    .ASCII  / PO/
006053      122      124      057                    .ASCII  /RT/<57>
006056      103      117      116                    .ASCII  /CON/
006061      124      122      117                    .ASCII  /TRO/
006064      114      114      105                    .ASCII  /LLE/
006067      122      040      124                    .ASCII  /R T/
006072      111      115      105                    .ASCII  /IME/
006075      117      125      124                    .ASCII  /OUT/
006100      040      105      122                    .ASCII  / ER/
006103      122      117      122                    .ASCII  /ROR/
006106      000      000                             .ASCII  <00><00>
006110   004236'                          P.ACG:     .WORD   P.ACH
006112   004302'                                     .WORD   P.ACI
006114   004352'                                     .WORD   P.ACJ
006116   004420'                                     .WORD   P.ACK
006120   004470'                                     .WORD   P.ACL
006122   004532'                                     .WORD   P.ACM
006124   004574'                                     .WORD   P.ACN
006126   004636'                                     .WORD   P.ACO
006130   004704'                                     .WORD   P.ACP
006132   004746'                                     .WORD   P.ACQ
006134   005006'                                     .WORD   P.ACR
006136   005054'                                     .WORD   P.ACS
006140   005120'                                     .WORD   P.ACT
006142   005210'                                     .WORD   P.ACU
006144   005310'                                     .WORD   P.ACV
006146   005410'                                     .WORD   P.ACW
006150   005472'                                     .WORD   P.ACX
006152   005546'                                     .WORD   P.ACY
006154   005624'                                     .WORD   P.ACZ
006156   005720'                                     .WORD   P.ADA
006160   005776'                                     .WORD   P.ADB
006162   006034'                                     .WORD   P.ADC


000000                                       .PSECT  $GLOB$,  RO ,  D  ,  GBL
000000                               COM.AREA::
                                             .BLKW   24
000050                               HEAD.AREA::
                                             .BLKW   1
000052                               RECEIVE.RING::
                                             .BLKW   1
000054                               SEND.RING::
                                             .BLKW   1
000056                               REC.ENVELOPE::
                                             .BLKW   200
000456                               SND.ENVELOPE::
```

```
                                        .BLKW   130
000736                          REC.BUF::
                                        .BLKW   170
001316                          SND.BUF::
                                        .BLKW   45
001430                          OUT$STD.BUF::
                                        .BLKW   10
001450                          RET.EN$AD::
                                        .BLKW   1
001452                          NXT.CRN::
                                        .BLKB   1
                                        .EVEN
001454  000000                  RET.STATUS::
                                        .WORD   0
001456                          LUN::   .BLKW   1
001460                          PID.SAVE::
                                        .BLKW   1
001462                          UC.VER::.BLKB   1
                                        .EVEN
001464                          NSD.SLOT::
                                        .BLKW   1
001466                          NRD.SLOT::
                                        .BLKW   1
001470                          RDRX.ADDR::
                                        .BLKW   1
001472                          VEC.ADDR::
                                        .BLKW   1
001474                          BR.LEVEL::
                                        .BLKW   1
001476                          UNIT.NO::
                                        .BLKW   1
001500                          PTBL.PTR::
                                        .BLKW   1
001502  000377                  RSVD.STRUCT::
                                        .WORD   377
001504  000000                          .WORD   0
001506  003400                          .WORD   3400
001510  003777                          .WORD   3777
001512     377                  ISD.STRUCT::
                                        .BYTE   377
001513     013                          .BYTE   13
001514     033                          .BYTE   33
001515     222                          .BYTE   222
001516     222                          .BYTE   222
001517     020                          .BYTE   20
001520  000010'                         .WORD   RINGBASE
001522     033                          .BYTE   33
001523     047                          .BYTE   47
001524  000000                          .WORD   0
001526     377                          .BYTE   377
001527     107                          .BYTE   107
001530     000                          .BYTE   0
001531     377                          .BYTE   377
```

```
                        .GLOBL  L$SOFT, T$PTHV, L$RPT, L$INIT
                        .GLOBL  L$CLEAN, L$LAST, L$HARD, L$DVTYP
                        .GLOBL  L$DESC, L$DU, L$AU, L$AUTO, T1

000130'                 L$ERRTBL==          ERRTYP
000156'                 L$SW==              L$SWLEN+2
000142'                 L$HW==              L$HWLEN+2
000011'                 L$DEPO==            L$REV+1
000142'                 DFPTBL==            L$HWLEN+2
000156'                 SFPTBL==            L$SWLEN+2
000000'                 FMT1==              P.AAA
000004'                 FMT2==              P.AAB
000046'                 FMT3==              P.AAC
000142'                 FMT4==              P.AAD
000220'                 CRLF==              P.AAE
000224'                 UNIT$.MSG==         P.AAF
000260'                 EXIST$.MSG==        P.AAG
000324'                 DOWN$.MSG==         P.AAH
000350'                 INACC$.MSG==        P.AAI
000430'                 DFLT$.MSG==         P.AAJ
000500'                 SERIAL$.MSG==       P.AAK
000540'                 DATE$.MSG==         P.AAL
000576'                 DATMSG==            P.AAM
000630'                 DEF.DATE==          P.AAN
000642'                 DEF.SERIAL==        P.AAO
000010'                 RINGBASE==          COM.AREA+10
000740'                 MSGADR==            REC.BUF+2
002546'                 PFE.STRUCT==        P.AAP
002622'                 PWR.MSG==           P.ABM
002704'                 ABO.MSG==           P.ABN
002764'                 TO.MANY.UNITS==     P.ABO
003042'                 GOOD.NUM.UNITS==    P.ABP
003134'                 BOOT.FAILURE==      P.ABQ
003216'                 PROTO.VIOLATION==   P.ABR
003264'                 PORT.INIT.ERR==     P.ABS
003340'                 HW.Q1.IP==          P.ABT
003364'                 HW.Q2.VECTOR==      P.ABU
003416'                 HW.Q3.BR==          P.ABV
003440'                 SW.Q1.UNIT==        P.ABW
003534'                 SW.Q2.MODE==        P.ABX
004220'                 FMT.ERR==           P.ABY
006110'                 EMSG.STRUCT==       P.ACG


;                       PSECT SUMMARY
;
;       Psect Name      Words     Attributes
;        $CODE$            61       RO  ,  I  ,  LCL,  REL,  CON
;        $GLOB$           429       RO  ,  D  ,  GBL,  REL,  CON
```

```
;       $PLIT$                          1594       RO , D , GBL, REL, CON



;                               Library Statistics
;
;                                        -------- Symbols --------      Pages       Processing
;       File                              Total    Loaded   Percent      Mapped      Time
;
;  DISK$USER2:[YOUNG.FMT]ZRQBB0.L16;4       358       223        62          18        00:00.1



;                                       COMMAND QUALIFIERS

;       BLISS/PDP11/LIST ZRQB1.B16/EN:NOEIS/SOURCE=PAGE:53

; Size:         0 code + 2084 data words
; Run Time:         00:20.9
; Elapsed Time:     00:49.1
; Lines/CPU Min:     6954
; Lexemes/CPU-Min: 38982
; Memory Used:  207 pages
; Compilation Complete
```

```
;    0001  0     MODULE ZRQB2 (%TITLE 'RDRX DISK FORMATTER'
;    0002  0                   IDENT = 'REV C PATCH 0',
;    0003  0                   ADDRESSING_MODE (ABSOLUTE) ,
;    0004  0                   ENVIRONMENT (NOEIS)
;    0005  0                   ) =
;    0006  1     BEGIN
;    0007  1
;    0008  1     !       CONTROLLER FUNCTIONS
;    0009  1
;    0010  1     library 'ZRQBB0.L16';                   !Define RDRX Library module
;    0011  1
;    0012  1     require 'BLSMAC.REQ';                   !Define Bliss Macro Library
;    1548  1
;    1549  1     %sbttl 'MODULE DECLARATIONS'
```

```
;    1550  1     !+
;    1551  1     ! Structure declarations used within this module.
;    1552  1     !-
;    1553  1
;    1554  1     structure
;    1555  1
;    1556  1         !+
;    1557  1         ! RDRX  register  accessing  structure. This
;    1558  1         ! structure  allows  RDRX register accessing
;    1559  1         ! to  be transportable between the PDP-11 and
;    1560  1         ! VAX Diagnostic Supervisors.
;    1561  1         !
;    1562  1         ! This  also  defines an access algorithm for
;    1563  1         ! VAX to allow field reference to MBA address
;    1564  1         ! space  without  generating  machine  checks.
;    1565  1         !-
;    1566  1
;    1567  1         RDRX [O, P, S, E] =
;    1568  2             begin
;    1569  2
;    1570  2             local
;    1571  2                 RC$S_REG;
;    1572  2
;    1573  2             RC$S_REG = .(RDRX + %upval*O)<0, %bpval, 0>;
;    1574  2             RC$S_REG
;    1575  2             end
;    1576  1             <P, S, E>;
;    1577  1
```

```
;    1578  1     !+
;    1579  1     ! The  psect  named "code  or  $code$"  is redefinded here
;    1580  1     ! to  be  called "aa$code".  This is done to force the tkb
;    1581  1     ! linker to place the programs header information starting
;    1582  1     ! at absolute address  2000.  Then for consistency "a$code"
;    1583  1     ! is  used  inplace  of "code or $code" across all modules.
;    1584  1     !-
;    1585  1     !psect
;    1586  1     !    code = aa$code;
;    1587  1
;    1588  1     !+
;    1589  1     ! External  Routine declared outside this module.
;    1590  1     !-
;    1591  1     !external routine
;    1592  1     !     DECODE : NOVALUE,
;    1593  1     !     LOAD_FILE;
```

```
;    1594  1     !+
;    1595  1     ! External Declaration of  datums
;    1596  1     ! declared outside of this module.
;    1597  1     !-
;    1598  1
;    1599  1     external
;    1600  1         !
;    1601  1         ! Hardware question ascii string messages
;    1602  1         !
;    1603  1         HW_Q1_IP,                                   !H/W question 1 for IP reg address
;    1604  1         HW_Q2_VECTOR,                               !H/W question 2 for interrupt vector address
;    1605  1         HW_Q3_BR,                                   !H/W question 3 for bus req level
;    1606  1
;    1607  1         SW_Q1_UNIT,                                 !software question 1 for unit
;    1608  1         SW_Q2_MODE,                                 !software question 2 for unit
;    1609  1         !
;    1610  1         ! Formatting print string
;    1611  1         !
;    1612  1         FMT2,                                       !Notifies unit being formatted
;    1613  1         FMT3,                                       !Notifies format abort
;    1614  1         !
;    1615  1         ! Init code error and informational messages
;    1616  1         !
;    1617  1         PWR_MSG,
;    1618  1         ABO_MSG,
;    1619  1         TC_MANY_UNITS,
;    1620  1         GOOD_NUM_UNITS,
;    1621  1         !
;    1622  1         ! Miscellaneous external data declarations
;    1623  1         !
;    1624  1         LUN : byte,
;    1625  1         PTBL_PTR : ref vector [4, word],
;    1626  1         !
;    1627  1         ! Supervisor defined data declarations
;    1628  1         !
;    1629  1         L$UNIT,
;    1630  1         !
;    1631  1         ! Hardware P_Table storage declarations
;    1632  1         !
;    1633  1         RDRX_ADDR : ref RDRX field (ISD_FIELD),
;    1634  1         VEC_ADDR : word,
;    1635  1         BR_LEVEL : word,
;    1636  1         UNIT_NO : word,
;    1637  1         !
;    1638  1         ! Formatter data structures
;    1639  1         !
;    1640  1         ISD_STRUCT : blockvector [4, 2, word] field (ISD_FIELD);
;    1641  1
```

```
;    1642  1     %sbttl 'TYPE AND DESCRIPTION'
;    1643  1     !+
;    1644  1     ! Two lines of text will be printed to the operator (in addition to  the
;    1645  1     ! program name).  The first will come from the "DESCRIPT" macro at start
;    1646  1     ! up time and will identify the diagnostics.  The second will come  from
;    1647  1     ! the  "DEVTYPE"  macro  at hardware dialogue time and will identify the
;    1648  1     ! device under test.  The arugments of  both  macros  are  72  character
;    1649  1     ! ascii strings enclosed in parenthesises:
;    1650  1     !-
;    1651  1     DESCRIPT (%asciz'RD51/52 DISK FORMATTER');
;    1652  1     DEVTYP (%asciz'RQDX1 DISK DRIVE SUBSYSTEM');
```

```
;    1653  1     %sbttl 'HARDWARE PARAMETER CODING'
;    1654  1     !*
;    1655  1     ! The  hardware parameter  coding section contains macros
;    1656  1     ! that are used by the supervisor to build P-Tables.  The
;    1657  1     ! macros are not executed as macnine instructions but are
;    1658  1     ! interpreted  by the supervisor as data structures.  The
;    1659  1     ! macros allow the supervisor to establish communications
;    1660  1     ! with the operator.
;    1661  1     !-
;    1662  1     BGNHRD;
;    1663  1     GPRMA (HW_Q1_IP, %o'0', 0, %o'16000', %o'177777', YES, 1);      !Get RDRX Controller IP register
;    1664  1     GPRMA (HW_Q2_VECTOR, %o'2', 0, 4, %o'774', YES, 1);             !Get RDRX Interrupt Vector address
;    1665  1     GPRMD (HW_Q3_BR, %o'4', 0, %o'177777', 0, 7, YES, 1);           !Get RDRX Bus Request Priority
;    1666  1     ENDHRD;
```

```
;    1667  1     %sbttl 'SOFTWARE PARAMETER CODING SECTION'
;    1668  1     !+
;    1669  1     ! The  software  parameter coding section contains macros
;    1670  1     ! that are used by the supervisor to build P-Tables.  The
;    1671  1     ! macros are not executed as machine instructions but are
;    1672  1     ! interpreted by  the supervisor as data structures.  The
;    1673  1     ! macros allow the supervisor to establish communications
;    1674  1     ! with the operator.
;    1675  1     !-
;    1676  1     BGNSFT;
;    1677  1     !
;    1678  1     ! Software parameter coding is accomplished via DM
;    1679  1     ! micro  code.  This is to conform to DUP protocol
;    1680  1     ! standards.
;    1681  1     !
;    1682  1     GPRMD (SW_Q1_UNIT, %o'0', 0, %o'177777', 0, 3, YES, 0);          !get unit number
;    1683  1     GPRMD (SW_Q2_MODE, %o'2', 0, %o'177777', 1, 3, YES, 0);          !Get mode
;    1684  1     ENDSFT;
```

```
;  1   %sbttl 'REPORT CODING SECTION'
;  1   !+
;  1   ! The  statistical  report  coding section contains the PRINTS macros  that
;  1   ! will  be  used to generate statistical reports.  The 'BGNRPT' and 'ENDRPT'
;  1   ! macros  are  used  as  begining and ending directives for the coding  con-
;  1   ! tained  in  the  section.  However, an externally located DORPT call, or
;  1   ! a   print  command  from the operator, may be used to request  the  execu-
;  1   ! tion of the report coding.
;  1   !-
;  2   BGNRPT;
;  2   return;
;  2   !;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;  2   !;      THIS SECTION CONTAINS THE CODE FOR PRINTING
;  2   !;      STATISTICAL INFORMATION GATHERED BY THE DIAGNOSTIC.  IT IS
;  2   !;      EXECUTED BY THE OPERATOR COMMAND "PRINT" OR BY THE MACRO CALL
;  2   !;      "DORPT".  USE THE PRINTS MACRO TO PRINT THE INFORMATION.
;  2   !;      USE FORMAT STATEMENTS AS IN THE PRINTB/PRINTX MACROS.  IT IS
;  2   !;      THE PROGRAMMER'S RESPONSIBILTY TO DEVISE AND IMPLEMENT THE
;  2   !;      FORM AND CONTENT OF THE STATISTICS.
;  2   !;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;  1   ENDRPT;

                                               .TITLE  ZRQB2 RDRX DISK FORMATTER
                                               .IDENT  /REV C /
                                               .ENABL  AMA


000000                                         .PSECT  $CODE$,  RO
000000     122      104      065     L$DESC::.ASCII  /RD5/
000003     061      057      065             .ASCII  /1/<57>/5/
000006     062      040      104             .ASCII  /2 D/
000011     111      123      113             .ASCII  /ISK/
000014     040      106      117             .ASCII  / FO/
000017     122      115      101             .ASCII  /RMA/
000022     124      124      105             .ASCII  /TTE/
000025     122      000      000             .ASCII  /R/<00><00>
000030     122      121      104     L$DVTYP::
                                             .ASCII  /RQD/
000033     130      061      040             .ASCII  /X1 /
000036     104      111      123             .ASCII  /DIS/
000041     113      040      104             .ASCII  /K D/
000044     122      111      126             .ASCII  /RIV/
000047     105      040      123             .ASCII  /E S/
000052     125      102      123             .ASCII  /UBS/
000055     131      123      124             .ASCII  /YST/
000060     105      115      000             .ASCII  /EM/<00>
000063     000                               .ASCII  <00>
000064  000000C                      L$HRDLN::
                                             .WORD   <<<L$NDHRD-L$HRDLN>/2>-1>
000066  000031                       GP$1::  .WORD   31
000070  000000G                              .WORD   HW.Q1.IP
000072  016000                               .WORD   16000
```

```
000074  177777                          .WORD   -1
000076  001031              GP$2::      .WORD   1031
000100  000000G                         .WORD   HW.Q2.VECTOR
000102  000004                          .WORD   4
000104  000774                          .WORD   774
000106  002032              GP$3::      .WORD   2032
000110  000000G                         .WORD   HW.Q3.BR
000112  177777                          .WORD   -1
000114  000000                          .WORD   0
000116  000007                          .WORD   7
000120                      L$NDHRD::
                                        .BLKW   1
000122  000000C             L$SFTLN::
                                        .WORD   <<<L$NDSFT-L$SFTLN>/2>-1>
000124  000032              GP$4::      .WORD   32
000126  000000G                         .WORD   SW.Q1.UNIT
000130  177777                          .WORD   -1
000132  000000                          .WORD   0
000134  000003                          .WORD   3
000136  001032              GP$5::      .WORD   1032
000140  000000G                         .WORD   SW.Q2.MODE
000142  177777                          .WORD   -1
000144  000001                          .WORD   1
000146  000003                          .WORD   3
000150                      L$NDSFT::
                                        .BLKW   1


                                        .GLOBL  HW.Q1.IP, HW.Q2.VECTOR, HW.Q3.BR
                                        .GLOBL  SW.Q1.UNIT, SW.Q2.MODE, FMT2, FMT3
                                        .GLOBL  PWR.MSG, ABO.MSG, TO.MANY.UNITS
                                        .GLOBL  GOOD.NUM.UNITS, LUN, PTBL.PTR
                                        .GLOBL  L$UNIT, RDRX.ADDR, VEC.ADDR, BR.LEVEL
                                        .GLOBL  UNIT.NO, ISD.STRUCT

        000066'                 L$HARD==        L$HRDLN*2
        000124'                 L$SOFT==        L$SFTLN*2


                                        .SBTTL  LRPT REPORT CODING SECTION
000000  000207                  LRPT:   RTS     PC                              ;                       1684

; Routine Size:  1 word,        Routine Base:  $CODE$ + 0152
; Maximum stack depth per invocation:  0 words


                                        .SBTTL  L$RPT REPORT CODING SECTION
000000  004737  000152'         L$RPT:: JSR     PC,LRPT                         ;                       1695
000004  104425                          TRAP    25
000006  000207                          RTS     PC
```

```
; Routine Size:  4 words,      Routine Base:  $CODE$ + 0154
; Maximum stack depth per invocation:  2 words
```

```
;    1706  1     %sbttl 'INITIALIZE SECTION'
;    1707  2     BGNINIT;
;    1708  2
;    1709  2     !++
;    1710  2     ! The  initialization code is executed at the beginning of every
;    1711  2     ! sub-pass  and  is primarily used for requesting P_Tables.  Any
;    1712  2     ! other  set-up type functions may also be performed in the init
;    1713  2     ! code.
;    1714  2     !
;    1715  2     ! The  initialize code is executed under five conditions.  There
;    1716  2     ! are supervisor event flags that are used to let the diagnostic
;    1717  2     ! know under which condition the execution is taking place.  The
;    1718  2     ! event flags are read using the "READEF" macro.
;    1719  2     !
;    1720  2     ! The  conditions  under which the init code is executed and the
;    1721  2     ! corrresponding event flags are:
;    1722  2     !
;    1723  2     !                START COMMAND          EF.START
;    1724  2     !                RESTART COMMAND        EF.RESTART
;    1725  2     !                CONTINUE COMMAND       EF.CONTINUE
;    1726  2     !                POWERDOWN/POWERUP      EF.PWR
;    1727  2     !                NEW PASS               EF.NEW
;    1728  2     !
;    1729  2     !       Example of event flag use:
;    1730  2     !
;    1731  2     !                if READEF(EF.START) then
;    1732  2     !                   START_FLAG = 1;
;    1733  2     !--
;    1734  2
;    1735  2     !+
;    1736  2     ! First  read  the  event flag EF_PWR to see
;    1737  2     ! if  this  init code is being performed due
;    1738  2     ! to  a  system  power  fail.  If it is then
;    1739  2     ! report the  incident  to  the operator and
;    1740  2     ! abort the DM machine and further execution
;    1741  2     ! of this program.
;    1742  2     !-
;    1743  2
;    1744  2     if READEF (EF_PWR)                          !Is the PWR event flag set
;    1745  2     then
;    1746  3         begin                                   !Report the incident and abort
;    1747  3         PRINTF (PWR_MSG);                       !Power fail print message
;    1748  3         PRINTF (ABO_MSG);                       !Aborting program message
;    1749  3         WRT_RDRX (RCIP, RC_ALL, ZEROS);         !Abort DM code execution
;    1750  3         DOCLN;                                  !Abort further program execution
;    1751  2         end;
;    1752  2
;    1753  2     !
;    1754  2     ! See if the DRS commands START or RESTART were used to start
;    1755  2     ! this formatting session.  If either of them were used then
;    1756  2     ! start the formatting session at logical unit zero.
;    1757  2     !
;    1758  2
```

```
;  1759  3     if (READEF (EF_START)) or (READEF (EF_RESTART)) !Did start of restart get us here
;  1760  2     then
;  1761  3         begin
;  1762  3         !
;  1763  3         ! Check the operator for triing to format
;  1764  3         ! more than the defined  limit.   If they
;  1765  3         ! are   then  report  the  error  and die.
;  1766  3         !
;  1767  3
;  1768  3         if .L$UNIT gtru L$LIMIT                     !Is the formatting limit exceeded
;  1769  3         then
;  1770  4             begin
;  1771  4             PRINTF (TO_MANY_UNITS);                 !Report the error
;  1772  4             PRINTF (GOOD_NUM_UNITS);                !Tell him/her the limit
;  1773  4             DOCLN;                                  !Go to cleanup and die
;  1774  3             end;
;  1775  3
;  1776  3         !+
;  1777  3         ! Everything  looks  good  so far so lets
;  1778  3         ! move  on and  get the hardware question
;  1779  3         ! responses and save them.
;  1780  3         !-
;  1781  3
;  1782  3         LUN = -1;                                   !Start formatting at logical unit 0
;  1783  3
;  1784  3         do
;  1785  4             begin
;  1786  4             LUN = .LUN + 1;
;  1787  4
;  1788  4             if .LUN gequ .L$UNIT then DOCLN;
;  1789  4
;  1790  4             end
;  1791  3         until (GPHARD (.LUN, PTBL_PTR)) nequ ZERO;
;  1792  3
;  1793  3         RDRX_ADDR = .PTBL_PTR [wrd0];               !Load up the controllers base addres
;  1794  3         VEC_ADDR = .PTBL_PTR [wrd1];                !Load up the controllers vector address
;  1795  3         BR_LEVEL = .PTBL_PTR [wrd2];                !Load up the controllers bus request
;  1796  3         UNIT_NO = .PTBL_PTR [wrd3];                 !Load up the unit number to format
;  1797  3         !+
;  1798  3         ! Before  leaving  the  init  code section we must
;  1799  3         ! first  do  some  house  keeping left behind from
;  1800  3         ! the ISD_STRUCT  preset declaration.  The adapter
;  1801  3         ! purge  interrupt  bit  must  be  defined for the
;  1802  3         ! type machine  this  formatter  is  running under.
;  1803  3         !-
;  1804  3
;  1805  3         if %bliss (bliss16)                         !Define compiler
;  1806  3         then
;  1807  3             ISD_STRUCT [BLK1, WRD1, S2W_PI] = ZERO  !No purging for PDP-11
;  1808  3         else
;  1809  3             ISD_STRUCT [BLK1, WRD1, S2W_PI] = ONE;  !Purging for VAX-11
;  1810  3
;  1811  3         !+
```

```
;    1812  3          ! Now load in the RDRX formatter DM code from
;    1813  3          ! the local boot device  into the
;    1814  3          ! blank buffer allocated in azkel6.  Report
;    1815  3          ! load error if an error code is returned.
;    1816  3          !
;    1817  3          ! The DM code will only be read in during start
;    1818  3          ! or restarts of the host code.  Block zero of the
;    1819  3          ! the DM .sav file will thrown out.
;    1820  3          !-
;    1821  3
;    1822  3      !     if LOAD_FILE (AZFMTR, DM_FN$EXT, DM_SIZE) then DECODE ();
;    1823  3
;    1824  3          !
;    1825  3          ! Now calculate the overlay sections starting address
;    1826  3          ! and store the result in global location 'ovsa' for
;    1827  3          ! future ref.
;    1828  3          !
;    1829  3          ! This takes the DM buffer starting adrs and adds to it
;    1830  3          ! the number of bytes in the (initial load + remote prog
;    1831  3          ! header size) resulting in the first adrs of the overlay
;    1832  3          ! section.
;    1833  3          !
;    1834  3
;    1835  3      !     OVSA = AZFMTR + (.AZFMTR [WRD0]);          !Calculate overlay start adrs
;    1836  3
;    1837  3          end
;    1838  2      else
;    1839  3          begin
;    1840  3
;    1841  3          if READEF (EF_CONTINUE)
;    1842  3          then
;    1843  4              begin
;    1844  4              !
;    1845  4              ! End this Host code if this .lun is the
;    1846  4              ! the last logical unit to format.
;    1847  4              !
;    1848  4
;    1849  4              if (.LUN + 1) gequ .L$UNIT then DOCLN;
;    1850  4
;    1851  4              PRINTF (FMT3, .LUN, .UNIT_NO);          !Report this luns format was aborted
;    1852  4
;    1853  3              end;
;    1854  3
;    1855  3          if READEF (EF_NEW) then GPHARD (0, 0);              !New pass means all units formatted
;    1856  3
;    1857  3          do
;    1858  4              begin
;    1859  4              LUN = .LUN + 1;
;    1860  4
;    1861  4              if .LUN gequ .L$UNIT then DOCLN;
;    1862  4
;    1863  4              end
;    1864  3          until (GPHARD (.LUN, PTBL_PTR)) nequ ZERO;
```

```
;    1865  3
;    1866  3          RDRX_ADDR = .PTBL_PTR [wrd0];          !Load up the controllers base addres
;    1867  3          VEC_ADDR = .PTBL_PTR [wrd1];           !Load up the controllers vector address
;    1868  3          BR_LEVEL = .PTBL_PTR [wrd2];           !Load up the controllers bus request
;    1869  3          UNIT_NO = .PTBL_PTR [wrd3];            !Load up the unit number to format
;    1870  2          end;
;    1871  2      !
;    1872  1      ENDINIT;


                                        .SBTTL  LINIT INITIALIZE SECTION
000000  005746                 LINIT:  TST     -(SP)                           ;                       1705
000002  012700  000034                 MOV     #34,R0                          ;                       1744
000006  104447                         TRAP    47
000010  103025                         BHIS    1$
000012  012746  000000G                MOV     #PWR.MSG,-(SP)                  ;                       1747
000016  012746  000001                 MOV     #1,-(SP)
000022  010600                         MOV     SP,R0                           ; SP,*
000024  104417                         TRAP    17
000026  012716  000000G                MOV     #ABO.MSG,(SP)                   ;                       1748
000032  012746  000001                 MOV     #1,-(SP)
000036  010600                         MOV     SP,R0                           ; SP,*
000040  104417                         TRAP    17
000042  017766  000000G 000006         MOV     @RDRX.ADDR,6(SP)                ; *,RC$S.REG            1749
000050  005000                         CLR     R0                              ; RC$M.REG
000052  005077  000000G                CLR     @RDRX.ADDR
000056  104444                         TRAP    44
000060  062706  000006                 ADD     #6,SP                           ;                       1746
000064  012700  000040         1$:     MOV     #40,R0                          ;                       1759
000070  104447                         TRAP    47
000072  103404                         BCS     2$
000074  012700  000037                 MOV     #37,R0
000100  104447                         TRAP    47
000102  103065                         BHIS    6$
000104  023727  000000G 000020 2$:     CMP     L$UNIT,#20                      ;                       1768
000112  101417                         BLOS    3$
000114  012746  000000G                MOV     #TO.MANY.UNITS,-(SP)            ;                       1771
000120  012746  000001                 MOV     #1,-(SP)
000124  010600                         MOV     SP,R0                           ; SP,*
000126  104417                         TRAP    17
000130  012716  000000G                MOV     #GOOD.NUM.UNITS,(SP)            ;                       1772
000134  012746  000001                 MOV     #1,-(SP)
000140  010600                         MOV     SP,R0                           ; SP,*
000142  104417                         TRAP    17
000144  104444                         TRAP    44
000146  062706  000006                 ADD     #6,SP                           ;                       1770
000152  112737  000377  000000G 3$:     MOVB    #377,LUN                        ;                       1782
000160  105237  000000G         4$:     INCB    LUN                             ;                       1786
000164  005000                         CLR     R0                              ;                       1788
000166  153700  000000G                BISB    LUN,R0
000172  020037  000000G                CMP     R0,L$UNIT
000176  103401                         BLO     5$
000200  104444                         TRAP    44
```

```
000202  005000                          5$:     CLR     R0                              ;                       1791
000204  153700  000000G                        BISB    LUN,R0
000210  104442                                 TRAP    42
000212  010037  000000G                        MOV     R0,PTBL.PTR
000216  001760                                 BEQ     4$
000220  011037  000000G                        MOV     (R0),RDRX.ADDR                  ;                       1793
000224  016037  000002  000000G               MOV     2(R0),VEC.ADDR                  ;                       1794
000232  016037  000004  000000G               MOV     4(R0),BR.LEVEL                  ;                       1795
000240  016037  000006  000000G               MOV     6(R0),UNIT.NO                   ;                       1796
000246  142737  000001  000006G               BICB    #1,ISD.STRUCT+6                 ;                       1805
000254  000474                                 BR      11$                             ;                       1759
000256  012700  000036                  6$:     MOV     #36,R0                          ;                       1841
000262  104447                                 TRAP    47
000264  103025                                 BHIS    8$
000266  005000                                 CLR     R0                              ;                       1849
000270  153700  000000G                        BISB    LUN,R0
000274  005200                                 INC     R0
000276  020037  000000G                        CMP     R0,L$UNIT
000302  103401                                 BLO     7$
000304  104444                                 TRAP    44
000306  013746  000000G                  7$:     MOV     UNIT.NO,-(SP)                   ;                       1851
000312  005046                                 CLR     -(SP)
000314  113716  000000G                        MOVB    LUN,(SP)
000320  012746  000000G                        MOV     #FMT3,-(SP)
000324  012746  000003                        MOV     #3,-(SP)
000330  010600                                 MOV     SP,R0                           ; SP,*
000332  104417                                 TRAP    17
000334  062706  000010                        ADD     #10,SP                          ;                       1843
000340  012700  000035                  8$:     MOV     #35,R0                          ;                       1855
000344  104447                                 TRAP    47
000346  103004                                 BHIS    9$
000350  005000                                 CLR     R0
000352  104442                                 TRAP    42
000354  010037  000000                        MOV     R0,@#0
000360  105237  000000G                 9$:     INCB    LUN                             ;                       1859
000364  005000                                 CLR     R0                              ;                       1861
000366  153700  000000G                        BISB    LUN,R0
000372  020037  000000G                        CMP     R0,L$UNIT
000376  103401                                 BLO     10$
000400  104444                                 TRAP    44
000402  005000                          10$:    CLR     R0                              ;                       1864
000404  153700  000000G                        BISB    LUN,R0
000410  104442                                 TRAP    42
000412  010037  000000G                        MOV     R0,PTBL.PTR
000416  001760                                 BEQ     9$
000420  011037  000000G                        MOV     (R0),RDRX.ADDR                  ;                       1866
000424  016037  000002  000000G               MOV     2(R0),VEC.ADDR                  ;                       1867
000432  016037  000004  000000G               MOV     4(R0),BR.LEVEL                  ;                       1868
000440  016037  000006  000000G               MOV     6(R0),UNIT.NO                   ;                       1869
000446  005726                          11$:    TST     (SP)+                           ;                       1705
000450  000207                                 RTS     PC

; Routine Size:  149 words,    Routine Base:  $CODE$ + 0164
```

```
; Maximum stack depth per invocation:  7 words


                                        .SBTTL  L$INIT INITIALIZE SECTION
000000  004737  000164'          L$INIT::JSR     PC,LINIT                ;                1870
000004  104411                          TRAP    11
000006  000207                          RTS     PC

; Routine Size:  4 words,       Routine Base:  $CODE$ + 0636
; Maximum stack depth per invocation:  2 words
```

```
;    1873  1     %sbttl 'AUTODROP SECTION'
;    1874  1     !+
;    1875  1     ! This code is executed immediately after the initialize code if
;    1876  1     ! the "ADR" flag was set.  The unit(s) under test are checked to
;    1877  1     ! see  if they  will  respond.  Those that don't are immediately
;    1878  1     ! dropped from testing.
;    1879  1     !-
;    1880  2     BGNAUTO;
;    1881  2     !
;    1882  2     ! Testing  devices  for    existence
;    1883  2     ! is done Via test 1 of this program.
;    1884  2     !
;    1885  2     return;
;    1886  1     ENDAUTO;


                                        .SBTTL  LAUTO AUTODROP SECTION
000000  000207                  LAUTO:  RTS     PC                              ;                               1872

; Routine Size:  1 word,        Routine Base:  $CODE$ + 0646
; Maximum stack depth per invocation:  0 words



                                        .SBTTL  L$AUTO AUTODROP SECTION
000000  004737  000646'          L$AUTO::JSR    PC,LAUTO                        ;                               1885
000004  104461                          TRAP    61
000006  000207                          RTS     PC

; Routine Size:  4 words,       Routine Base:  $CODE$ + 0650
; Maximum stack depth per invocation:  2 words
```

```
;    1887  1     %sbttl 'CLEANUP CODING SECTION'
;    1888  1     !+
;    1889  1     ! Cleanup coding is assembled with the diagnostic program, utilizing in-
;    1890  1     ! itiating  (BGNCLN)  and ending (ENDCLN) directives.  The coding can be
;    1891  1     ! used by either the diagnostic program or the supervisor to affect  the
;    1892  1     ! return of a test device to a static state.
;    1893  1     !
;    1894  1     ! The clean-up code is invoked in three different ways:
;    1895  1     !
;    1896  1     !       A. At end of every sub-pass
;    1897  1     !       B. Issuance of DOCLN macro
;    1898  1     !       C. Operator ^C
;    1899  1     !-
;    1900  2     BGNCLN;
;    1901  2     BRESET;
;    1902  2     return;
;    1903  1     ENDCLN;


                                        .SBTTL  LCLEAN CLEANUP CODING SECTION
000000  104433                  LCLEAN: TRAP    33                      ;                       1900
000002  000207                          RTS     PC                      ;                       1886

; Routine Size:  2 words,      Routine Base:  $CODE$ + 0660
; Maximum stack depth per invocation:  2 words



                                        .SBTTL  L$CLEAN CLEANUP CODING SECTION
000000  004737  000660'          L$CLEAN::
                                        JSR     PC,LCLEAN               ;               1902
000004  104412                          TRAP    12
000006  000207                          RTS     PC

; Routine Size:  4 words,      Routine Base:  $CODE$ + 0664
; Maximum stack depth per invocation:  2 words
```

```
;    1904  1     %sbttl 'DROP UNIT SECTION'
;    1905  1     !*
;    1906  1     ! The drop code is invoked by a DODU macro or a drop command,  and  con-
;    1907  1     ! tains any code that needs to be executed in conjunction with the drop-
;    1908  1     ! ping of a unit from the test cycle.  No coding  is  required  in  this
;    1909  1     ! section.
;    1910  1     !
;    1911  1     ! The effect of a DODU is the same whether executed in the init code  or
;    1912  1     ! in a hardware test.  It invokes the drop unit coding and causes subse-
;    1913  1     ! quent GPHARD'S for that logical unit to be  returned  "NOT  COMPLETE".
;    1914  1     ! This   effect   lasts only  for  the  duration  of the current command.
;    1915  1     !*
;    1916  2     BGNDU;
;    1917  2     return;
;    1918  1     ENDDU;


                                        .SBTTL  LDU DROP UNIT SECTION
000000  000207                  LDU:    RTS     PC                              ;    1903

; Routine Size:  1 word,        Routine Base:  $CODE$ + 0674
; Maximum stack depth per invocation:  0 words



                                        .SBTTL  L$DU DROP UNIT SECTION
000000  004737  000674'          L$DU:: JSR     PC,LDU                          ;    1917
000004  104453                          TRAP    53
000006  000207                         RTS     PC

; Routine Size:  4 words,       Routine Base:  $CODE$ + 0676
; Maximum stack depth per invocation:  2 words
```

```
;    1919  1     %sbttl 'ADD UNIT SECTION'
;    1920  1     !+
;    1921  1     ! The add code is invoked by the ADD command, and contains any code that
;    1922  1     ! needs  to  be  executed  in conjunction with adding a unit back to the
;    1923  1     ! test cycle.  No coding is required in this section.
;    1924  1     !
;    1925  1     ! Units may be added to the test sequence only through the use of opera-
;    1926  1     ! tor  ADD  command.   Each unit must have a P-TABLE in memory due to an
;    1927  1     ! earlier hardware dialogue (i.e.  the  unit  was  previously  dropped).
;    1928  1     ! The  ADD code must be delimited by BGNAU, ENDAU.  There is no particu-
;    1929  1     ! lar coding required in the add code to cause the add to be  effective:
;    1930  1     !
;    1931  1     ! The section is just for programmer housekeeping.
;    1932  1     !-
;    1933  2     BGNAU;
;    1934  2     return;
;    1935  1     ENDAU;


                                        .SBTTL  LAU ADD UNIT SECTION
000000  000207                  LAU:    RTS     PC                              ;    1918

; Routine Size:  1 word,        Routine Base:  $CODE$ + 0706
; Maximum stack depth per invocation:  0 words


                                        .SBTTL  L$AU ADD UNIT SECTION
000000  004737  000706'          L$AU:: JSR     PC,LAU                          ;    1934
000004  104452                          TRAP    52
000006  000207                          RTS     PC

; Routine Size:  4 words,       Routine Base:  $CODE$ + 0710
; Maximum stack depth per invocation:  2 words

;    1936  1     end
;    1937  1
;    1938  0     eludom


;                                       PSECT SUMMARY
;
;       Psect Name                      Words     Attributes
;         $CODE$                          232       RO  ,  I  ,  LCL,  REL,  CON


;                                   Library Statistics
;
```

```
;                                    -------- Symbols --------      Pages      Processing
;       File                         Total    Loaded   Percent     Mapped     Time
;
;  DISK$USER2:[YOUNG.FMT]ZRQBBO.L16;4    358       104        29         18       00:00.1



;                                    COMMAND QUALIFIERS

;       BLISS/PDP11/LIST ZRQB2.B16/EN:NOEIS/SOURCE=PAGE:53

; Size:         179 code + 53 data words
; Run Time:       00:15.4
; Elapsed Time:   02:12.3
; Lines/CPU Min:   7535
; Lexemes/CPU-Min: 47202
; Memory Used:  194 pages
; Compilation Complete
```

```
;   0   MODULE ZRQB3 (%TITLE 'RDRX DISK FORMATTER'
;   0                       IDENT = 'REV C PATCH 0',
;   0                       ADDRESSING_MODE (ABSOLUTE) ,
;   0                       ENVIRONMENT (NOEIS)
;   0                       ) =
;   1   BEGIN
;   1
;   1   !                   MAIN CODE FOR FORMATTER
;   1
;   1   library 'ZRQBB0.L16';                        !Define RDRX formatter library
;   1
;   1   require 'BLSMAC.REQ';                        !Define Bliss Macro require file
;   1
;   1   %sbttl 'MODULE DECLARATIONS'
```

```
;    1550  1      forward routine
;    1551  1              btod ,
;    1552  1              REV8 : novalue ,
;    1553  1              REV9 : novalue;
;    1554  1      !+
;    1555  1      ! The  psect named  "code or $code$" is redefinded here
;    1556  1      ! to be called "ab$code".  This is done to organize the
;    1557  1      ! formatters  test  sections  into  a  seperate  psect.
;    1558  1      !-
;    1559  1      !psect
;    1560  1      !    code = ab$code;
;    1561  1
;    1562  1      !+
;    1563  1      ! Structure declarations used within this module.
;    1564  1      !-
;    1565  1
;    1566  1      structure
;    1567  1
;    1568  1          !+
;    1569  1          ! RDRX  register  accessing  structure. This
;    1570  1          ! structure  allows  RDRX register accessing
;    1571  1          ! to  be transportable between the PDP-11 and
;    1572  1          ! VAX Diagnostic Supervisors.
;    1573  1          !
;    1574  1          ! This  also  defines an access algorithm for
;    1575  1          ! VAX to allow field reference to MBA address
;    1576  1          ! space  without  generating  machine checks.
;    1577  1          !-
;    1578  1
;    1579  1          RDRX [O, P, S, E] =
;    1580  2              begin
;    1581  2
;    1582  2              local
;    1583  2                  RC$S_REG;
;    1584  2
;    1585  2              RC$S_REG = .(RDRX + %upval*0)<0, %bpval, 0>;
;    1586  2              RC$S_REG
;    1587  2              end
;    1588  1              <P, S, E>;
;    1589  1
```

```
;    1590  1      !+
;    1591  1      ! External Routines declared outside this module.
;    1592  1      !-
;    1593  1
;    1594  1      external routine
;    1595  1          setcpu,
;    1596  1          copy,
;    1597  1          ABORT,
;    1598  1          GET_DUST_STATUS,
;    1599  1          EX_LOC_PROG,
;    1600  1          REC_DATA,
;    1601  1          SEND_DATA,
;    1602  1          SET_CNTLR_CHAR,
;    1603  1          DUP$I_SERVICE : INT_LNK$TYP novalue,
;    1604  1          INT$I_SERVICE : INT_LNK$TYP novalue,
;    1605  1          INIT_COM_AREA,
;    1606  1          BOOT_RDRX,
;    1607  1          DECODE : novalue;
;    1608  1
```

```
;    1609  1      !+
;    1610  1      ! External Declaration of  datums
;    1611  1      ! declared outside of this module.
;    1612  1      !-
;    1613  1
;    1614  1      external
;    1615  1          !
;    1616  1          ! Micellanious external data declarations
;    1617  1          !
;    1618  1          REC_ENVELOPE : blockvector [REC_ALLOCATE, RB_SIZE + 2, word] field (ENV_FIELD),
;    1619  1          NXT_CRN : byte,                                  !Next seq command ref number
;    1620  1          RET_EN$AD : ref block [RB_SIZE + 2, word] field (ENV_FIELD),
;    1621  1          SND_BUF : vector [SNDB_SIZE, word],              !DUP send cmd text buffer
;    1622  1          REC_BUF : block [RECB_SIZE, word] field (RECB_FIELD),       !DUP receive  cmd text buffer
;    1623  1          MSGADR,                                          !Pointer to DM sent ascii text
;    1624  1          NSD_SLOT : word,                                 !Stores next send ring slot to load cmd into
;    1625  1          NRD_SLOT : word,                                 !Stores next receive slot to expect response in
;    1626  1          VEC_ADDR : word,                                 !Stores controllers vector address
;    1627  1          RET_STATUS : word,                               !Stores return status of called routines
;    1628  1          PID_SAVE : word,                                 !Saves process indicator word
;    1629  1          RDRX_ADDR : ref RDRX field (ISD_FIELD),          !RDRX reference structure
;    1630  1          PTBL_PTR : ref vector [4, word],                 !table pointer for fetching unit number
;    1631  1          SW_UNIT_NO : word,                               !software table unit number
;    1632  1          SW_MODE : word,                                  !software table mode
;    1633  1          UC_VER : byte,                                   !controller version number
;    1634  1          UNIT_NO : word,                                  !unit number to format
;    1635  1          !
;    1636  1          ! Init sequence code error and informational messages
;    1637  1          !
;    1638  1          BOOT_FAILURE,
;    1639  1          PROTO_VIOLATION,
;    1640  1          PORT_INIT_ERR,
;    1641  1          !
;    1642  1          ! Default values
;    1643  1          !
;    1644  1          DEF_SERIAL,
;    1645  1          DEF_DATE,
;    1646  1          !
;    1647  1          ! Printing format strings
;    1648  1          !
;    1649  1          FMT1,
;    1650  1          FMT2,
;    1651  1          FMT4,
;    1652  1          FMT_ERR      : vector [7],
;    1653  1          DATMSG,
;    1654  1          CRLF,
;    1655  1          !
;    1656  1          !
;    1657  1          !Special questions so the formatter can save code.
;    1658  1          !
;    1659  1          UNIT$_MSG,
;    1660  1          EXIST$_MSG,
;    1661  1          DOWN$_MSG,
```

```
;    1662  1          INACC$_MSG,
;    1663  1          DFLT$_MSG,
;    1664  1          SERIAL$_MSG,
;    1665  1          DATE$_MSG;
;    1666  1
```

```
;    1667  1     %sbttl 'FORMATTER SECTION 1'
;    1668  3     BGNTST;
;    1669  3
;    1670  3     !++
;    1671  3     ! Functional Description :
;    1672  3     ! Implicit Inputs :
;    1673  3     ! Implicit Outputs :
;    1674  3     ! Completion Codes :
;    1675  3     ! Side Effects :
;    1676  3     !--
;    1677  3     LABEL   TLOOP;
;    1678  3     own
;    1679  3         RETRIES,                                    !Store number of retries to perform
;    1680  3             loopc;
;    1681  3     !+
;    1682  3     ! The  next thing to be done is to boot the
;    1683  3     ! RDRX  controller.  We  will  allow a few
;    1684  3     ! retries if not successful after the first
;    1685  3     ! boot  before we concidered the Controller
;    1686  3     ! dead.
;    1687  3     !
;    1688  3     ! But before we do the boot sequence the processors
;    1689  3     ! priority and interrupt vector address must first
;    1690  3     ! be loaded.  During the init sequence the Init
;    1691  3     ! sequence interrupt service routine will just flag
;    1692  3     ! any interrupts and ignore them since interrupts are
;    1693  3     ! disabled during init sequence.  Later the DUP interrupt
;    1694  3     ! service routine will be load and do the DUP communications
;    1695  3     ! protocol.
;    1696  3     !
;    1697  3     ! The following priorities will be assigned:
;    1698  3     !       1. Processor  will  run  at  priority zero.
;    1699  3     !       2. The RDRX runs at priority 5 by default.
;    1700  3     !       3. The  DUP interrupt routine will run at priority 7.
;    1701  3     !-
;    1702  3     SETPRI (PRI00);                                 !Set the processers priority
;    1703  3     CLRVEC (.VEC_ADDR);                             !Clear out the vector before setting
;    1704  3     SETVEC (.VEC_ADDR, INT$I_SERVICE, PRI07);       !Set the interrupt service priority
;    1705  3                                                     !
;    1706  3     !+
;    1707  3     ! Retry  the RDRX booting until the
;    1708  3     ! return is true or the retry limit
;    1709  3     ! is reached.
;    1710  3     !-
;    1711  3     setcpu ();                                      !set up delay constant for timing loops
;    1712  3
;    1713  3     RETRIES = -1;                                   !Reset the retry counter
;    1714  3
;    1715  3     do
;    1716  4         begin
;    1717  4         RET_STATUS = BOOT_RDRX ();                  !Boot the RDRX controller
;    1718  4         RETRIES = .RETRIES + 1;                     !Up the retry count
;    1719  4         end
```

```
;    1720  3      until (.RET_STATUS) or (.RETRIES eqlu ONE);      !Repeat the Boot until done
;    1721  3
;    1722  3      !
;    1723  3      ! Report booting error if the return
;    1724  3      ! status  never  came  back as true.
;    1725  3      !
;    1726  3      if not .RET_STATUS                               !Did the Controller boot
;    1727  3      then
;    1728  4          begin                                        !Report a boot error
;    1729  4          PRINTB (BOOT_FAILURE);
;    1730  4          DORPT;                                       !Report some statistical data
;    1731  4          DOCLN;                                       !Abort the program
;    1732  3          end;
;    1733  3
;    1734  3      !.
;    1735  3      ! Now  that  the RDRX controller is
;    1736  3      ! booted, check to make sure that the
;    1737  3      ! controller has done its part of the
;    1738  3      ! DUP  protocol  by clearing  out the
;    1739  3      ! port communications area.
;    1740  3      !
;    1741  3      ! While we're there set up the communication
;    1742  3      ! area  for the up and coming communications
;    1743  3      ! between the remote and host program.
;    1744  3      !.
;    1745  3
;    1746  3      if INIT_COM_AREA ()                              !Was the com area cleared out as expected
;    1747  3      then
;    1748  4          begin                                        !Com area not inited so error and abort
;    1749  4          PRINTB (PROTO_VIOLATION);
;    1750  4          PRINTB (PORT_INIT_ERR);
;    1751  4          DOCLN;
;    1752  3          end;
;    1753  3
;    1754  3      !
;    1755  3      ! Before writing the go to the IP register
;    1756  3      ! load the DUP interrupt service adrs into
;    1757  3      ! the RDRX vector address for the up and
;    1758  3      ! coming communications between the host
;    1759  3      ! and controller.
;    1760  3      !
;    1761  3      CLRVEC (.VEC_ADDR);                              !Clear out the vector before starting
;    1762  3      SETVEC (.VEC_ADDR, DUP$I_SERVICE, P%I07);        !Set the interrupt service priority
;    1763  3      !.
;    1764  3      ! All   systems  are  go.   Start   the controllers
;    1765  3      ! functional  micro  code  off  and running.  Write
;    1766  3      ! to  the SA Register with the go bit set and start
;    1767  3      ! things rolling.
;    1768  3      !.
;    1769  3      WRT_RDRX (RCSA, RC_ALL, ONE);                    !Let the controller go
;    1770  3                                                       !
```

```
;    1771  3      !%sbttl 'FORMATTER SECTION 2'
;    1772  3      !
;    1773  3
;    1774  3      !++
;    1775  3      ! Functional Description :
;    1776  3      ! Implicit Inputs :
;    1777  3      ! Implicit Outputs :
;    1778  3      ! Completion Codes :
;    1779  3      ! Side Effects :
;    1780  3      !--
;    1781  3
;    1782  3      !+
;    1783  3      ! Set  the controller characteristics.  Default
;    1784  3      ! values  will be taken in all cases except for
;    1785  3      ! the host time out value which will be changed
;    1786  3      ! to 'wait for ever'.
;    1787  3      !-
;    1788  3
;    1789  3      if SET_CNTLR_CHAR () then DECODE ();          !Call decode if not successful
;    1790  3
;    1791  3      !+
;    1792  3      ! Do a unit on line  cmd which will spin up the
;    1793  3      ! device  and  load  the heads.  The unit to be
;    1794  3      ! placed  on line will be the last unit number
;    1795  3      ! received from  the 'gphard' macro in the init
;    1796  3      ! code.
;    1797  3      !-
;    1798  3
```

```
;    1799  3      !%sbttl 'FORMATTER SECTION 3'
;    1800  3
;    1801  3      ! See if the Dup server in the RDRX Controller is in an
;    1802  3      ! idle  state.  To do this first get the dust status and
;    1803  3      ! then look at the flag field bit 3 for a :
;    1804  3      !       0 = idle
;    1805  3      !       1 = active
;    1806  3      !
;    1807  3
;    1808  3      if GET_DUST_STATUS () then DECODE ();           !Call Decode if connection error
;    1809  3
;    1810  3      !
;    1811  3      ! Look in the flag field bit 3 to see if the server is active.
;    1812  3      !
;    1813  3
;    1814  3      if .RET_EN$AD [FLG_B3]                          !If the server is active exit and reboot
;    1815  3      then
;    1816  4          begin
;    1817  4          !******************
;    1818  4          ! Do some stat recording
;    1819  4          !******************
;    1820  4          DOCLN;                                     !Exit this passes execution
;    1821  3          end;
;    1822  3
;    1823  3      !
;    1824  3      ! The  server  is  not  active  so  down  line load the formatter
;    1825  3      ! and start its execution by issuing a "Execute local program".
;    1826  3      ! Call  the  decode  routine  if  a  connection error is detected.
;    1827  3      !
;    1828  3
;    1829  3      if EX_LOC_PROG () then DECODE ();               !Call decode if connection error
;    1830  3
;    1831  3      !
;    1832  3      ! Get the dust status to see if the server is in an active
;    1833  3      ! state.  An  active state is what we want so error if the
;    1834  3      ! server is in an idle state.
;    1835  3      !
;    1836  3      ! If  in the active state then save the progress indicator
;    1837  3      ! in "Pid_save" for future reference.
;    1838  3      !
;    1839  3
;    1840  3      if GET_DUST_STATUS () then DECODE ();           !Call decode if connection error
;    1841  3
;    1842  3      !
;    1843  3      ! Look at the flag field bit 3 to see if the server is active.
;    1844  3      !
;    1845  3
;    1846  4      if not (.RET_EN$AD [FLG_B3])                    !Reboot if server is idle
;    1847  3      then
;    1848  4          begin
;    1849  4          !******************
;    1850  4          ! Do some stat recording
;    1851  4          !******************
```

```
;  1852  4          DOCLN;                                  !Exit this passes execution
;  1853  4          end
;  1854  3      else
;  1855  3          PID_SAVE = .RET_EN$AD [PLO_IND];        !Save the progress indicator
;  1856  3
;  1857  3      !+
;  1858  3      ! The  Dup  server is in the active state running the formatter
;  1859  3      ! program.  This  DO  LOOP  will  loop  on the DUP sub-protocol
;  1860  3      ! doing  the "send  and receive" data commands.  These commands
;  1861  3      ! establish  the communications  between  this host program and
;  1862  3      ! the  remote formatter program running in the RDRX controller.
;  1863  3      !-
;  1864  3      loopc = 0;
;  1865  3
;  1866  3      while .loopc eqlu 0  do
;  1867  4          begin
;  1868  4
;  1869  4          !+
;  1870  4          ! Do a 'Receive_data' command which poll's the remote program
;  1871  4          ! for  a  message.  The  returned  message  can  either be a:
;  1872  4          !
;  1873  4          !   1. Question
;  1874  4          !         Where the ascii text is a prompt for information.
;  1875  4          !
;  1876  4          !   2. Default question
;  1877  4          !         Where  the  default question message is identical
;  1878  4          !         to  the  question message exept that a null (zero
;  1879  4          !         length) send data is taken to be a default answer
;  1880  4          !         to the question.
;  1881  4          !
;  1882  4          !   3. Information
;  1883  4          !         Where the  ascii  text  is an informative message.
;  1884  4          !
;  1885  4          !   4. Termination
;  1886  4          !         Where the ascii text is an normal termination message.
;  1887  4          !
;  1888  4          !   5. Fatal Error
;  1889  4          !         Where  the  ascii  text  is a fatal error message.
;  1890  4          !
;  1891  4          !   6. Special
;  1892  4          !         This  type is used when only a host program could
;  1893  4          !         respond.
;  1894  4          !-
;  1895  4
;  1896  4          CLR_RBUF;                                !Clear out the receive buffer area
;  1897  4          if REC_DATA () then DECODE ();           !Call decode if connection error
;  1898  4
;  1899  4          !
;  1900  4          ! From  the first word in the send/receive data buffer, look
;  1901  4          ! to  see  what type  message the remote program has sent to
;  1902  4          ! the  Host program.  Use this  message type number to index
;  1903  4          ! into the select expression to perform the requested action
;  1904  4          ! by the remote program.
```

```
;    1905  4          !
;    1906  4          CLR_SBUF;                                   !Clear out the send buffer area
;    1907  4              CH$PTR (MSGADR, .REC_ENVELOPE [.NSD_SLOT, BLO_CNT] - 2)<0, 8> = 0;
;    1908  4
;    1909  4              if .UC_VER lequ 8 then REV8 (loopc)     !be compatable with version 8 and 9
;    1910  4                   else REV9 (RETRIES, loopc);
;    1911  4
;    1912  3      end;
;    1913  3      WRT_RDRX (RCIP, RC_ALL, ONES);  ! shut down the controller
;    1914  3      if .loopc eqlu 1 then
;    1915  4           begin
;    1916  4               EXIT_TST;              ! normal termination
;    1917  4           end
;    1918  4      else begin
;    1919  4               DOCLN;                         ! error abort
;    1920  3      end;
;    1921  1      ENDTST;


                                        .TITLE  ZRQB3 RDRX DISK FORMATTER
                                        .IDENT  /REV C /
                                        .ENABL  AMA

000000                                  .PSECT  $OWN$,  D
000000                          RETRIES:.BLKW   1
000002                          LOOPC:  .BLKW   1

                                        .GLOBL  SETCPU, COPY, ABORT, GET.DUST.STATUS
                                        .GLOBL  EX.LOC.PROG, REC.DATA, SEND.DATA
                                        .GLOBL  SET.CNTLR.CHAR, DUP$I.SERVICE
                                        .GLOBL  INT$I.SERVICE, INIT.COM.AREA, BOOT.RDRX
                                        .GLOBL  DECODE, REC.ENVELOPE, NXT.CRN
                                        .GLOBL  RET.EN$AD, SND.BUF, REC.BUF, MSGADR
                                        .GLOBL  NSD.SLOT, NRD.SLOT, VEC.ADDR, RET.STATUS
                                        .GLOBL  PID.SAVE, RDRX.ADDR, PTBL.PTR
                                        .GLOBL  SW.UNIT.NO, SW.MODE, UC.VER, UNIT.NO
                                        .GLOBL  BOOT.FAILURE, PROTO.VIOLATION
                                        .GLOBL  PORT.INIT.ERR, DEF.SERIAL, DEF.DATE
                                        .GLOBL  FMT1, FMT2, FMT4, FMT.ERR, DATMSG
                                        .GLOBL  CRLF, UNIT$.MSG, EXIST$.MSG, DOWN$.MSG
                                        .GLOBL  INACC$.MSG, DFLT$.MSG, SERIAL$.MSG
                                        .GLOBL  DATE$.MSG


                                        .SBTTL  $T1 FORMATTER SECTION 1
000000                                  .PSECT  $CODE$,  RO

000000  010146                  $T1:    MOV     R1,-(SP)                        ;                       1665
000002  162706  000006                  SUB     #6,SP
000006  005000                          CLR     R0                              ;                       1702
000010  104441                          TRAP    41
```

```
000012  013700  000000G                    MOV     VEC.ADDR,R0                     ;                               1703
000016  104436                              TRAP    36
000020  012746  000340                      MOV     #340,-(SP)                      ;                               1704
000024  012746  000000G                     MOV     #INT$I.SERVICE,-(SP)
000030  013746  000000G                     MOV     VEC.ADDR,-(SP)
000034  012746  000003                      MOV     #3,-(SP)
000040  104437                              TRAP    37
000042  004737  000000G                     JSR     PC,SETCPU                       ;                               1711
000046  012737  177777  000000'            MOV     #-1,RETRIES                     ;                               1713
000054  004737  000000G          1$:        JSR     PC,BOOT.RDRX                    ;                               1717
000060  010037  000000G                     MOV     R0,RET.STATUS
000064  005237  000000'                     INC     RETRIES                         ;                               1718
000070  032700  000001                      BIT     #1,R0                           ; #,RET.STATUS                  1720
000074  001004                              BNE     2$
000076  023727  000000'  000001             CMP     RETRIES,#1
000104  001363                              BNE     1$
000106  032737  000001  000000G    2$:       BIT     #1,RET.STATUS                   ;                               1726
000114  001011                              BNE     3$
000116  012716  000000G                     MOV     #BOOT.FAILURE,(SP)              ;                               1729
000122  012746  000001                      MOV     #1,-(SP)
000126  010600                              MOV     SP,R0                           ; SP,*
000130  104414                              TRAP    14
000132  104424                              TRAP    24
000134  104444                              TRAP    44                              ;                               1730
000136  005726                              TST     (SP)+                           ;                               1728
000140  004737  000000G          3$:        JSR     PC,INIT.COM.AREA                ;                               1746
000144  006000                              ROR     R0
000146  103016                              BCC     4$
000150  012716  000000G                     MOV     #PROTO.VIOLATION,(SP)           ;                               1749
000154  012746  000001                      MOV     #1,-(SP)
000160  010600                              MOV     SP,R0                           ; SP,*
000162  104414                              TRAP    14
000164  012716  000000G                     MOV     #PORT.INIT.ERR,(SP)             ;                               1750
000170  012746  000001                      MOV     #1,-(SP)
000174  010600                              MOV     SP,R0                           ; SP,*
000176  104414                              TRAP    14
000200  104444                              TRAP    44
000202  022626                              CMP     (SP)+,(SP)+                     ;                               1748
000204  013700  000000G          4$:        MOV     VEC.ADDR,R0                     ;                               1761
000210  104436                              TRAP    36
000212  012716  000340                      MOV     #340,(SP)                       ;                               1762
000216  012746  000000G                     MOV     #DUP$I.SERVICE,-(SP)
000222  013746  000000G                     MOV     VEC.ADDR,-(SP)
000226  012746  000003                      MOV     #3,-(SP)
000232  104437                              TRAP    37
000234  013700  000000G                     MOV     RDRX.ADDR,R0                    ;                               1769
000240  016066  000002  000016             MOV     2(R0),16(SP)                    ; *,RC$S.REG
000246  012701  000001                      MOV     #1,R1                           ; *,RC$M.REG
000252  010160  000002                      MOV     R1,2(R0)                        ; RC$M.REG,*
000256  004737  000000G                     JSR     PC,SET.CNTLR.CHAR               ;                               1789
000262  006000                              ROR     R0
000264  103002                              BCC     5$
000266  004737  000000G                     JSR     PC,DECODE
```

```
000272  004737  000000G                 5$:     JSR     PC,GET.DUST.STATUS              ;                               1808
000276  006000                                  ROR     R0
000300  103002                                  BCC     6$
000302  004737  000000G                         JSR     PC,DECODE
000306  013700  000000G                 6$:     MOV     RET.EN$AD,R0                    ;                               1814
000312  032760  004000  000022                 BIT     #4000,22(R0)
000320  001401                                  BEQ     7$
000322  104444                                  TRAP    44                              ;                               1816
000324  004737  000000G                 7$:     JSR     PC,EX.LOC.PROG                  ;                               1829
000330  006000                                  ROR     R0
000332  103002                                  BCC     8$
000334  004737  000000G                         JSR     PC,DECODE
000340  004737  000000G                 8$:     JSR     PC,GET.DUST.STATUS              ;                               1840
000344  006000                                  ROR     R0
000346  103002                                  BCC     9$
000350  004737  000000G                         JSR     PC,DECODE
000354  013700  000000G                 9$:     MOV     RET.EN$AD,R0                    ;                               1846
000360  032760  004000  000022                 BIT     #4000,22(R0)
000366  001002                                  BNE     10$
000370  104444                                  TRAP    44                              ;                               1848
000372  000405                                  BR      11$                             ;                               1846
000374  013700  000000G                 10$:    MOV     RET.EN$AD,R0                    ;                               1855
000400  016037  000024  000000G                MOV     24(R0),PID.SAVE
000406  005037  000002'                 11$:    CLR     LOOPC                           ;                               1864
000412  012701  000020                         MOV     #20,R1                          ;                               1867
000416  060601                                  ADD     SP,R1                           ; I,*
000420  005737  000002'                 12$:    TST     LOOPC                           ;                               1866
000424  001065                                  BNE     17$
000426  005066  000020                         CLR     20(SP)                          ; I                             1867
000432  005061  000000G                 13$:    CLR     MSGADR(R1)
000436  005266  000020                         INC     20(SP)                          ; I
000442  026627  000020  000057                 CMP     20(SP),#57                      ; I,*
000450  101770                                  BLOS    13$
000452  004737  000000G                         JSR     PC,REC.DATA                     ;                               1897
000456  006000                                  ROR     R0
000460  103002                                  BCC     14$
000462  004737  000000G                         JSR     PC,DECODE
000466  005000                         14$:    CLR     R0                              ; I
000470  005060  000000G                 15$:    CLR     SND.BUF(R0)                     ; *(I)
000474  062700  000002                         ADD     #2,R0                           ; *,I
000500  020027  000110                         CMP     R0,#110                         ; I,*
000504  101771                                  BLOS    15$
000506  013700  000000G                         MOV     NSD.SLOT,R0                     ;                               1907
000512  000300                                  SWAB    R0
000514  106000                                  RORB    R0
000516  006000                                  ROR     R0
000520  006000                                  ROR     R0
000522  142700  000077                         BICB    #77,R0
000526  016000  000020G                        MOV     REC.ENVELOPE+20(R0),R0
000532  105060  177776G                        CLRB    MSGADR-2(R0)
000536  123727  000000G 000010                 CMPB    UC.VER,#10                      ;                               1909
000544  101005                                  BHI     16$
000546  012716  000002'                         MOV     #LOOPC,(SP)
```

```
000552  004737  000000V                        JSR     PC,REV8
000556  000720                                  BR      12$
000560  012716  000000'              16$:      MOV     #RETRIES,(SP)                   ;                       1910
000564  012746  000002'                        MOV     #LOOPC,-(SP)
000570  004737  000000V                        JSR     PC,REV9
000574  005726                                  TST     (SP)+
000576  000710                                  BR      12$                             ;                       1866
000600  017766  000000G 000022      17$:      MOV     @RDRX.ADDR,22(SP)               ; *,RC$S.REG            1913
000606  012700  177777                        MOV     #-1,R0                          ; *,RC$M.REG
000612  010077  000000G                        MOV     R0,@RDRX.ADDR                   ; RC$M.REG,*
000616  023727  000002' 000001               CMP     LOOPC,#1                        ;                       1914
000624  001002                                  BNE     18$
000626  104463                                  TRAP    63                              ;                       1915
000630  000401                                  BR      19$
000632  104444              18$:      TRAP    44                              ;                       1918
000634  062706  000024              19$:      ADD     #24,SP                          ;                       1665
000640  012601                                  MOV     (SP)+,R1
000642  000207                                  RTS     PC

; Routine Size:  210 words,     Routine Base:  $CODE$ + 0000
; Maximum stack depth per invocation:  13 words



                                        .SBTTL  T1 FORMATTER SECTION 1
000000  004737  000000'              T1::
000000                               1$:       JSR     PC,$T1                          ;                       1920
000004  104466                                  TRAP    66
000006  006000                                  ROR     R0
000010  103773                                  BLO     1$
000012  000207                                  RTS     PC

; Routine Size:  6 words,       Routine Base:  $CODE$ + 0644
; Maximum stack depth per invocation:  2 words


;   1922  1
;   1923  1     routine REV8 (endflg) : novalue =
;   1924  2     begin
;   1925  2         selectoneu .REC_BUF [MSG_TYP] of            !Select the appropriate action
;   1926  2             set
;   1927  2
;   1928  2             [1] :                                   !Question message type
;   1929  3                 begin
;   1930  3                 !
;   1931  3                 ! Look  into  the  send/receive data buffer  at the message
;   1932  3                 ! number field and  see  what  question the remote  program
;   1933  3                 ! is   asking.  Use the  fields  value  to  index  into the
;   1934  3                 ! select  expression  to  perform  the  appropriate  action.
;   1935  3                 !
;   1936  3
;   1937  3                 selectoneu .REC_BUF [MSG_NUM] of    !Select the requested question
```

```
;    1938  3                    set
;    1939  3
;    1940  3                    [0] :                           !Enter unit number to format
;    1941  4                        begin
;    1942  5                        if (MANUAL) then        GMANID (UNIT$_MSG, SND_BUF, D, %o'3', 0, 3, NO)
;    1943  4                        else SND_BUF = .SW_UNIT_NO;
;    1944  4                        if SEND_DATA (SNDB_COUNT) then DECODE ();
;    1945  3                        end;
;    1946  3
;    1947  3                    [1] :                           !Choose one of the responses
;    1948  4                        begin
;    1949  4                            if (MANUAL)  then
;    1950  5                            begin
;    1951  5                                SND_BUF = 0;
;    1952  5                                GMANIL (EXIST$_MSG, SND_BUF, %o'1', YES, 1);
;    1953  6                                if ( .SND_BUF<0, 1, 0> NEQU 1)
;    1954  6                                    then    begin
;    1955  6                                            GMANIL (DOWN$_MSG, SND_BUF, %o'2', YES, 1);
;    1956  7                                            if ( .SND_BUF<1, 1, 0> NEQU 1)
;    1957  7                                            then    begin
;    1958  7                                                    GMANIL (INACC$_MSG, SND_BUF, %o'4', YES, 1);
;    1959  8                                                    if ( .SND_BUF<2, 1, 0> NEQU 1)
;    1960  8                                                    then    begin
;    1961  8                                                            SND_BUF<0, 1, 0> = 1;
;    1962  8                                                            PRINTB(DFLT$_MSG);
;    1963  7                                                            end;
;    1964  6                                                    end;
;    1965  5                                            end;
;    1966  5                            end
;    1967  4                            else    SND_BUF = .SW_MODE;
;    1968  4
;    1969  4                        if SEND_DATA (SNDB_COUNT) then DECODE ();
;    1970  3                        end;
;    1971  3
;    1972  3                    [2] :                           !Enter an eight character non-zero ascii serial number
;    1973  4                        begin                       !(Any eight characters will do)
;    1974  5                            if (MANUAL) then GMANID (SERIAL$_MSG, SND_BUF, A, %o'177777', 8, 8, NO)
;    1975  4                            else    copy (DEF_SERIAL, SND_BUF, 8);
;    1976  4                        if SEND_DATA (SNDB_COUNT) then DECODE ();
;    1977  3                        end;
;    1978  3
;    1979  3                    [3] :                           !Enter current date MM-DD-YY <- exact format required
;    1980  4                        begin                       !(Actually, any six characters will do)
;    1981  5                        if (MANUAL) then GMANID (DATE$_MSG, SND_BUF, A, %o'177777', 8, 8, NO)
;    1982  4                        else    copy (DEF_DATE, SND_BUF, 8);
;    1983  4                        if SEND_DATA (SNDB_COUNT) then DECODE ();
;    1984  3                        end;
;    1985  3
;    1986  3                    [otherwise] :                   !This message number is unknown
;    1987  4                        begin
;    1988  4                        RET_STATUS = UMN_CODE;      !Unknown message number error code
;    1989  4                        DECODE ();                  !Report error and die
;    1990  3                        end;
```

```
;  1991  3                    tes;
;  1992  3
;  1993  2                end;
;  1994  2
;  1995  2            [2] :                                   !Default Question
;  1996  3                begin
;  1997  3
;  1998  3                selectoneu .REC_BUF [MSG_NUM] of
;  1999  3                    set
;  2000  3                    [otherwise] :                   !This message number is unknown
;  2001  4                        begin
;  2002  4                        RET_STATUS = UMN_CODE;      !Unknown message number error code
;  2003  4                        DECODE ();                  !Report error and die
;  2004  3                        end;
;  2005  3                    tes;
;  2006  3
;  2007  2                end;
;  2008  2
;  2009  2            [3] :                                   !Informational message
;  2010  3                begin
;  2011  3
;  2012  3                selectoneu .REC_BUF [MSG_NUM] of
;  2013  3                    set
;  2014  3
;  2015  3                    [0, 9] :                        !Format begun plus my debug msg
;  2016  4                        begin
;  2017  4                        PRINTB (FMT1, MSGADR);
;  2018  3                        end;
;  2019  3
;  2020  3                    [otherwise] :                   !This message number is unknown
;  2021  4                        begin
;  2022  4                        RET_STATUS = UMN_CODE;      !Unknown message number error code
;  2023  4                        DECODE ();                  !Report error and die
;  2024  3                        end;
;  2025  3                    tes;
;  2026  3
;  2027  2                end;
;  2028  2
;  2029  2            [4] :                                   !Termination message
;  2030  3                begin
;  2031  3                    .endflg = 1;
;  2032  3                        PRINTB (FMT4, .REC_BUF[MSG_NUM]);
;  2033  2                end;
;  2034  2
;  2035  2            [5] :                                   !Fatal error message
;  2036  3                begin
;  2037  3
;  2038  3                        PRINTB (.FMT_ERR [.REC_BUF[MSG_NUM] - 1]);
;  2039  3                    .endflg = 2;
;  2040  2                end;
;  2041  2
;  2042  2            [6] :                                   !Special message type
;  2043  3                begin
```

```
;   2044  3
;   2045  3                 selectu .REC_BUF [MSG_NUM] of
;   2046  3                     set
;   2047  3                     [always] :
;   2048  4                         begin
;   2049  4                         RET_STATUS = UMN_CODE;
;   2050  4                         DECODE ();
;   2051  3                         end;
;   2052  3                     tes;
;   2053  3
;   2054  2                 end;
;   2055  2
;   2056  2             [otherwise] :                       !This message number is unknown
;   2057  3                 begin
;   2058  3                 RET_STATUS = UMT_CODE;          !Unknown message number error code
;   2059  3                 DECODE ();                      !Report error and die
;   2060  2                 end;
;   2061  2             tes;
;   2062  1         end;


                                        .SBTTL  REV8 FORMATTER SECTION 1
000000  004137  000000G         REV8:   JSR     R1,$SAVE2                       ;                       1923
000004  013701  000000G                 MOV     REC.BUF,R1                      ;                       1925
000010  006201                          ASR     R1
000012  006201                          ASR     R1
000014  006201                          ASR     R1
000016  006201                          ASR     R1
000020  000301                          SWAB    R1
000022  042701  177760                  BIC     #177760,R1
000026  020127  000001                  CMP     R1,#1                           ;                       1928
000032  001171                          BNE     10$
000034  013702  000000G                 MOV     REC.BUF,R2                      ;                       1937
000040  042702  170000                  BIC     #170000,R2
000044  001017                          BNE     2$                              ;                       1940
000046  104450                          TRAP    50                              ;                       1942
000050  103011                          BHIS    1$
000052  104443                          TRAP    43
000054  000406                          .WORD   406
000056  000000G                         .WORD   SND.BUF
000060  000042                          .WORD   42
000062  000000G                         .WORD   UNIT$.MSG
000064  000003                          .WORD   3
000066  000000                          .WORD   0
000070  000003                          .WORD   3
000072  000467                          BR      4$
000074  013737  000000G 000000G   1$:    MOV     SW.UNIT.NO,SND.BUF              ;                       1943
000102  000463                          BR      4$                              ;                       1944
000104  020227  000001          2$:     CMP     R2,#1                           ;                       1947
000110  001070                          BNE     5$
000112  104450                          TRAP    50                              ;                       1949
000114  103053                          BHIS    3$
000116  005037  000000G                 CLR     SND.BUF                         ;                       1951
```

```
000122  104443                          TRAP    43                              ;                                       1952
000124  000404                          .WORD   404
000126  000000G                         .WORD   SND.BUF
000130  000130                          .WORD   130
000132  000000G                         .WORD   EXIST$.MSG
000134  000001                          .WORD   1
000136  032737  000001  000000G         BIT     #1,SND.BUF                      ;                                       1953
000144  001042                          BNE     4$
000146  104443                          TRAP    43                              ;                                       1955
000150  000404                          .WORD   404
000152  000000G                         .WORD   SND.BUF
000154  000130                          .WORD   130
000156  000000G                         .WORD   DOWN$.MSG
000160  000002                          .WORD   2
000162  032737  000002  000000G         BIT     #2,SND.BUF                      ;                                       1956
000170  001030                          BNE     4$
000172  104443                          TRAP    43                              ;                                       1958
000174  000404                          .WORD   404
000176  000000G                         .WORD   SND.BUF
000200  000130                          .WORD   130
000202  000000G                         .WORD   INACC$.MSG
000204  000004                          .WORD   4
000206  032737  000004  000000G         BIT     #4,SND.BUF                      ;                                       1959
000214  001016                          BNE     4$
000216  152737  000001  000000G         BISB    #1,SND.BUF                      ;                                       1961
000224  012746  000000G                 MOV     #DFLT$.MSG,-(SP)                ;                                       1962
000230  012746  000001                  MOV     #1,-(SP)
000234  010600                          MOV     SP,R0                           ; SP,*
000236  104414                          TRAP    14
000240  022626                          CMP     (SP)+,(SP)+                     ;                                       1960
000242  000403                          BR      4$                              ;                                       1949
000244  013737  000000G 000000G  3$:     MOV     SW.MODE,SND.BUF                 ;                                       1967
000252  012746  000020          4$:     MOV     #20,-(SP)                       ;                                       1969
000256  004737  000000G                 JSR     PC,SEND.DATA
000262  005726                          TST     (SP)+
000264  006000                          ROR     R0
000266  103164                          BCC     19$
000270  000561                          BR      18$
000272  020227  000002          5$:     CMP     R2,#2                           ;                                       1972
000276  001016                          BNE     7$
000300  104450                          TRAP    50                              ;                                       1974
000302  103011                          BHIS    6$
000304  104443                          TRAP    43
000306  000406                          .WORD   406
000310  000000G                         .WORD   SND.BUF
000312  000142                          .WORD   142
000314  000000G                         .WORD   SERIAL$.MSG
000316  177777                          .WORD   -1
000320  000010                          .WORD   10
000322  000010                          .WORD   10
000324  000752                          BR      4$
000326  012746  000000G          6$:     MOV     #DEF.SERIAL,-(SP)               ;                                       1975
000332  000420                          BR      9$
```

```
000334  020227  000003          7$:   CMP     R2,#3                           ;                                       1979
000340  001126                        BNE     16$
000342  104450                        TRAP    50                              ;                                       1981
000344  103011                        BHIS    8$
000346  104443                        TRAP    43
000350  000406                        .WORD   406
000352  000000G                       .WORD   SND.BUF
000354  000142                        .WORD   142
000356  000000G                       .WORD   DATE$.MSG
000360  177777                        .WORD   -1
000362  000010                        .WORD   10
000364  000010                        .WORD   10
000366  000731                        BR      4$
000370  012746  000000G         8$:   MOV     #DEF.DATE,-(SP)                 ;                                       1982
000374  012746  000000G         9$:   MOV     #SND.BUF,-(SP)
000400  012746  000010                MOV     #10,-(SP)
000404  004737  000000G               JSR     PC,COPY
000410  062706  000006                ADD     #6,SP
000414  000716                        BR      4$                              ;                                       1983
000416  020127  000002        10$:    CMP     R1,#2                           ;                                       1995
000422  001475                        BEQ     16$                             ;                                       2002
000424  020127  000003                CMP     R1,#3                           ;                                       2009
000430  001021                        BNE     12$
000432  013700  000000G               MOV     REC.BUF,R0                      ;                                       2012
000436  042700  170000                BIC     #170000,R0
000442  001403                        BEQ     11$                             ;                                       2015
000444  020027  000011                CMP     R0,#11
000450  001062                        BNE     16$
000452  012746  000000G         11$:  MOV     #MSGADR,-(SP)                   ;                                       2017
000456  012746  000000G               MOV     #FMT1,-(SP)
000462  012746  000002                MOV     #2,-(SP)
000466  010600                        MOV     SP,R0                           ; SP,*
000470  104414                        TRAP    14
000472  000420                        BR      13$                             ;                                       2016
000474  020127  000004        12$:    CMP     R1,#4                           ;                                       2029
000500  001020                        BNE     14$
000502  012776  000001  000010        MOV     #1,@10(SP)                      ; *,ENDFLG                              2031
000510  013746  000000G               MOV     REC.BUF,-(SP)                   ;                                       2032
000514  042716  170000                BIC     #170000,(SP)
000520  012746  000000G               MOV     #FMT4,-(SP)
000524  012746  000002                MOV     #2,-(SP)
000530  010600                        MOV     SP,R0                           ; SP,*
000532  104414                        TRAP    14
000534  062706  000006        13$:    ADD     #6,SP                           ;                                       2030
000540  000207                        RTS     PC                              ;                                       1925
000542  020127  000005        14$:    CMP     R1,#5                           ;                                       2035
000546  001020                        BNE     15$
000550  013700  000000G               MOV     REC.BUF,R0                      ;                                       2038
000554  042700  170000                BIC     #170000,R0
000560  006300                        ASL     R0
000562  016046  177776G               MOV     FMT.ERR-2(R0),-(SP)
000566  012746  000001                MOV     #1,-(SP)
000572  010600                        MOV     SP,R0                           ; SP,*
```

```
000574  104414                                  TRAP    14
000576  012776  000002  000014                  MOV     #2,B14(SP)                      ; #,ENDFLG                      2039
000604  022626                                  CMP     (SP)+,(SP)+                     ;                               2036
000606  000207                                  RTS     PC                              ;                               1925
000610  020127  000006                  15$:    CMP     R1,#6                           ;                               2042
000614  001004                                  BNE     17$
000616  012737  004001  000000G         16$:    MOV     #4001,RET.STATUS                ;                               2049
000624  000403                                  BR      18$                             ;                               2050
000626  012737  001001  000000G         17$:    MOV     #1001,RET.STATUS                ;                               2058
000634  004737  000000G                 18$:    JSR     PC,DECODE                       ;                               2059
000640  000207                          19$:    RTS     PC                              ;                               1923

; Routine Size:  209 words,    Routine Base:  $CODE$ + 0660
; Maximum stack depth per invocation:  8 words


;    2063  1
;    2064  1     routine REV9 (retrys, endflg) : novalue =
;    2065  2     begin
;    2066  2     local
;    2067  2         cptr,
;    2068  2         cnum;
;    2069  2         selectoneu .REC_BUF [MSG_TYP] of            !Select the appropriate action
;    2070  2             set
;    2071  2
;    2072  2             [1] :                                   !Question message type
;    2073  3                 begin
;    2074  3                 !
;    2075  3                 ! Look  into  the  send/receive data buffer  at the message
;    2076  3                 ! number field and  see  what  question the remote  program
;    2077  3                 ! is   asking.  Use the  fields  value  to  index  into the
;    2078  3                 ! select  expression  to  perform  the  appropriate  action.
;    2079  3                 !
;    2080  3
;    2081  3                 selectoneu .REC_BUF [MSG_NUM] of    !Select the requested question
;    2082  3                     set
;    2083  3
;    2084  3                     [0] :                           !Enter current date <MM-DD-YYYY>
;    2085  4                         begin
;    2086  4                         !
;    2087  4                         ! If in un-attended reformat mode then give to the DM the
;    2088  4                         ! date received from the init code else let the DM prompt
;    2089  4                         ! the operator for the date.
;    2090  4                         !
;    2091  4
;    2092  5                         if (MANUAL)             !Is the host running in unattended mode
;    2093  4                         then
;    2094  5                             begin
;    2095  5                             GMANID (MSGADR, SND_BUF, A, %o'177777', 8, 10, NO);
;    2096  5                             end
;    2097  4                         else
;    2098  5                             begin
;    2099  5
```

```
;  2100  5                          if ..retrys                !Did DM not verify the date the last time
;  2101  5                          then
;  2102  6                              begin
;  2103  6                              !
;  2104  6                              ! Clear out the date text buffer before loading in
;  2105  6                              ! date and ask the operator for the date again.
;  2106  6                              !
;  2107  6
;  2108  6                              incr i from 0 to 11 do
;  2109  6                                  SND_BUF [.i] = ZEROS;
;  2110  6
;  2111  6                              GMANID (DATMSG, SND_BUF, A, %o'177777', 8, 10, NO); !Get date from operator
;  2112  5                              end;
;  2113  5
;  2114  5                          .retrys = ONE;             !Set the retry flag
;  2115  4                          end;
;  2116  4
;  2117  4     !                PRINTB (CRLF);
;  2118  4
;  2119  4                      if SEND_DATA (SNDB_COUNT) then DECODE ();
;  2120  4
;  2121  3                      end;
;  2122  3
;  2123  3                  [2] :                   ! get cylinder number
;  2124  4                  begin
;  2125  4                      GMANID (MSGADR, SND_BUF, D, %o'177777', 0, 999, NO);
;  2126  4                      btod(.SND_BUF, SND_BUF);
;  2127  4                  if SEND_DATA (SNDB_COUNT) then DECODE ();
;  2128  3                  end;
;  2129  3
;  2130  3                  [3] :
;  2131  4                  begin
;  2132  4                      GMANID (MSGADR, SND_BUF, D, %o'177777', 0, 11700, NO);
;  2133  4                      btod(.SND_BUF, SND_BUF);
;  2134  4                  if SEND_DATA (SNDB_COUNT) then DECODE ();
;  2135  3                  end;
;  2136  3
;  2137  3                  [7] :                           !Enter a non-zero serial number
;  2138  4                      begin
;  2139  4                      cnum = SNDB_COUNT;
;  2140  5                      if (MANUAL)
;  2141  5                      then begin
;  2142  5                          GMANID (MSGADR, SND_BUF, A, %o'177777', 1, 64, NO);
;  2143  5     !                    PRINTB (CRLF);
;  2144  5                          cptr = CH$PTR (SND_BUF);
;  2145  5                          cnum = 0;                       !start at 2 to include message type and number
;  2146  5                          while .CH$PTR(.cptr)<0, 8> nequ 0
;  2147  6                          do begin
;  2148  6                              cnum = .cnum + 1;
;  2149  6                              cptr = CH$PLUS (.cptr, 1);
;  2150  5                          end;
;  2151  5                          if .cnum AND 1 neq 0
;  2152  6                          then begin
```

```
;  2153  6                                        CH$PTR (.cptr)<0, 8> = %C'.';
;  2154  6                                        cnum = .cnum + 1;
;  2155  5                                    end;
;  2156  5                                end
;  2157  4                                else    copy (DEF_SERIAL, SND_BUF, 8);
;  2158  4                                if SEND_DATA (.cnum) then DECODE ();
;  2159  4
;  2160  3                            end;
;  2161  3
;  2162  3                        [otherwise] :                   !This message number is unknown
;  2163  4                            begin
;  2164  4                            RET_STATUS = UMN_CODE;      !Unknown message number error code
;  2165  4                            DECODE ();                  !Report error and die
;  2166  3                            end;
;  2167  3                        tes;
;  2168  3
;  2169  2                    end;
;  2170  2
;  2171  2                [2] :                                   !Default Question
;  2172  3                    begin
;  2173  3
;  2174  3                    selectoneu .REC_BUF [MSG_NUM] of
;  2175  3                        set
;  2176  3
;  2177  3                        [1] :                           !Enter unit number to format <0>
;  2178  4                            begin
;  2179  4                            !
;  2180  4                            ! If in un-attended reformat mode then give to the DM the
;  2181  4                            ! unit number obtained from the hardware P_Table else let
;  2182  4                            ! the DM prompt the operator for the unit to format.
;  2183  4                            !
;  2184  4
;  2185  5                            if (MANUAL)             !Is the host running in unattended mode
;  2186  4                            then
;  2187  5                                begin
;  2188  5                                GMANID (MSGADR, SND_BUF, D, %o'177777', 0, 15, YES);
;  2189  5                                end
;  2190  4                            else
;  2191  5                                begin
;  2192  5                                SND_BUF = .UNIT_NO;
;  2193  5                                PRINTB (FMT2, .UNIT_NO);        !Report which unit is being formatted
;  2194  4                                end;
;  2195  4
;  2196  4    !                       PRINTB (CRLF);
;  2197  4                                if .SND_BUF gequ 10 then        SND_BUF = .SND_BUF - 10 + %ascii'10'
;  2198  4                                else SND_BUF = .SND_BUF + %c'0';
;  2199  4
;  2200  4                            if SEND_DATA (SNDB_COUNT) then DECODE ();
;  2201  4
;  2202  3                            end;
;  2203  3
;  2204  3                        [2] :   !get head number (CR to signal end)
;  2205  4                        begin
```

```
;    2206  4                          SND_BUF = 10;                       ! default says finished
;    2207  4                          cnum = 0;                           !set for default answer
;    2208  4                          GMANID (MSGADR, SND_BUF, D, %o'177777', 0, 10, YES);
;    2209  5                          if .SND_BUF lequ 9 then begin       !oops - not default
;    2210  5                                btod(.SND_BUF, SND_BUF);
;    2211  5                                cnum = 2;
;    2212  4                          end;
;    2213  4                      if SEND_DATA (.cnum) then DECODE ();
;    2214  3                  end;
;    2215  3
;    2216  3                  [4] :                               !Use existing bad block information <N>
;    2217  4                      begin
;    2218  4                      !
;    2219  4                      ! If in un-attended reformat mode then tell the DM to
;    2220  4                      ! do a reformat mode else let the DM prompt the operator
;    2221  4                      ! for the formatting mode to use for this unit.
;    2222  4                      !
;    2223  4
;    2224  5                      if (MANUAL)             !Is the host running in unattended mode
;    2225  4                      then
;    2226  5                          begin
;    2227  5                          GMANIL (MSGADR, SND_BUF, %o'1', YES, 1);
;    2228  5                          SND_BUF = .SND_BUF + %c'Y' - 1; !host version only cares Y or non-Y, so send N as X
;    2229  5                          end
;    2230  4                      else
;    2231  5                          begin
;    2232  5                          SND_BUF = %c'Y';            !Default to do reformat mode
;    2233  4                          end;
;    2234  4
;    2235  4      !                   PRINTB (CRLF);
;    2236  4
;    2237  4                      if SEND_DATA (SNDB_COUNT) then DECODE ();
;    2238  4
;    2239  3                      end;
;    2240  3
;    2241  3                  [5] :                               !Use down-line load <N>
;    2242  4                      begin
;    2243  4
;    2244  5                      if (MANUAL)             !Is the host running in unattended mode
;    2245  4                      then
;    2246  5                          begin
;    2247  5                          GMANIL (MSGADR, SND_BUF, %o'1', YES, 1);
;    2248  5                          SND_BUF = .SND_BUF + %c'Y' - 1;
;    2249  5                          end
;    2250  4                      else
;    2251  5                          begin
;    2252  5                          SND_BUF = %c'N';            !Default not to continue
;    2253  4                          end;
;    2254  4
;    2255  4                      if SEND_DATA (SNDB_COUNT) then DECODE ();
;    2256  4
;    2257  3                      end;
;    2258  3
```

```
;  2259  3                    [6] :                           !Continue if bad block information is inaccessable <N>
;  2260  4                        begin
;  2261  4
;  2262  5                        if (MANUAL)          !Is the host running in unattended mode
;  2263  4                        then
;  2264  5                            begin
;  2265  5                            GMANIL (MSGADR, SND_BUF, %o'1', YES, 1);
;  2266  5                            SND_BUF = .SND_BUF + %c'Y' - 1;
;  2267  5                            end
;  2268  4                        else
;  2269  5                            begin
;  2270  5                            SND_BUF = %c'N';        !Default not to continue
;  2271  4                            end;
;  2272  4
;  2273  4     !                  PRINTB (CRLF);
;  2274  4
;  2275  4                        if SEND_DATA (SNDB_COUNT) then DECODE ();
;  2276  4
;  2277  3                        end;
;  2278  3
;  2279  3                    [otherwise] :                   !This message number is unknown
;  2280  4                        begin
;  2281  4                        GMANID (MSGADR, SND_BUF, A, %o'177777', 1, 64, NO);
;  2282  4                        if SEND_DATA (SNDB_COUNT) then DECODE ();
;  2283  3                        end;
;  2284  3                    tes;
;  2285  3
;  2286  2                end;
;  2287  2
;  2288  2            [3] :                               !Informational message
;  2289  3                begin
;  2290  3
;  2291  3                    PRINTB (CRLF);
;  2292  3                    PRINTB (FMT1, MSGADR);
;  2293  3
;  2294  2                end;
;  2295  2
;  2296  2            [4] :                               !Termination message
;  2297  3                begin
;  2298  3
;  2299  3                    PRINTB (CRLF);
;  2300  3                    PRINTB (FMT1, MSGADR);
;  2301  3                .endflg = 1;                    !mark as normal termination
;  2302  3
;  2303  2                end;
;  2304  2
;  2305  2            [5] :                               !Fatal error message
;  2306  3                begin
;  2307  3
;  2308  3                    PRINTB (CRLF);
;  2309  3                    PRINTB (FMT1, MSGADR);
;  2310  3                .endflg = 2;                    ! mark as error
;  2311  3
```

```
;   2312  2                 end;
;   2313  2
;   2314  2             [6] :                                   !Special message type
;   2315  3                 begin
;   2316  3                         RET_STATUS = UMN_CODE;
;   2317  3                         DECODE ();
;   2318  3
;   2319  2                 end;
;   2320  2
;   2321  2             [otherwise] :                           !This message number is unknown
;   2322  3                 begin
;   2323  3                 RET_STATUS = UMT_CODE;              !Unknown message number error code
;   2324  3                 DECODE ();                          !Report error and die
;   2325  2                 end;
;   2326  2             tes;
;   2327  1     end;


                                        .SBTTL  REV9 FORMATTER SECTION 1
000000  004137  000000G         REV9:   JSR     R1,$SAVE2                       ;                       2064
000004  013701  000000G                 MOV     REC.BUF,R1                      ;                       2069
000010  006201                          ASR     R1
000012  006201                          ASR     R1
000014  006201                          ASR     R1
000016  006201                          ASR     R1
000020  000301                          SWAB    R1
000022  042701  177760                  BIC     #177760,R1
000026  020127  000001                  CMP     R1,#1                           ;                       2072
000032  001402                          BEQ     1$
000034  000137  002204'                 JMP     16$
000040  013702  000000G         1$:     MOV     REC.BUF,R2                      ;                       2081
000044  042702  170000                  BIC     #170000,R2
000050  001044                          BNE     6$                              ;                       2084
000052  104450                          TRAP    50                              ;                       2092
000054  103011                          BHIS    2$
000056  104443                          TRAP    43                              ;                       2095
000060  000406                          .WORD   406
000062  000000G                         .WORD   SND.BUF
000064  000142                          .WORD   142
000066  000000G                         .WORD   MSGADR
000070  177777                          .WORD   -1
000072  000010                          .WORD   10
000074  000012                          .WORD   12
000076  000427                          BR      5$                              ;
000100  032776  000001  000012  2$:     BIT     #1,@12(SP)                      ; *,RETRYS              2100
000106  001420                          BEQ     4$
000110  005000                          CLR     R0                              ; I                     2108
000112  005060  000000G         3$:     CLR     SND.BUF(R0)                     ; *(I)                  2109
000116  062700  000002                  ADD     #2,R0                           ; *,I                   2108
000122  020027  000026                  CMP     R0,#26                          ; I,*
000126  003771                          BLE     3$
000130  104443                          TRAP    43                              ;                       2111
000132  000406                          .WORD   406
```

```
000134  000000G                         .WORD   SND.BUF
000136  000142                         .WORD   142
000140  000000G                         .WORD   DATMSG
000142  177777                         .WORD   -1
000144  000010                         .WORD   10
000146  000012                         .WORD   12
000150  012776  000001  000012  4$:    MOV     #1,@12(SP)              ; *,RETRYS                      2114
000156  000137  002614'          5$:    JMP     29$                     ;                               2119
000162  020227  000002          6$:    CMP     R2,#2                   ;                               2123
000166  001025                         BNE     7$
000170  104443                         TRAP    43                      ;                               2125
000172  000406                         .WORD   406
000174  000000G                         .WORD   SND.BUF
000176  000042                         .WORD   42
000200  000000G                         .WORD   MSGADR
000202  177777                         .WORD   -1
000204  000000                         .WORD   0
000206  001747                         .WORD   1747
000210  013746  000000G                 MOV     SND.BUF,-(SP)           ;                               2126
000214  012746  000000G                 MOV     #SND.BUF,-(SP)
000220  004737  000000V                 JSR     PC,BTOD
000224  012716  000020                  MOV     #20,(SP)                ;                               2127
000230  004737  000000G                 JSR     PC,SEND.DATA
000234  006000                         ROR     R0
000236  103430                         BLO     8$
000240  000431                         BR      9$                      ;                               2124
000242  020227  000003          7$:    CMP     R2,#3                   ;                               2130
000246  001030                         BNE     10$
000250  104443                         TRAP    43                      ;                               2132
000252  000406                         .WORD   406
000254  000000G                         .WORD   SND.BUF
000256  000042                         .WORD   42
000260  000000G                         .WORD   MSGADR
000262  177777                         .WORD   -1
000264  000000                         .WORD   0
000266  026664                         .WORD   26664
000270  013746  000000G                 MOV     SND.BUF,-(SP)           ;                               2133
000274  012746  000000G                 MOV     #SND.BUF,-(SP)
000300  004737  000000V                 JSR     PC,BTOD
000304  012716  000020                  MOV     #20,(SP)                ;                               2134
000310  004737  000000G                 JSR     PC,SEND.DATA
000314  006000                         ROR     R0
000316  103002                         BCC     9$
000320  004737  000000G          8$:    JSR     PC,DECODE
000324  022626                 9$:     CMP     (SP)+,(SP)+             ;                               2131
000326  000207                         RTS     PC                      ;                               2081
000330  020227  000007          10$:    CMP     R2,#7                   ;                               2137
000334  001402                         BEQ     11$
000336  000137  003036'                 JMP     36$
000342  012701  000020          11$:    MOV     #20,R1                  ; *,CNUM                        2139
000346  104450                         TRAP    50                      ;                               2140
000350  103027                         BHIS    14$
000352  104443                         TRAP    43                      ;                               2142
```

```
000354  000406                          .WORD   406
000356  000000G                         .WORD   SND.BUF
000360  000142                          .WORD   142
000362  000000G                         .WORD   MSGADR
000364  177777                          .WORD   -1
000366  000001                          .WORD   1
000370  000100                          .WORD   100
000372  012700  000000G                 MOV     #SND.BUF,R0             ; *,CPTR                2144
000376  005001                          CLR     R1                      ; CNUM                  2145
000400  105710                  12$:    TSTB    (R0)                    ; CPTR                  2146
000402  001403                          BEQ     13$
000404  005201                          INC     R1                      ; CNUM                  2148
000406  005200                          INC     R0                      ; CPTR                  2149
000410  000773                          BR      12$                     ;                       2146
000412  032701  000001          13$:    BIT     #1,R1                   ; *,CNUM                2151
000416  001416                          BEQ     15$
000420  112710  000056                  MOVB    #56,(R0)                ; *,CPTR                2153
000424  005201                          INC     R1                      ; CNUM                  2154
000426  000412                          BR      15$                     ;                       2140
000430  012746  000000G         14$:    MOV     #DEF.SERIAL,-(SP)       ;                       2157
000434  012746  000000G                 MOV     #SND.BUF,-(SP)
000440  012746  000010                  MOV     #10,-(SP)
000444  004737  000000G                 JSR     PC,COPY
000450  062706  000006                  ADD     #6,SP
000454  010146                  15$:    MOV     R1,-(SP)                ; CNUM,*                2158
000456  000137  002620'                 JMP     30$
000462  020127  000002          16$:    CMP     R1,#2                   ;                       2171
000466  001402                          BEQ     17$
000470  000137  002634'                 JMP     31$
000474  013702  000000G         17$:    MOV     REC.BUF,R2              ;                       2174
000500  042702  170000                  BIC     #170000,R2
000504  020227  000001                  CMP     R2,#1                   ;                       2177
000510  001044                          BNE     21$
000512  104450                          TRAP    50                      ;                       2185
000514  103011                          BHIS    18$
000516  104443                          TRAP    43                      ;                       2188
000520  000406                          .WORD   406
000522  000000G                         .WORD   SND.BUF
000524  000052                          .WORD   52
000526  000000G                         .WORD   MSGADR
000530  177777                          .WORD   -1
000532  000000                          .WORD   0
000534  000017                          .WORD   17
000536  000415                          BR      19$                     ;                       2185
000540  013737  000000G 000000G  18$:    MOV     UNIT.NO,SND.BUF         ;                       2192
000546  013746  000000G                 MOV     UNIT.NO,-(SP)           ;                       2193
000552  012746  000000G                 MOV     #FMT2,-(SP)
000556  012746  000002                  MOV     #2,-(SP)
000562  010600                          MOV     SP,R0                   ; SP,*
000564  104414                          TRAP    14
000566  062706  000006                  ADD     #6,SP                   ;                       2191
000572  023727  000000G 000012  19$:    CMP     SND.BUF,#12             ;                       2197
000600  103404                          BLO     20$
```

```
000602  062737  030047  000000G             ADD     #30047,SND.BUF
000610  000530                              BR      29$
000612  062737  000060  000000G     20$:    ADD     #60,SND.BUF             ;                       2198
000620  000524                              BR      29$                     ;                       2200
000622  020227  000002              21$:    CMP     R2,#2                   ;                       2204
000626  001032                              BNE     22$
000630  012737  000012  000000G             MOV     #12,SND.BUF             ;                       2206
000636  005001                              CLR     R1                      ; CNUM                  2207
000640  104443                              TRAP    43                      ;                       2208
000642  000406                              .WORD   406
000644  000000G                             .WORD   SND.BUF
000646  000052                              .WORD   52
000650  000000G                             .WORD   MSGADR
000652  177777                              .WORD   -1
000654  000000                              .WORD   0
000656  000012                              .WORD   12
000660  023727  000000G 000011              CMP     SND.BUF,#11             ;                       2209
000666  101272                              BHI     15$
000670  013746  000000G                     MOV     SND.BUF,-(SP)           ;                       2210
000674  012746  000000G                     MOV     #SND.BUF,-(SP)
000700  004737  000000V                     JSR     PC,BTOD
000704  012701  000002                      MOV     #2,R1                   ; *,CNUM                2211
000710  022626                              CMP     (SP)+,(SP)+             ;                       2209
000712  000660                              BR      15$                     ;                       2213
000714  020227  000004              22$:    CMP     R2,#4                   ;                       2216
000720  001015                              BNE     24$
000722  104450                              TRAP    50                      ;                       2224
000724  103007                              BHIS    23$
000726  104443                              TRAP    43                      ;                       2227
000730  000404                              .WORD   404
000732  000000G                             .WORD   SND.BUF
000734  000130                              .WORD   130
000736  000000G                             .WORD   MSGADR
000740  000001                              .WORD   1
000742  000433                              BR      26$                     ;                       2228
000744  012737  000131  000000G     23$:    MOV     #131,SND.BUF            ;                       2232
000752  000447                              BR      29$                     ;                       2237
000754  020227  000005              24$:    CMP     R2,#5                   ;                       2241
000760  001011                              BNE     25$
000762  104450                              TRAP    50                      ;                       2244
000764  103026                              BHIS    27$
000766  104443                              TRAP    43                      ;                       2247
000770  000404                              .WORD   404
000772  000000G                             .WORD   SND.BUF
000774  000130                              .WORD   130
000776  000000G                             .WORD   MSGADR
001000  000001                              .WORD   1
001002  000413                              BR      26$                     ;                       2248
001004  020227  000006              25$:    CMP     R2,#6                   ;                       2259
001010  001020                              BNE     28$
001012  104450                              TRAP    50                      ;                       2262
001014  103012                              BHIS    27$
001016  104443                              TRAP    43                      ;                       2265
```

```
001020  000404                          .WORD   404
001022  000000G                         .WORD   SND.BUF
001024  000130                          .WORD   130
001026  000000G                         .WORD   MSGADR
001030  000001                          .WORD   1
001032  062737  000130  000000G  26$:    ADD     #130,SND.BUF                    ;                       2266
001040  000414                          BR      29$                             ;                       2262
001042  012737  000116  000000G  27$:    MOV     #116,SND.BUF                    ;                       2270
001050  000410                          BR      29$                             ;                       2275
001052  104443                  28$:    TRAP    43                              ;                       2281
001054  000406                          .WORD   406
001056  000000G                         .WORD   SND.BUF
001060  000142                          .WORD   142
001062  000000G                         .WORD   MSGADR
001064  177777                          .WORD   -1
001066  000001                          .WORD   1
001070  000100                          .WORD   100
001072  012746  000020          29$:    MOV     #20,-(SP)                       ;                       2282
001076  004737  000000G         30$:    JSR     PC,SEND.DATA
001102  005726                          TST     (SP)+
001104  006000                          ROR     R0
001106  103113                          BCC     39$
001110  000510                          BR      38$
001112  020127  000003          31$:    CMP     R1,#3                           ;                       2288
001116  001017                          BNE     32$
001120  012746  000000G                 MOV     #CRLF,-(SP)                     ;                       2291
001124  012746  000001                  MOV     #1,-(SP)
001130  010600                          MOV     SP,R0                           ; SP,*
001132  104414                          TRAP    14
001134  012716  000000G                 MOV     #MSGADR,(SP)                    ;                       2292
001140  012746  000000G                 MOV     #FMT1,-(SP)
001144  012746  000002                  MOV     #2,-(SP)
001150  010600                          MOV     SP,R0                           ; SP,*
001152  104414                          TRAP    14
001154  000451                          BR      34$                             ;                       2289
001156  020127  000004          32$:    CMP     R1,#4                           ;                       2296
001162  001022                          BNE     33$
001164  012746  000000G                 MOV     #CRLF,-(SP)                     ;                       2299
001170  012746  000001                  MOV     #1,-(SP)
001174  010600                          MOV     SP,R0                           ; SP,*
001176  104414                          TRAP    14
001200  012716  000000G                 MOV     #MSGADR,(SP)                    ;                       2300
001204  012746  000000G                 MOV     #FMT1,-(SP)
001210  012746  000002                  MOV     #2,-(SP)
001214  010600                          MOV     SP,R0                           ; SP,*
001216  104414                          TRAP    14
001220  012776  000001  000020          MOV     #1,@20(SP)                      ; *,ENDFLG              2301
001226  000424                          BR      34$                             ;                       2297
001230  020127  000005          33$:    CMP     R1,#5                           ;                       2305
001234  001024                          BNE     35$
001236  012746  000000G                 MOV     #CRLF,-(SP)                     ;                       2308
001242  012746  000001                  MOV     #1,-(SP)
001246  010600                          MOV     SP,R0                           ; SP,*
```

```
001250  104414                                  TRAP    14
001252  012716  000000G                         MOV     #MSGADR,(SP)            ;                               2309
001256  012746  000000G                         MOV     #FMT1, (SP)
001262  012746  000002                          MOV     #2,-(SP)
001266  010600                                  MOV     SP,R0                   ; SP,*
001270  104414                                  TRAP    14
001272  012776  000002  000020                  MOV     #2,@20(SP)              ; *,ENDFLG                      2310
001300  062706  000010                  34$:    ADD     #10,SP                  ;                               2306
001304  000207                                  RTS     PC                      ;                               2069
001306  020127  000006                  35$:    CMP     R1,#6                   ;                               2314
001312  001004                                  BNE     37$
001314  012737  004001  000000G         36$:    MOV     #4001,RET.STATUS        ;                               2316
001322  000403                                  BR      38$                     ;                               2317
001324  012737  001001  000000G         37$:    MOV     #1001,RET.STATUS        ;                               2323
001332  004737  000000G                 38$:    JSR     PC,DECODE               ;                               2324
001336  000207                          39$:    RTS     PC                      ;                               2064

; Routine Size:  368 words,     Routine Base:  $CODE$ + 1522
; Maximum stack depth per invocation:  9 words


;    2328  1
;    2329  1     routine btod(num, ptr) =
;    2330  2     begin
;    2331  2             local   i, j;
;    2332  2             i = 1;
;    2333  2             j = 0;
;    2334  2             while .num/.i gequ 10 do i = .i * 10;
;    2335  2
;    2336  2             while .i gequ 1 do
;    2337  3             begin
;    2338  3                     j = .j + 1;
;    2339  3                     ch$ptr(.ptr)<0,8> = %c'0' + .num/.i;
;    2340  3                     ptr = ch$plus(.ptr, 1);
;    2341  3                     num = .num - (.num/.i)*.i;
;    2342  3                     i = .i/10;
;    2343  2             end;
;    2344  2             return .j;
;    2345  1     end;


                                        .SBTTL  BTOD FORMATTER SECTION 1
000000  004137  000000G                 BTOD:   JSR     R1,$SAVE3               ;                               2329
000004  012701  000001                          MOV     #1,R1                   ; *,I                           2332
000010  005002                                  CLR     R2                      ; J                             2333
000012  016646  000014                  1$:     MOV     14(SP),-(SP)            ; NUM,*                         2334
000016  010146                                  MOV     R1,-(SP)                ; I,*
000020  004737  000000G                         JSR     PC,BL$DIV
000024  022626                                  CMP     (SP)+,(SP)+
000026  020027  000012                          CMP     R0,#12
000032  103410                                  BLO     2$
000034  010146                                  MOV     R1,-(SP)                ; I,*
000036  012746  000012                          MOV     #12,-(SP)
```

```
000042  004737  000000G                        JSR     PC,BL$MUL
000046  010001                                  MOV     R0,R1                   ; *,I
000050  022626                                  CMP     (SP)+,(SP)+
000052  000757                                  BR      1$
000054  005701                          2$:     TST     R1                      ; I                             2336
000056  001434                                  BEQ     3$
000060  005202                                  INC     R2                      ; J                             2338
000062  016646  000014                          MOV     14(SP),-(SP)            ; NUM,*                         2339
000066  010146                                  MOV     R1,-(SP)                ; I,*
000070  004737  000000G                        JSR     PC,BL$DIV
000074  010003                                  MOV     R0,R3
000076  062703  000060                          ADD     #60,R3
000102  110376  000016                          MOVB    R3,@16(SP)              ; *,PTR
000106  005266  000016                          INC     16(SP)                  ; PTR                           2340
000112  010016                                  MOV     R0,(SP)                 ;                               2341
000114  010146                                  MOV     R1,-(SP)                ; I,*
000116  004737  000000G                        JSR     PC,BL$MUL
000122  160066  000022                          SUB     R0,22(SP)               ; *,NUM
000126  010116                                  MOV     R1,(SP)                 ; I,*                           2342
000130  012746  000012                          MOV     #12,-(SP)
000134  004737  000000G                        JSR     PC,BL$DIV
000140  010001                                  MOV     R0,R1                   ; *,I
000142  062706  000010                          ADD     #10,SP                  ;                               2337
000146  000742                                  BR      2$                      ;                               2336
000150  010200                          3$:     MOV     R2,R0                   ; J,*                           2330
000152  000207                                  RTS     PC                      ;                               2329

; Routine Size:  54 words,      Routine Base:  $CODE$ + 3062
; Maximum stack depth per invocation:  9 words


;   2346  1
;   2347  1     end
;   2348  0     eludom


;                                       OTS external references
                                                .GLOBL  $SAVE3, $SAVE2, BL$DIV, BL$MUL


;                                       PSECT SUMMARY
;
;       Psect Name                      Words     Attributes
;         $OWN$                             2       RW  ,  D  ,  LCL,  REL,  CON
;         $CODE$                          847       RO  ,  I  ,  LCL,  REL,  CON


;                                   Library Statistics
;
;                                           -------- Symbols --------      Pages       Processing
```

```
;       File                                    Total    Loaded   Percent      Mapped      Time
;
;  DISK$USER2:[YOUNG.FMT]ZRQBB0.L16;4             358       208        58          18        00:00.1



;                                       COMMAND QUALIFIERS

;       BLISS/PDP11/LIST ZRQB3.B16/EN:NOEIS/SOURCE=PAGE:53

; Size:          847 code + 2 data words
; Run Time:        00:28.2
; Elapsed Time:    02:26.6
; Lines/CPU Min:    4993
; Lexemes/CPU-Min: 32641
; Memory Used:  306 pages
; Compilation Complete
```

```
;    0001  0     MODULE ZRQB4 (%TITLE 'RDRX DISK FORMATTER'
;    0002  0                   IDENT = 'REV C PATCH 0',
;    0003  0                   ADDRESSING_MODE (ABSOLUTE) ,
;    0004  0                   ENVIRONMENT (NOEIS)
;    0005  0                   ) =
;    0006  1     BEGIN
;    0007  1
;    0008  1     !                 SUBROUTINES
;    0009  1
;    0010  1     library 'ZRQB80.L16';                   !Define RDRX Formatter library
;    0011  1
;    0012  1     require 'BLSMAC.REQ';                   !Define Bliss Macro require file
;    1548  1
;    1549  1     %sbttl 'MODULE DECLARATIONS'
```

```
;    1550  1
;    1551  1     forward routine
;    1552  1         GET_NSD,                                !Get next send descriptor slot index
;    1553  1         GET_NRD,                                !Get next receive descriptor slot index
;    1554  1         LOAD_OUT$STD_BUF,                       !Load out standing command buffer
;    1555  1         GET_CMD$REF,                            !Get unique command reference number
;    1556  1         DECODE : novalue,                       !Decode return status error code
;    1557  1         DUP$I_SERVICE : INT_LNK$TYP novalue,        !Dup/UQ port interrupt service routine
;    1558  1         CTO_WAIT,                               !Command time out wait
;    1559  1         ABORT,                                  !Abort Dup command
;    1560  1         GET_DUST_STATUS,                        !Get Dust Status command
;    1561  1         EX_LOC_PROG,                            !Execute Local Program command
;    1562  1         SEND_DATA,                              !Send Data command
;    1563  1         REC_DATA,                               !Receive Data command
;    1564  1         SET_CNTLR_CHAR,                         !Set Controller Characteristics command
;    1565  1         INT$I_SERVICE : INT_LNK$TYP novalue,        !Initialization sequence interrupt service
;    1566  1         IS_TIMER,                               !Initialization sequence time-out wait
;    1567  1         BOOT_RDRX,                              !Initialize sequence for RDRX controller
;    1568  1         INIT_COM_AREA;                          !Initialize UQ Port communication area
;    1569  1
```

```
;    1570  1      !*
;    1571  1      ! The psect named "code or $code$" is redefinded here
;    1572  1      ! to  be called "ac$code".   This is done to organize
;    1573  1      ! formatter    routine   code  into  a seperate psect.
;    1574  1      !-
;    1575  1      !psect
;    1576  1      !    code = ac$code;
;    1577  1
;    1578  1      !*
;    1579  1      ! Structure declarations used within this module.
;    1580  1      !-
;    1581  1
;    1582  1      structure
;    1583  1
;    1584  1          !*
;    1585  1          ! RDRX  register  accessing  structure. This
;    1586  1          ! structure  allows  RDRX register accessing
;    1587  1          ! to  be transportable between the PDP-11 and
;    1588  1          ! VAX Diagnostic Supervisors.
;    1589  1          !
;    1590  1          ! This  also  defines an access algorithm for
;    1591  1          ! VAX to allow field reference to MBA address
;    1592  1          ! space  without  generating  machine checks.
;    1593  1          !-
;    1594  1
;    1595  1          RDRX [0, P, S, E] =
;    1596  2              begin
;    1597  2
;    1598  2              local
;    1599  2                  RC$S_REG;
;    1600  2
;    1601  2              RC$S_REG = .(RDRX + %upval*0)<0, %bpval, 0>;
;    1602  2              RC$S_REG
;    1603  2              end
;    1604  1              <P, S, E>;
;    1605  1
```

```
;  1606  1      !+
;  1607  1      ! External Declaration of  datums declared outside of this module.
;  1608  1      !-
;  1609  1
;  1610  1      external
;  1611  1          !
;  1612  1          ! Communications area declarations
;  1613  1          !
;  1614  1          COM_AREA : blockvector [REC_ALLOCATE + SND_ALLOCATE + HDR_SIZ, 2, word],
;  1615  1          HEAD_AREA : ref block [4, word] field (HDR_FIELD),
;  1616  1          RECEIVE_RING : ref blockvector [REC_ALLOCATE, 2, word] field (DSC_FIELD),
;  1617  1          SEND_RING : ref blockvector [SND_ALLOCATE, 2, word] field (DSC_FIELD),
;  1618  1          REC_ENVELOPE : blockvector [REC_ALLOCATE, RB_SIZE + 2, word] field (ENV_FIELD),
;  1619  1          SND_ENVELOPE : blockvector [SND_ALLOCATE, SB_SIZE + 2, word] field (ENV_FIELD),
;  1620  1          RET_EN$AD : ref block [RB_SIZE + 2, word] field (ENV_FIELD),
;  1621  1          REC_BUF : block [RECB_SIZE, word] field (RECB_FIELD),
;  1622  1          SND_BUF : vector [SNDB_SIZE, word],
;  1623  1          OUT$STD_BUF : blockvector [REC_ALLOCATE, 2, word] field (OUT$FIELD),
;  1624  1          !
;  1625  1          ! Miscellanious external data declarations
;  1626  1          !
;  1627  1          NRD_SLOT : word,                                !Next receive descriptor slot
;  1628  1          NSD_SLOT : word,                                !Next send descriptor slot
;  1629  1          RDRX_ADDR : ref RDRX field (ISD_FIELD),         !Controller reg access struct
;  1630  1          RET_STATUS : word,                              !Global return status location
;  1631  1          PID_SAVE : word,                                !Saves program indicator field
;  1632  1          VEC_ADDR : word,                                !RDRX interrupt vector address
;  1633  1          UC_VER : byte,                                  !Micro code version storage
;  1634  1          RSVD_STRUCT : vector [4, word],                 !Stores init seq resvered fields
;  1635  1          ISD_STRUCT : blockvector [4, 2, word] field (ISD_FIELD),    !Init seq data
;  1636  1          UNIT_NO : word,                                 !P_table unit number to format
;  1637  1          NXT_CRN : byte,                                 !Stores next cmd ref number
;  1638  1          !
;  1639  1          ! Error Messages Structures
;  1640  1          !
;  1641  1          PFE_STRUCT : vector [22],                       !Port fatal error msg struct
;  1642  1          EMSG_STRUCT : vector [22];                      !Error message structure
;  1643  1          !
;  1644  1          !
```

```
;  1646  1     global routine GET_NSD =        !Chooses the next send slot
;  1647  1
;  1648  1     !++
;  1649  1     ! Functional Description :
;  1650  1     !       This  routine  will determine  which send ring  descriptor  the
;  1651  1     !       port/controller  is polling  and  returns that  dsc slot number
;  1652  1     !       to  the  calling routine.  This host  program  will  call  this
;  1653  1     !       routine each time it wishes to deposit another command into the
;  1654  1     !       command (send) ring.
;  1655  1     !
;  1656  1     ! Formal Parameters :
;  1657  1     !       none
;  1658  1     !
;  1659  1     ! Implicit Inputs :
;  1660  1     !       NSD_SLOT :      Global  storage for  the next  send descriptor slot.
;  1661  1     !                       Stores where the host should place this command for
;  1662  1     !                       processing by the port/controller.
;  1663  1     !
;  1664  1     ! Implicit Outputs :
;  1665  1     !       The   global  storage  "Nsd_slot"  is  updated  to the
;  1666  1     !       present send slot where the port/controller is polling.
;  1667  1     !
;  1668  1     ! Completion Codes :
;  1669  1     !       Returns the contents of "Nsd_slot" to the calling routine.
;  1670  1     !
;  1671  1     ! Side Effects :
;  1672  1     !       none
;  1673  1     !--
;  1674  1
;  1675  2     begin
;  1676  2     !
;  1677  2     ! Increment the next send descriptor_slot by one
;  1678  2     !
;  1679  2     NSD_SLOT = .NSD_SLOT + 1;
;  1680  2     !
;  1681  2     ! Set the slot pointer back to zero if it wraps around
;  1682  2     ! to the top of the ring.
;  1683  2     !
;  1684  2
;  1685  2     if .NSD_SLOT gtru SND_ALLOCATE - 1 then NSD_SLOT = ZERO;
;  1686  2
;  1687  2     !
;  1688  2     ! Return the next send descriptor_slot to the caller
;  1689  2     !
;  1690  2     return .NSD_SLOT;
;  1691  1     end;


                                        .TITLE  ZRQB4 RDRX DISK FORMATTER
                                        .IDENT  /REV C /
                                        .ENABL  AMA

                                        .GLOBL  COM.AREA, HEAD.AREA, RECEIVE.RING
```

```
                                        .GLOBL  SEND.RING, REC.ENVELOPE, SND.ENVELOPE
                                        .GLOBL  RET.EN$AD, REC.BUF, SND.BUF, OUT$STD.BUF
                                        .GLOBL  NRD.SLOT, NSD.SLOT, RDRX.ADDR
                                        .GLOBL  RET.STATUS, PID.SAVE, VEC.ADDR
                                        .GLOBL  UC.VER, RSVD.STRUCT, ISD.STRUCT
                                        .GLOBL  UNIT.NO, NXT.CRN, PFE.STRUCT, EMSG.STRUCT


                                        .SBTTL  GET.NSD GLOBAL ROUTINE DECLARATIONS
000000                                  .PSECT  $CODE$,  RO

000000  005237  000000G          GET.NSD::
                                        INC     NSD.SLOT                        ;                       1679
000004  023727  000000G 000003         CMP     NSD.SLOT,#3                     ;                       1685
000012  101402                  BLOS    1$
000014  005037  000000G                CLR     NSD.SLOT
000020  013700  000000G          1$:    MOV     NSD.SLOT,R0                     ;                       1675
000024  000207                  RTS     PC                                      ;                       1646

; Routine Size:  11 words,      Routine Base:  $CODE$ + 0000
; Maximum stack depth per invocation:  0 words


;   1692  1
```

```
;    1693  1     global routine GET_NRD =        !Chooses the next receive slot
;    1694  1
;    1695  1     !++
;    1696  1     ! Functional Description :
;    1697  1     !       This   routine will determine which receive ring descriptor the
;    1698  1     !       port/controller 's polling and returns that dsc slot number to
;    1699  1     !       the calling routine.  This host program will call this routine
;    1700  1     !       each time it wishes to process another receive ring descriptor.
;    1701  1     !
;    1702  1     ! Formal Parameters :
;    1703  1     !       none
;    1704  1     !
;    1705  1     ! Implicit Inputs :
;    1706  1     !       NRD_SLOT :      Global storage for the next receive descriptor slot.
;    1707  1     !                       Stores  where  the port should return this commands
;    1708  1     !                       response indicator.
;    1709  1     !
;    1710  1     ! Implicit Outputs :
;    1711  1     !       The global storage "Nrd_slot" is updated to the
;    1712  1     !       present  receive slot where the port/controller
;    1713  1     !       is polling.
;    1714  1     !
;    1715  1     ! Completion Codes :
;    1716  1     !       Returns the contents of "Nrd_slot" to the calling routine.
;    1717  1     !
;    1718  1     ! Side Effects :
;    1719  1     !       none
;    1720  1     !--
;    1721  1
;    1722  2         begin
;    1723  2         !
;    1724  2         ! Increment the next receive descriptor_slot by one
;    1725  2         !
;    1726  2         NRD_SLOT = .NRD_SLOT + 1;
;    1727  2         !
;    1728  2         ! Set the slot pointer back to zero if it wraps around
;    1729  2         ! to the top of the ring.
;    1730  2         !
;    1731  2
;    1732  2         if .NRD_SLOT gtru REC_ALLOCATE - 1 then NRD_SLOT = ZERO;
;    1733  2
;    1734  2         !
;    1735  2         ! Return the next receive descriptor_slot to the caller
;    1736  2         !
;    1737  2         return .NRD_SLOT;
;    1738  1         end;

                                        .SBTTL  GET.NRD GLOBAL ROUTINE DECLARATIONS
000000  005237  000000G          GET.NRD::
                                        INC     NRD.SLOT                        ;                       1726
000004  023727  000000G 000003         CMP     NRD.SLOT,#3                     ;                       1732
000012  101402                          BLOS    1$
```

```
000014  005037  000000G                    CLR     NRD.SLOT
000020  013700  000000G            1$:     MOV     NRD.SLOT,R0                     ;                       1722
000024  000207                            RTS     PC                              ;                       1693

; Routine Size:  11 words,      Routine Base:  $CODE$ + 0026
; Maximum stack depth per invocation:  0 words


;   1739  1
```

```
;  1740  1     global routine LOAD_OUT$STD_BUF (REF_NUM) =     !Load out$std_buffer with this command
;  1741  1
;  1742  1     !++
;  1743  1     ! Functional Description :
;  1744  1     !       The  outstanding  command  buffer "out$std_buf"  is  used by
;  1745  1     !       this  host  program to  determine  if an outstanding command
;  1746  1     !       issued to the  port has  been  processed yet.  This  is done
;  1747  1     !       by  examining the  receive flag 'Rec_flg' in  a  buffer slot
;  1748  1     !       for  a '1' which  is  set by  the interrupt service  routine
;  1749  1     !       during  response ring interrupts.
;  1750  1     !
;  1751  1     !       This buffer can be  looked  at as  a window between the port
;  1752  1     !       driver receiving & processing the responce envelopes and the
;  1753  1     !       host class driver issuing commands to the port.
;  1754  1     !
;  1755  1     !       This  routine   loads   into  an  empty out$std_buf slot the
;  1756  1     !       following values:
;  1757  1     !
;  1758  1     !       1.  This commands reference number.
;  1759  1     !       2.  Clears 'rec_flg' indicating this command is outstanding.
;  1760  1     !       3.  Clears out the  second  word in slot where the returned
;  1761  1     !           envelope address will go.
;  1762  1     !
;  1763  1     !
;  1764  1     !       IMPORTANT NOTE:
;  1765  1     !       ---------------
;  1766  1     !       To  quarentee a command loaded into the out$std_buffer will
;  1767  1     !       never  be lost  (i.e.  having this  routine return a buffer
;  1768  1     !       slot not yet   received  by the  interrupt service routine),
;  1769  1     !       only the  cto_wait (controller  time  out wait) routine  is
;  1770  1     !       permitted to  return  a out$std_buf slot to the unused pool
;  1771  1     !       (i.e.  by loading  a slots   first  word  with %o'100000').
;  1772  1     !       This  routine is therefore guarenteed  to return  an unused
;  1773  1     !       out$std_buffer slot  when this unique  value  of %o'100000'
;  1774  1     !       is  found.  To  further  quarentee this, unique command ref
;  1775  1     !       numbers will never use zero as a reference number.
;  1776  1     !
;  1777  1     ! Formal Parameters :
;  1778  1     !       REF_NUM          This  is the unique reference number of this
;  1779  1     !                        command set to the port.
;  1780  1     !
;  1781  1     ! Implicit Inputs :
;  1782  1     !       none
;  1783  1     !
;  1784  1     ! Implicit Outputs :
;  1785  1     !       none
;  1786  1     !
;  1787  1     ! Completion Codes :
;  1788  1     !       The out$standing buffer slot index where this command was put
;  1789  1     !       is routines value and is returned to the caller.
;  1790  1     !
;  1791  1     ! Side Effects :
;  1792  1     !       none
```

```
;    1793  1     !
;    1794  1     !--
;    1795  1
;    1796  2         begin
;    1797  2
;    1798  2         !+
;    1799  2         ! Search  through  the  out  standing  command  buffer
;    1800  2         ! and  look  for  the  first  open slot.  Return  this
;    1801  2         ! first  slot  index  to the  caller  if  one is found.
;    1802  2         ! If no open slots are found then return an error code.
;    1803  2         !-
;    1804  2
;    1805  2         incru i from 0 to REC_ALLOCATE - 1 do       !Find the first open slot
;    1806  2             !
;    1807  2             ! An open slot is by definition the value
;    1808  2             ! %o'100000'  in  the  slots  first  word.
;    1809  2             !
;    1810  2
;    1811  2             if .OUT$STD_BUF [.i, CMD_WRD] eqlu %o'100000'   !Is this slot open
;    1812  2             then
;    1813  3                 begin
;    1814  3                 OUT$STD_BUF [.i, REC_FLG] = FALSE;  !Clear the received flag
;    1815  3                 OUT$STD_BUF [.i, CMD_REF] = .REF_NUM;       !Load this cmd's ref num
;    1816  3                 OUT$STD_BUF [.i, ENV_ADR] = ZERO;   !Clear the previous env adr
;    1817  3                 return .i;                          !Return buffer index to caller
;    1818  2                 end;
;    1819  2
;    1820  2         !
;    1821  2         ! The buffer is full if the code reaches
;    1822  2         ! here.  This  should  never  happen  so
;    1823  2         ! report   an  error  for debug purposes.
;    1824  2         !
;    1825  2         return RET_STATUS = OBF_CODE;               !Report an 'out$std buffer full" error
;    1826  1         end;


                                .SBTTL  LOAD.OUT$STD.BUF GLOBAL ROUTINE DECLARATIONS
000000  004137  000000G    LOAD.OUT$STD.BUF::
                            JSR     R1,$SAVE2                   ;                       1740
000004  005002              CLR     R2                          ; I                     1805
000006  010200      1$:     MOV     R2,R0                       ; I,*                   1811
000010  006300              ASL     R0
000012  006300              ASL     R0
000014  012701  000000G     MOV     #OUT$STD.BUF,R1
000020  060001              ADD     R0,R1
000022  021127  100000      CMP     (R1),#-100000
000026  001010              BNE     2$
000030  042711  100000      BIC     #100000,(R1)                ;                       1814
000034  116611  000010      MOVB    10(SP),(R1)                 ; REF.NUM,*             1815
000040  005060  000002G     CLR     OUT$STD.BUF+2(R0)           ;                       1816
000044  010200              MOV     R2,R0                       ; I,*                   1813
000046  000207              RTS     PC
000050  005202      2$:     INC     R2                          ; I                     1805
```

```
000052  020227  000003                        CMP     R2,#3                           ; I,*
000056  101753                                  BLOS    1$
000060  012700  002001                        MOV     #2001,R0                        ;                                       1825
000064  010037  000000G                       MOV     R0,RET.STATUS
000070  000207                                 RTS     PC                              ;                                       1740

; Routine Size:  29 words,      Routine Base:  $CODE$ + 0054
; Maximum stack depth per invocation:  4 words


;   1827  1
```

```
;  1828  1     global routine GET_CMD$REF =    !Gets next unique cmd ref number
;  1829  1
;  1830  1     !++
;  1831  1     ! Functional Description :
;  1832  1     !       A  32 bit  unique non-zero  number   used  to identify host commands.
;  1833  1     !       Class  drivers  should  supply  a unique  reference number  in  each
;  1834  1     !       command  that the  send  to a DUP server.  A class driver may supply
;  1835  1     !       a  zero reference number  if it does not need to associate a command
;  1836  1     !       with its end message.
;  1837  1     !
;  1838  1     !       Command reference  numbers must  be unique across all commands that
;  1839  1     !       are outstanding on  the  same connection  i.e., they must be unique
;  1840  1     !       across all outstanding commands  issued by  a  single  class driver
;  1841  1     !       (Host) to  a  single  DUP  server.   The  class driver may re-use a
;  1842  1     !       commands  reference  number   when    the  command    is  no  longer
;  1843  1     !       outstanding -- i.e.,  after  receiving the commands  end message or
;  1844  1     !       after  re-synchronizing  with  the DUP  server.  Command  reference
;  1845  1     !       numbers  need not be unique for commands  issued by different class
;  1846  1     !       drivers --- i.e. commands issued by different  host or commands for
;  1847  1     !       diffeent DUP  servers  from  the same host.  Therefore  controllers
;  1848  1     !       must  internally  use the combination of a command reference number
;  1849  1     !       and the connection on  which the command was received as the unique
;  1850  1     !       identifier of an outstanding command.
;  1851  1     !
;  1852  1     !       This  routine will generate a unique  command reference number and
;  1853  1     !       will search the out$standing command buffer to see if already uesd.
;  1854  1     !       The first unused unique command  reference found will  be returned
;  1855  1     !       to the calling routine.
;  1856  1     !
;  1857  1     ! Formal Parameters :
;  1858  1     !       none
;  1859  1     !
;  1860  1     ! Implicit Inputs :
;  1861  1     !       NXT_CRN          This global location stores the next unique cmd
;  1862  1     !                        reference number to be used.
;  1863  1     !
;  1864  1     ! Implicit Outputs :
;  1865  1     !       NXT_CRN          This global location is loaded with the next
;  1866  1     !                        unigue command reference number.
;  1867  1     !
;  1868  1     ! Completion Codes :
;  1869  1     !       The contents of .NXT_CRN is returned to the calling routine.
;  1870  1     !
;  1871  1     ! Side Effects :
;  1872  1     !       none
;  1873  1     !--
;  1874  1
;  1875  2         begin
;  1876  2
;  1877  2         local
;  1878  2             DONE;
;  1879  2
;  1880  2         !
```

```
;    1881  2          ! Increment  the   global   unique command
;    1882  2          ! reference number before anything is done.
;    1883  2          !
;    1884  2          NXT_CRN = .NXT_CRN + 1;
;    1885  2
;    1886  2          !+
;    1887  2          ! Repeat generating and searching the out$standing
;    1888  2          ! command buffer until a unique command reference
;    1889  2          ! number is found.
;    1890  2          !-
;    1891  2
;    1892  2          do
;    1893  3              begin
;    1894  3              !
;    1895  3              ! Wrap this next command reference number around
;    1896  3              ! back to one if it is greater then 255 (decimal).
;    1897  3              !
;    1898  3
;    1899  3              if .NXT_CRN gequ 255 then NXT_CRN = 1;
;    1900  3
;    1901  3              DONE = TRUE;                            !Clear the all done indicator flag
;    1902  3
;    1903  3              !+
;    1904  3              ! Now  search  the  out$std_buffer for this  command
;    1905  3              ! reference  number.  If  not  there then done stays
;    1906  3              ! true  and the loop will  end else increment to the
;    1907  3              ! next unique command ref number and make done false
;    1908  3              ! to continue the loop.
;    1909  3              !-
;    1910  3
;    1911  3              incru i from 0 to REC_ALLOCATE - 1 do   !Search buffer for this cmd ref num
;    1912  3
;    1913  3              if .OUT$STD_BUF [.i, CMD_REF] eqlu .NXT_CRN !Does it already exist
;    1914  3              then
;    1915  4                  begin                               !It already exists
;    1916  4                  NXT_CRN = .NXT_CRN + 1;             !Try the next sequential cmd ref num
;    1917  4                  DONE = FALSE;                       !Make code loop again
;    1918  4                  exitloop;                           !Exit this incr loop
;    1919  3                  end;
;    1920  3
;    1921  3              end
;    1922  2          until .DONE;                                !Repeat loop until done
;    1923  2
;    1924  2          !
;    1925  2          ! Return the unique command reference number to the caller.
;    1926  2          !
;    1927  2          return .NXT_CRN;
;    1928  1          end;


                                    .SBTTL  GET.CMD$REF GLOBAL ROUTINE DECLARATIONS
000000  010146                 GET.CMD$REF::
                                    MOV     R1,-(SP)                        ;                       1828
```

```
000002  105237  000000G                        INCB    NXT.CRN                         ;                               1884
000006  123727  000000G 000377          1$:     CMPB    NXT.CRN,#377                    ;                               1899
000014  103403                                 BLO     2$
000016  112737  000001  000000G                MOVB    #1,NXT.CRN
000024  012701  000001                  2$:     MOV     #1,R1                           ; *,DONE                        1901
000030  005000                                 CLR     R0                              ; I                             1911
000032  126037  000000G 000000G        3$:     CMPB    OUT$STD.BUF(R0),NXT.CRN         ; *(I),*                        1913
000040  001004                                 BNE     4$
000042  105237  000000G                        INCB    NXT.CRN                         ;                               1916
000046  005001                                 CLR     R1                              ; DONE                          1917
000050  000405                                 BR      5$                              ;                               1915
000052  062700  000004                  4$:     ADD     #4,R0                           ; *,I                           1911
000056  020027  000014                         CMP     R0,#14                          ; I,*
000062  101763                                 BLOS    3$
000064  006001                          5$:     ROR     R1                              ; DONE                          1922
000066  103347                                 BCC     1$
000070  005000                                 CLR     R0                              ;                               1927
000072  153700  000000G                        BISB    NXT.CRN,R0
000076  012601                                 MOV     (SP)+,R1                        ;                               1828
000100  000207                                 RTS     PC

; Routine Size:  33 words,    Routine Base:  $CODE$ + 0146
; Maximum stack depth per invocation:  2 words

;  1929  1
```

```
;    1930  1      global routine DECODE : novalue =       !Decodes failing SA reg data
;    1931  1
;    1932  1      !++
;    1933  1      ! Functional Description :
;    1934  1      !       Due  to the implimentation of the DUP  and UQ Port protocol there
;    1935  1      !       are  two levels  at which an issued command to  a port/controller
;    1936  1      !       can fail and they are:
;    1937  1      !
;    1938  1      !         1. The issued command can time out.
;    1939  1      !
;    1940  1      !         2. An  error  can be posted in SA register bit 15 by the port  to
;    1941  1      !            report an error.
;    1942  1      !
;    1943  1      !         3. The  issued  command  to  the port/controller can  be executed
;    1944  1      !            correctly without  any  errors but the response  packet status
;    1945  1      !            field could have an error or  status other than success posted.
;    1946  1      !
;    1947  1      !       These errors or status's  returned  are  all returned to the host
;    1948  1      !       routine which queued  the  DUP  command  via  the  global storage
;    1949  1      !       "RET_STATUS". The   host   to port/controller   communications
;    1950  1      !       connection  having  the highest priority (ie. if the SA Reg error
;    1951  1      !       bit  is set the DUP interrupt routine returns a PFE_CODE and this
;    1952  1      !       code is  passed  up  to  the  calling host routine with out being
;    1953  1      !       intercepted by any routine on the way up).
;    1954  1      !
;    1955  1      !       This  routine  will  then be called when the return from a queued
;    1956  1      !       command  comes  back with an error code or non successfull status
;    1957  1      !       code.  This is  by  definition  when bit 0 in the returned status
;    1958  1      !       is equal to 1.
;    1959  1      !
;    1960  1      !       An appropriate  recovery  action will be done for each individual
;    1961  1      !       error.
;    1962  1      !
;    1963  1      ! Formal Parameters :
;    1964  1      !       none
;    1965  1      !
;    1966  1      ! Implicit Inputs :
;    1967  1      !       RET_STATUS:     Stored in this global storage is the returned error
;    1968  1      !                       code or  non-successful  status  code from a queued
;    1969  1      !                       command.
;    1970  1      !
;    1971  1      ! Implicit Outputs :
;    1972  1      !       none
;    1973  1      !
;    1974  1      ! Completion Codes :
;    1975  1      !       none
;    1976  1      !
;    1977  1      ! Side Effects :
;    1978  1      !       All formatter errors are fatal, therefor after execution
;    1979  1      !       of  this  routine  the  RDRX  controller  is initialized
;    1980  1      !       aborting any DM code running in the controller.
;    1981  1      !--
;    1982  1
```

```
;  1983  2          begin
;  1984  2
;  1985  2          !+
;  1986  2          ! Use the contents of "RET_STATUS" to select what
;  1987  2          ! type error or non-successful status code is to
;  1988  2          ! be processed.
;  1989  2          !-
;  1990  2
;  1991  2          selectoneu .RET_STATUS of
;  1992  2              set
;  1993  2              !
;  1994  2              ! "Communication area initialize" error code
;  1995  2              !
;  1996  2              ! This error  code indicates that  the  port did
;  1997  2              ! not init the com area in the host memory after
;  1998  2              ! step  2  of   the    initialization   sequence.
;  1999  2              !
;  2000  2
;  2001  2              [CIE_CODE] :                            !Code equals %o'01'
;  2002  3                  begin
;  2003  3                  PRINTF (.EMSG_STRUCT [MSG4]);
;  2004  2                  end;
;  2005  2              !
;  2006  2              ! "Port/Controller time out" error code
;  2007  2              !
;  2008  2              ! Port/Controller timed out after the specified
;  2009  2              ! time out interval.
;  2010  2              !
;  2011  2
;  2012  2              [CTO_CODE] :                            !Code equals %o'11'
;  2013  3                  begin
;  2014  3                  PRINTF (.EMSG_STRUCT [MSG21]);
;  2015  2                  end;
;  2016  2              !
;  2017  2              ! "Port fatal error" code
;  2018  2              !
;  2019  2              ! The error bit in the SA Register was set when
;  2020  2              ! examined in the DUP$I_SERVICE routine.  This
;  2021  2              ! error indicates a Port fatal error code.
;  2022  2              !
;  2023  2
;  2024  2              [PFE_CODE] :                            !Code equals %o'21'
;  2025  3                  begin
;  2026  3                  PRINTF (.PFE_STRUCT [.RDRX_ADDR [RCSA, ERR_CODE]]);
;  2027  2                  end;
;  2028  2              !
;  2029  2              ! "Return status error" code
;  2030  2              !
;  2031  2              ! This indicates that a non-successful return status
;  2032  2              ! code was returned  from an issued command.
;  2033  2              !
;  2034  2
;  2035  2              [RSE_CODE] :                            !Code equals %o'31'
```

```
;  2036  3                 begin
;  2037  3                 PRINTF (.EMSG_STRUCT [MSG0]);
;  2038  2                 end;
;  2039  2             !
;  2040  2             ! "Port Portocol violation" error code
;  2041  2             !
;  2042  2             ! A protocol violation error was detected during
;  2043  2             ! host processing of an issued command.
;  2044  2             !
;  2045  2
;  2046  2             [PVE_CODE] :                            !Code equals %o'41'
;  2047  3                 begin
;  2048  3                 PRINTF (.EMSG_STRUCT [MSG1]);
;  2049  2                 end;
;  2050  2             !
;  2051  2             ! "Remote program died" error code
;  2052  2             !
;  2053  2             ! This indicates that the remote program running
;  2054  2             ! in the DM machine did not responded within the
;  2055  2             ! designated time out interval and that the progress
;  2056  2             ! indicator was not increase after subsiquent time out
;  2057  2             ! delays.  It is assumed that the remote program is dead
;  2058  2             ! and is treated as a fatal error.
;  2059  2             !
;  2060  2
;  2061  2             [RPD_CODE] :                            !Code equals %o'51'
;  2062  3                 begin
;  2063  3                 PRINTF (.EMSG_STRUCT [MSG2]);
;  2064  2                 end;
;  2065  2             !
;  2066  2             ! "Port to host synchronious error" code
;  2067  2             !
;  2068  2
;  2069  2             [PSE_CODE] :                            !Code equals %o'61'
;  2070  3                 begin
;  2071  3                 PRINTF (.EMSG_STRUCT [MSG5]);
;  2072  2                 end;
;  2073  2             !
;  2074  2             ! "Message length error" code
;  2075  2             !
;  2076  2
;  2077  2             [MLE_CODE] :                            !Code equals %o'71'
;  2078  3                 begin
;  2079  3                 PRINTF (.EMSG_STRUCT [MSG6]);
;  2080  2                 end;
;  2081  2             !
;  2082  2             ! "Unknown end code" error code
;  2083  2             !
;  2084  2
;  2085  2             [UEC_CODE] :                            !Code equals %o'101'
;  2086  3                 begin
;  2087  3                 PRINTF (.EMSG_STRUCT [MSG7]);
;  2088  2                 end;
```

```
;  2089  2              !
;  2090  2              ! "Adaptor purge request" error code
;  2091  2              !
;  2092  2
;  2093  2              [APR_CODE] :                            !Code equals %o'201'
;  2094  3                  begin
;  2095  3                  PRINTF (.EMSG_STRUCT [MSG8]);
;  2096  2                  end;
;  2097  2              !
;  2098  2              ! "Unknown interrupt" error code
;  2099  2              !
;  2100  2
;  2101  2              [UIN_CODE] :                            !Code equals %o'301'
;  2102  3                  begin
;  2103  3                  PRINTF (.EMSG_STRUCT [MSG9]);
;  2104  2                  end;
;  2105  2              !
;  2106  2              ! "ATTENTION MSG ENDCODE" error code
;  2107  2              !
;  2108  2
;  2109  2              [ATN_CODE] :                            !Code equals %o'401'
;  2110  3                  begin
;  2111  3                  PRINTF (.EMSG_STRUCT [MSG12]);
;  2112  2                  end;
;  2113  2              !
;  2114  2              ! "COMMAND MSG ENDCODE" error code
;  2115  2              !
;  2116  2
;  2117  2              [CMD_CODE] :                            !Code equals %o'501'
;  2118  3                  begin
;  2119  3                  PRINTF (.EMSG_STRUCT [MSG13]);
;  2120  2                  end;
;  2121  2              !
;  2122  2              ! "SERIOUS EXCEPTION" error code
;  2123  2              !
;  2124  2
;  2125  2              [SEX_CODE] :                            !Code equals %o'601'
;  2126  3                  begin
;  2127  3                  PRINTF (.EMSG_STRUCT [MSG14]);
;  2128  2                  end;
;  2129  2              !
;  2130  2              ! "INVALID COMMAND" error code
;  2131  2              !
;  2132  2
;  2133  2              [IVC_CODE] :                            !Code equals %o'701'
;  2134  3                  begin
;  2135  3                  PRINTF (.EMSG_STRUCT [MSG15]);
;  2136  2                  end;
;  2137  2              !
;  2138  2              ! "UNKNOWN MESSAGE TYPE" error code
;  2139  2              !
;  2140  2
;  2141  2              [UMT_CODE] :                            !Code equals %o'1001'
```

```
;  2142  3                begin
;  2143  3                PRINTF (.EMSG_STRUCT [MSG16]);
;  2144  2                end;
;  2145  2            !
;  2146  2            ! "UNKNOWN MESSAGE NUMBER" error code
;  2147  2            !
;  2148  2
;  2149  2            [UMN_CODE] :                                !Code equals %o'4001'
;  2150  3                begin
;  2151  3                PRINTF (.EMSG_STRUCT [MSG19]);
;  2152  2                end;
;  2153  2            !
;  2154  2            ! Out standing buffer slots are all filled up
;  2155  2            !
;  2156  2
;  2157  2            [OBF_CODE] :                                !Code equals %o'2001'
;  2158  3                begin
;  2159  3                PRINTF (.EMSG_STRUCT [MSG17]);
;  2160  2                end;
;  2161  2            !
;  2162  2            ! Out standing command buffer out of sync error
;  2163  2            !
;  2164  2
;  2165  2            [OSE_CODE] :                                !Code equals %o'3001'
;  2166  3                begin
;  2167  3                PRINTF (.EMSG_STRUCT [MSG18]);
;  2168  2                end;
;  2169  2            !
;  2170  2            ! File read error from local load media
;  2171  2            !
;  2172  2
;  2173  2            [FRE_CODE] :                                !Code equals %o'5001'
;  2174  3                begin
;  2175  3                PRINTF (.EMSG_STRUCT [MSG20]);
;  2176  2                end;
;  2177  2            !
;  2178  2            ! This is here to trap any unknow return status codes
;  2179  2            ! sent to this routine.
;  2180  2            !
;  2181  2
;  2182  2            [otherwise] :                               !Code equals non of the above
;  2183  3                begin
;  2184  3                PRINTF (.EMSG_STRUCT [MSG3]);
;  2185  2                end;
;  2186  2            tes;
;  2187  2        !
;  2188  2        ! All errors are fatal so init the RDRX
;  2189  2        ! and jump to the clean-up code section
;  2190  2        ! to abort this units format.
;  2191  2        !
;  2192  2
;  2193  2        WRT_RDRX (RCIP, RC_ALL, ONES);                  !Init the controller
;  2194  2        DOCLN;                                          !Jump to the clean-up code section
```

```
;  2195  2
;  2196  1        end;


                                        .SBTTL  DECODE GLOBAL ROUTINE DECLARATIONS
000000  010146                  DECODE::MOV     R1,-(SP)                        ;                               1930
000002  024646                          CMP     -(SP),-(SP)
000004  013701  000000G                 MOV     RET.STATUS,R1                   ;                               1991
000010  020127  000001                  CMP     R1,#1                           ;                               2001
000014  001007                          BNE     1$
000016  013746  000010G                 MOV     EMSG.STRUCT+10,-(SP)            ;                               2003
000022  012746  000001                  MOV     #1,-(SP)
000026  010600                          MOV     SP,R0                           ; SP,*
000030  104417                          TRAP    17
000032  000577                          BR      13$                             ;                               2002
000034  020127  000011          1$:     CMP     R1,#11                          ;                               2012
000040  001007                          BNE     2$
000042  013746  000052G                 MOV     EMSG.STRUCT+52,-(SP)            ;                               2014
000046  012746  000001                  MOV     #1,-(SP)
000052  010600                          MOV     SP,R0                           ; SP,*
000054  104417                          TRAP    17
000056  000577                          BR      15$                             ;                               2015
000060  020127  000021          2$:     CMP     R1,#21                          ;                               2024
000064  001017                          BNE     3$
000066  013700  000000G                 MOV     RDRX.ADDR,R0                    ;                               2026
000072  016016  000002                  MOV     2(R0),(SP)                      ; *,RC$S.REG
000076  011600                          MOV     (SP),R0                         ; RC$S.REG,*
000100  042700  174000                  BIC     #174000,R0
000104  006300                          ASL     R0
000106  016046  000000G                 MOV     PFE.STRUCT(R0),-(SP)
000112  012746  000001                  MOV     #1,-(SP)
000116  010600                          MOV     SP,R0                           ; SP,*
000120  104417                          TRAP    17
000122  000567                          BR      17$                             ;                               2025
000124  020127  000031          3$:     CMP     R1,#31                          ;                               2035
000130  001007                          BNE     4$
000132  013746  000000G                 MOV     EMSG.STRUCT,-(SP)               ;                               2037
000136  012746  000001                  MOV     #1,-(SP)
000142  010600                          MOV     SP,R0                           ; SP,*
000144  104417                          TRAP    17
000146  000567                          BR      19$                             ;                               2036
000150  020127  000041          4$:     CMP     R1,#41                          ;                               2046
000154  001007                          BNE     5$
000156  013746  000002G                 MOV     EMSG.STRUCT+2,-(SP)             ;                               2048
000162  012746  000001                  MOV     #1,-(SP)
000166  010600                          MOV     SP,R0                           ; SP,*
000170  104417                          TRAP    17
000172  000567                          BR      21$                             ;                               2047
000174  020127  000051          5$:     CMP     R1,#51                          ;                               2061
000200  001007                          BNE     6$
000202  013746  000004G                 MOV     EMSG.STRUCT+4,-(SP)             ;                               2063
000206  012746  000001                  MOV     #1,-(SP)
000212  010600                          MOV     SP,R0                           ; SP,*
```

```
000214  104417                          TRAP    17
000216  000567                          BR      23$                             ;                               2062
000220  020127  000061          6$:     CMP     R1,#61                          ;                               2069
000224  001007                          BNE     7$
000226  013746  000012G                 MOV     EMSG.STRUCT+12,-(SP)            ;                               2071
000232  012746  000001                  MOV     #1,-(SP)
000236  010600                          MOV     SP,R0                           ; SP,*
000240  104417                          TRAP    17
000242  000567                          BR      25$                             ;                               2070
000244  020127  000071          7$:     CMP     R1,#71                          ;                               2077
000250  001007                          BNE     8$
000252  013746  000014G                 MOV     EMSG.STRUCT+14,-(SP)            ;                               2079
000256  012746  000001                  MOV     #1,-(SP)
000262  010600                          MOV     SP,R0                           ; SP,*
000264  104417                          TRAP    17
000266  000576                          BR      28$                             ;                               2078
000270  020127  000101          8$:     CMP     R1,#101                         ;                               2085
000274  001007                          BNE     9$
000276  013746  000016G                 MOV     EMSG.STRUCT+16,-(SP)            ;                               2087
000302  012746  000001                  MOV     #1,-(SP)
000306  010600                          MOV     SP,R0                           ; SP,*
000310  104417                          TRAP    17
000312  000564                          BR      28$                             ;                               2086
000314  020127  000201          9$:     CMP     R1,#201                         ;                               2093
000320  001007                          BNE     10$
000322  013746  000020G                 MOV     EMSG.STRUCT+20,-(SP)            ;                               2095
000326  012746  000001                  MOV     #1,-(SP)
000332  010600                          MOV     SP,R0                           ; SP,*
000334  104417                          TRAP    17
000336  000552                          BR      28$                             ;                               2094
000340  020127  000301          10$:    CMP     R1,#301                         ;                               2101
000344  001007                          BNE     11$
000346  013746  000022G                 MOV     EMSG.STRUCT+22,-(SP)            ;                               2103
000352  012746  000001                  MOV     #1,-(SP)
000356  010600                          MOV     SP,R0                           ; SP,*
000360  104417                          TRAP    17
000362  000540                          BR      28$                             ;                               2102
000364  020127  000401          11$:    CMP     R1,#401                         ;                               2109
000370  001007                          BNE     12$
000372  013746  000030G                 MOV     EMSG.STRUCT+30,-(SP)            ;                               2111
000376  012746  000001                  MOV     #1,-(SP)
000402  010600                          MOV     SP,R0                           ; SP,*
000404  104417                          TRAP    17
000406  000526                          BR      28$                             ;                               2110
000410  020127  000501          12$:    CMP     R1,#501                         ;                               2117
000414  001007                          BNE     14$
000416  013746  000032G                 MOV     EMSG.STRUCT+32,-(SP)            ;                               2119
000422  012746  000001                  MOV     #1,-(SP)
000426  010600                          MOV     SP,R0                           ; SP,*
000430  104417                          TRAP    17
000432  000514                  13$:    BR      28$                             ;                               2118
000434  020127  000601          14$:    CMP     R1,#601                         ;                               2125
000440  001007                          BNE     16$
```

```
000442  013746  000034G                        MOV     EMSG.STRUCT+34,-(SP)            ;                               2127
000446  012746  000001                          MOV     #1,-(SP)
000452  010600                                  MOV     SP,R0                           ; SP,*
000454  104417                                  TRAP    17
000456  000502                          15$:    BR      28$                             ;                               2126
000460  020127  000701                  16$:    CMP     R1,#701                         ;                               2133
000464  001007                                  BNE     18$
000466  013746  000036G                        MOV     EMSG.STRUCT+36,-(SP)            ;                               2135
000472  012746  000001                          MOV     #1,-(SP)
000476  010600                                  MOV     SP,R0                           ; SP,*
000500  104417                                  TRAP    17
000502  000470                          17$:    BR      28$                             ;                               2134
000504  020127  001001                  18$:    CMP     R1,#1001                        ;                               2141
000510  001007                                  BNE     20$
000512  013746  000040G                        MOV     EMSG.STRUCT+40,-(SP)            ;                               2143
000516  012746  000001                          MOV     #1,-(SP)
000522  010600                                  MOV     SP,R0                           ; SP,*
000524  104417                                  TRAP    17
000526  000456                          19$:    BR      28$                             ;                               2142
000530  020127  004001                  20$:    CMP     R1,#4001                        ;                               2149
000534  001007                                  BNE     22$
000536  013746  000046G                        MOV     EMSG.STRUCT+46,-(SP)            ;                               2151
000542  012746  000001                          MOV     #1,-(SP)
000546  010600                                  MOV     SP,R0                           ; SP,*
000550  104417                                  TRAP    17
000552  000444                          21$:    BR      28$                             ;                               2150
000554  020127  002001                  22$:    CMP     R1,#2001                        ;                               2157
000560  001007                                  BNE     24$
000562  013746  000042G                        MOV     EMSG.STRUCT+42,-(SP)            ;                               2159
000566  012746  000001                          MOV     #1,-(SP)
000572  010600                                  MOV     SP,R0                           ; SP,*
000574  104417                                  TRAP    17
000576  000432                          23$:    BR      28$                             ;                               2158
000600  020127  003001                  24$:    CMP     R1,#3001                        ;                               2165
000604  001007                                  BNE     26$
000606  013746  000044G                        MOV     EMSG.STRUCT+44,-(SP)            ;                               2167
000612  012746  000001                          MOV     #1,-(SP)
000616  010600                                  MOV     SP,R0                           ; SP,*
000620  104417                                  TRAP    17
000622  000420                          25$:    BR      28$                             ;                               2166
000624  020127  005001                  26$:    CMP     R1,#5001                        ;                               2173
000630  001007                                  BNE     27$
000632  013746  000050G                        MOV     EMSG.STRUCT+50,-(SP)            ;                               2175
000636  012746  000001                          MOV     #1,-(SP)
000642  010600                                  MOV     SP,R0                           ; SP,*
000644  104417                                  TRAP    17
000646  000406                                  BR      28$                             ;                               2174
000650  013746  000006G                  27$:    MOV     EMSG.STRUCT+6,-(SP)             ;                               2184
000654  012746  000001                          MOV     #1,-(SP)
000660  010600                                  MOV     SP,R0                           ; SP,*
000662  104417                                  TRAP    17
000664  022626                          28$:    CMP     (SP)+,(SP)+                     ;                               2183
000666  017766  000000G 000002                  MOV     @RDRX.ADDR,2(SP)                ; *,RC$S.REG                    2193
```

```
000674  012700  177777                    MOV     #-1,R0                  ; *,RC$M.REG
000700  010077  000000G                   MOV     R0,@RDRX.ADDR           ; RC$M.REG,*
000704  104444                            TRAP    44
000706  022626                            CMP     (SP)+,(SP)+             ;                                       1930
000710  012601                            MOV     (SP)+,R1
000712  000207                            RTS     PC

; Routine Size:  230 words,    Routine Base:  $CODE$ + 0250
; Maximum stack depth per invocation:  7 words


;   2197  1
```

```
;  2198  1     global routine DUP$I_SERVICE : INT_LNK$TYP novalue =    !Signals receive queue entry
;  2199  1
;  2200  1     !++
;  2201  1     ! Functional Description :
;  2202  1     !       The transmission of a message will result in a host interrupt if and
;  2203  1     !       only if interrupts were armed suitably during initialization and one
;  2204  1     !       of the following conditions has been met:
;  2205  1     !
;  2206  1     !       1.  The message was a command with F=1 and the port's fetching it
;  2207  1     !           caused  the  command ring to transition from full to not-full.
;  2208  1     !           (This interrupt means that the host may place another command
;  2209  1     !           in the ring.)
;  2210  1     !
;  2211  1     !       2.  The message was a response with F=1 and the port's depositing
;  2212  1     !           it caused the response ring to transition from empty to not-
;  2213  1     !           empty.  (This  interrupt  means  that there is a response for
;  2214  1     !           the host to process.)
;  2215  1     !
;  2216  1     !       3.  The  port  is  interfaced to the host via a bus abapter and a
;  2217  1     !           command requires  the  port/controller  to  re-access a given
;  2218  1     !           location  during   data transfer.  (This interrupt means that
;  2219  1     !           the  port/controller  is  requesting  the  host  to purge the
;  2220  1     !           indicated chanel of the bus adapter.)
;  2221  1     !
;  2222  1     !       This  interrupt  service routine is entered when any of the above
;  2223  1     !       conditions  occure. When  entered it will be determined what type
;  2224  1     !       interrupt was executed and take the necessary action.
;  2225  1     !
;  2226  1     ! Formal Parameters :
;  2227  1     !       none
;  2228  1     !
;  2229  1     ! Implicit Inputs :
;  2230  1     !       Nrd_slot:       A global flag which points to the next receive descriptor
;  2231  1     !                       slot where the port/controller should be polling on and
;  2232  1     !                       where to expect the first response packet to process.
;  2233  1     !
;  2234  1     ! Implicit Outputs :
;  2235  1     !       Ret_status:     This  global  flag  is the mechinism by which these DUP
;  2236  1     !                       and  UQ Port protocol routines pass status code back to
;  2237  1     !                       to  the  host  routine's requesting communications over
;  2238  1     !                       the  established  connections.  The status returned is
;  2239  1     !                       decoded by  the  caller  to determine if an error or bad
;  2240  1     !                       response packet status was discovered.
;  2241  1     !
;  2242  1     !       Out$std_buf:    This  buffer  is used to save all commands issued to the
;  2243  1     !                       port  and are considered outstanding when in this buffer.
;  2244  1     !                       This interrupt service routine will indicate this command
;  2245  1     !                       is  nolonger  outstanding by setting the rec_flg in the
;  2246  1     !                       slot  matching this response envelope command ref number.
;  2247  1     !
;  2248  1     ! Completion Codes :
;  2249  1     !       none
;  2250  1     !
```

```
;  2251  1     ! Side Effects :
;  2252  1     !       none
;  2253  1     !--
;  2254  1
;  2255  2         begin
;  2256  2
;  2257  2         local
;  2258  2             TEMP,                                   !Holds nrd_slot + 1 reference
;  2259  2             FOUND_CMD,                              !Found command flag
;  2260  2             REF_NUM;                                !Stores response packets cmd ref number
;  2261  2
;  2262  2         !+
;  2263  2         ! Before this  interrupt service routine does anything
;  2264  2         ! look   at the SA  register for any port fatal errors
;  2265  2         ! posted.  If there are  errors posted then report the
;  2266  2         ! error and kick the bucket.
;  2267  2         !-
;  2268  2
;  2269  2         if .RDRX_ADDR [RCSA, ERR_BIT]               !Are there any errors posted
;  2270  2         then
;  2271  3             begin
;  2272  3             RET_STATUS = PFE_CODE;                  !Indicate the error type
;  2273  3             DECODE ();                              !Decode and print the error
;  2274  3             return;                                 !Just for show. Decode will kill it
;  2275  2             end;
;  2276  2
;  2277  2         !+
;  2278  2         ! See what kind of interrupt got us here.
;  2279  2         !
;  2280  2         ! We could have a:
;  2281  2         !
;  2282  2         !   1. Response   ring   transition    interrupt.
;  2283  2         !   2. Send  ring  transition interrupt.
;  2284  2         !   3. A  adaptor  purge request  interrupt  (which is
;  2285  2         !      illegal  running  under  the  PDP-11 Diagnostic
;  2286  2         !      supervisor and is flagged as a fatal controller
;  2287  2         !      error.)
;  2288  2         !   4. Or an unknown interrupt not known by this program
;  2289  2         !      which  also  results  in a fatal controller error.
;  2290  2         !
;  2291  2         !-
;  2292  2
;  2293  2         !
;  2294  2
;  2295  2         ! Check to see if we get here because of a response ring
;  2296  2         ! transition  interrupt.  This is  more likely to be the
;  2297  2         ! most frequent interrupt so check it first.
;  2298  2         !
;  2299  2
;  2300  2         if .HEAD_AREA [RSP_INT]
;  2301  2         then
;  2302  3             begin
;  2303  3             HEAD_AREA [RSP_INT] = ZEROS;            !Clear the interrupt indicator location
```

```
;  2304  3              GET_NRD ();                                 !Get the resp slot location to process
;  2305  3              !
;  2306  3              ! Check the host  protocol for being out of sequence
;  2307  3              ! with the controller by  making sure that this slot
;  2308  3              ! is owned by the host (meaning that there is a resp
;  2309  3              ! envelope at this ring slot to process.
;  2310  3              !
;  2311  3
;  2312  3              if .RECEIVE_RING [.NRD_SLOT, OWN_BIT] nequ HOST_OWNED   !Is this owned by host
;  2313  3              then
;  2314  4                  begin                                   !Host/port is out of sequence
;  2315  4                  RET_STATUS = PSE_CODE;                  !Load a "Port sync error" code
;  2316  4                  DECODE ();                              !Report the error and kick the bucket
;  2317  4                  return;                                 !Just for show Decode kills it
;  2318  3                  end;
;  2319  3
;  2320  3              !+
;  2321  3              ! Per DUP protocol  once interrupted  due  to  a response ring
;  2322  3              ! interrupt, the host code should process all response packets
;  2323  3              ! found  in the response  ring.  This while loop will continue
;  2324  3              ! to process the response packets  in  the response ring until
;  2325  3              ! none remain.
;  2326  3              !-
;  2327  3
;  2328  3              while TRUE do                               !Process all response packets in ring
;  2329  4                  begin
;  2330  4                  BREAK;                                  !Look for control c's
;  2331  4                  !
;  2332  4                  ! Load the  Reference structure "Ret_en$ad" with the address
;  2333  4                  ! of this response  envelope  to process (The minus %o'4' is
;  2334  4                  ! done   to  address  the first word in the envelope  packet
;  2335  4                  ! and is equal to location "text-4").
;  2336  4                  !
;  2337  4                  RET_EN$AD = (.RECEIVE_RING [.NRD_SLOT, LO_EN$AD]) - %o'4';
;  2338  4
;  2339  4              !+
;  2340  4              ! Test the end packet for its possible three end types.
;  2341  4              !
;  2342  4              ! End message opcodes  (also called endcodes) are formed by adding   the end
;  2343  4              ! message  flag  to  the  command opcode.  For  example, a READ commands end
;  2344  4              ! message contains the value OP.RED + OP.END in its opcode field. The Invalid
;  2345  4              ! command  end  message contains just the end message flag (i.e., OP.END) in
;  2346  4              ! its opcode field.  The serious exception opcode shown above (i.e. OP.SEX +
;  2347  4              ! OP.END) in its opcode field.
;  2348  4              !
;  2349  4              ! Commands opcode bits 6 and 7 indicate the type of message (command, end or
;  2350  4              ! attention  message.  Command opcodes bits 3 through 5 indicate the command
;  2351  4              ! catagory  (immediate, sequential  or no-sequential) and whether or not the
;  2352  4              ! command includes a buffer descriptor.
;  2353  4              !
;  2354  4              ! See MSCP document appendix "A-1 NOTE:" for more information on this topic.
;  2355  4              !-
;  2356  4
```

```
;  2357  4                selectoneu .RET_EN$AD [TYP$MSG] of  !Select the endpacket size
;  2358  4                    set
;  2359  4
;  2360  4                    [END$MSG] :                     !Is this an end code packet
;  2361  5                        begin
;  2362  5
;  2363  5                        !+
;  2364  5                        ! Select off of the endcode to make sure the communications
;  2365  5                        ! mechanism  transfered the correct number of byte for this
;  2366  5                        ! end  packet.  If  this  number of bytes transfered is not
;  2367  5                        ! correct  for  the commands end packet then load the error
;  2368  5                        ! code  into   return   status.  call decode to report  the
;  2369  5                        ! error and kick the bucket and die. This endcode is formed
;  2370  5                        ! by adding  the  end  message flag %o'200' to the commands
;  2371  5                        ! opcode.
;  2372  5                        !-
;  2373  5
;  2374  5                        selectoneu .RET_EN$AD [ENDCODE] of
;  2375  5                            set
;  2376  5                            !
;  2377  5                            ! "RECEIVE DATA" command end packet
;  2378  5                            !
;  2379  5
;  2380  5                            [EOP_RED] :
;  2381  6                                begin
;  2382  6
;  2383  6                                if .RET_EN$AD [MSG_LENGTH] gtru ESZ_RED     !Is the byte count correct
;  2384  6                                then
;  2385  7                                    begin
;  2386  7                                    RET_STATUS = MLE_CODE;  !Return a "message length error code"
;  2387  7                                    DECODE ();      !Report the error and kick the bucket
;  2388  7                                    return;         !Just for show.  Decode kills it
;  2389  6                                    end;
;  2390  6
;  2391  5                                end;
;  2392  5                            !
;  2393  5                            ! "SEND DATA" command end packet
;  2394  5                            ! "SET CONTROLLER CHAR" command end packet (same opcode)
;  2395  5                            !
;  2396  5
;  2397  5                            [EOP_SED] :
;  2398  5                            ! [EOP_SCC] :
;  2399  6                                begin
;  2400  6
;  2401  8                                if (not ((.RET_EN$AD [MSG_LENGTH] lequ ESZ_SED) or  !Is the byte count correct
;  2402  7                                        (.RET_EN$AD [MSG_LENGTH] equ ESZ_SCC)) )
;  2403  6                                then
;  2404  7                                    begin
;  2405  7                                    RET_STATUS = MLE_CODE;  !Return a "message length error code"
;  2406  7                                    DECODE ();      !Report the error and kick the bucket
;  2407  7                                    return;         !Just for show.  Decode kills it
;  2408  6                                    end;
;  2409  6
```

```
;  2410  5                                    end;
;  2411  5                                !
;  2412  5                                !" GET DUST STATUS"  command end packet
;  2413  5                                !
;  2414  5
;  2415  5                                [EOP_GDS] :
;  2416  6                                    begin
;  2417  6
;  2418  6                                    if .RET_EN$AD [MSG_LENGTH] nequ ESZ_GDS     !Is the byte count correct
;  2419  6                                    then
;  2420  7                                        begin
;  2421  7                                        RET_STATUS = MLE_CODE;  !Return a "message length error code"
;  2422  7                                        DECODE ();      !Report the error and kick the bucket
;  2423  7                                        return;         !Just for show.  Decode kills it
;  2424  6                                        end;
;  2425  6
;  2426  5                                    end;
;C 2427  5  %(                           !
;C 2428  5                               ! "EXECUTE SUPPLIED PROGRAM" command end packet
;C 2429  5                               !
;C 2430  5
;C 2431  5                               [EOP_ESP] :
;C 2432  5                                   begin
;C 2433  5
;C 2434  5                                   if .RET_EN$AD [MSG_LENGTH] nequ ESZ_ESP     !Is the byte count correct
;C 2435  5                                   then
;C 2436  5                                       begin
;C 2437  5                                       RET_STATUS = MLE_CODE;  !Return a "message length error code"
;C 2438  5                                       DECODE ();      !Report the error and kick the bucket
;C 2439  5                                       return;         !Just for show.  Decode kills it
;C 2440  5                                       end;
;C 2441  5
;C 2442  5                                   end;
;  2443  5  )%                           !
;  2444  5                               ! "EXECUTE LOCAL PROGRAM" command end packet
;  2445  5                               !
;  2446  5
;  2447  5                               [EOP_ELP] :
;  2448  6                                   begin
;  2449  6
;  2450  6                                   if .RET_EN$AD [MSG_LENGTH] nequ ESZ_ELP     !Is the byte count correct
;  2451  6                                   then
;  2452  7                                       begin
;  2453  7                                       RET_STATUS = MLE_CODE;  !Return a "message length error code"
;  2454  7                                       DECODE ();      !Report the error and kick the bucket
;  2455  7                                       return;         !Just for show.  Decode kills it
;  2456  6                                       end;
;  2457  6
;  2458  5                                   end;
;  2459  5                               !
;  2460  5                               ! "ABORT PROGRAM" command end packet
;  2461  5                               !
;  2462  5
```

```
;    2463  5                          [EOP_ABT] :
;    2464  6                              begin
;    2465  6
;    2466  6                              if .RET_EN$AD [MSG_LENGTH] nequ ESZ_ABT     !Is the byte count correct
;    2467  6                              then
;    2468  7                                  begin
;    2469  7                                  RET_STATUS = MLE_CODE;  !Return a "message length error code"
;    2470  7                                  DECODE ();      !Report the error and kick the bucket
;    2471  7                                  return;         !Just for show.  Decode kills it
;    2472  6                                  end;
;    2473  6
;    2474  5                              end;
; C  2475  5      %(                  !
; C  2476  5                          ! "ON LINE" command end packet
; C  2477  5                          !
; C  2478  5
; C  2479  5                          [EOP_ONL] :
; C  2480  5                              begin
; C  2481  5
; C  2482  5                              if .RET_EN$AD [MSG_LENGTH] nequ ESZ_ONL     !Is the byte count correct
; C  2483  5                              then
; C  2484  5                                  begin
; C  2485  5                                  RET_STATUS = MLE_CODE;  !Return a "message length error code"
; C  2486  5                                  DECODE ();      !Report the error and kick the bucket
; C  2487  5                                  return;         !Just for show.  Decode kills it
; C  2488  5                                  end;
; C  2489  5
; C  2490  5                              end;
;    2491  5      )%                  !
;    2492  5                          ! The  "OP_END"  end message flag all by its self tells
;    2493  5                          ! us that the controller is  flagging us  of an illegal
;    2494  5                          ! command sent over the connection.  Error and kick the
;    2495  5                          ! bucket.
;    2496  5                          !
;    2497  5
;    2498  5                          [OP_END] :
;    2499  6                              begin
;    2500  6                              RET_STATUS = IVC_CODE;
;    2501  6                              DECODE ();
;    2502  6                              return;
;    2503  5                              end;
;    2504  5                          !
;    2505  5                          ! The controller is telling us that a serious exception
;    2506  5                          ! has occured.  Error and kick the bucket.
;    2507  5                          !
;    2508  5
;    2509  5                          [EOP_SEX] :
;    2510  6                              begin
;    2511  6                              RET_STATUS = SEX_CODE;
;    2512  6                              DECODE ();
;    2513  6                              return;
;    2514  5                              end;
;    2515  5                          !
```

```
;  2516  5                        ! Unknown end packet endcode type
;  2517  5                        !
;  2518  5
;  2519  5                        [otherwise] :
;  2520  6                            begin
;  2521  6                            RET_STATUS = UEC_CODE;      !Return an "unknown end code"
;  2522  6                            DECODE ();          !Report the error and kick the bucket
;  2523  6                            return;             !Just for show.  Decode kills it
;  2524  5                            end;
;  2525  5                        tes;
;  2526  5
;  2527  5                      !
;  2528  5                      ! The port/controller sent the endpacket over the connection
;  2529  5                      ! with out any problems.  Now find this commands owner in the
;  2530  5                      ! out$standing buffer and indicate to them that the command
;  2531  5                      ! has been received.
;  2532  5                      !
;  2533  5                      REF_NUM = .RET_EN$AD [CMD_LREF];    !Get this rec packets cmd ref number
;  2534  5                      !
;  2535  5                      ! Search  the outstanding  command buffer for  this commands
;  2536  5                      ! reference number.
;  2537  5                      !
;  2538  5                      ! If  found,  load  the  buffer location with the ret_en$ad
;  2539  5                      ! and set the received  flag  to signify that  this command
;  2540  5                      ! has been received by this interrupt service routine.
;  2541  5                      !
;  2542  5                      FOUND_CMD = FALSE;          !Clear the found cmd flag
;  2543  5
;  2544  5                      incru i from 0 to REC_ALLOCATE - 1 do       !Search the buffer
;  2545  5
;  2546  5                          if .OUT$STD_BUF [.i, CMD_REF] eqlu .REF_NUM     !Is this the comd ref
;  2547  5                          then
;  2548  6                              begin
;  2549  6                              OUT$STD_BUF [.i, REC_FLG] = TRUE;   !Indicate command is received
;  2550  6                              OUT$STD_BUF [.i, ENV_ADR] = .RET_EN$AD;     !Return envelope adrs
;  2551  6                              FOUND_CMD = TRUE;   !Indicate it was found
;  2552  6                              exitloop;           !Exit the loop
;  2553  5                              end;
;  2554  5
;  2555  5                      !
;  2556  5                      ! If  the search through the  command ref
;  2557  5                      ! buffer failed to find this commands cmd
;  2558  5                      ! reference number then die.
;  2559  5                      !
;  2560  5
;  2561  5                      if not .FOUND_CMD
;  2562  5                      then
;  2563  6                          begin
;  2564  6                          RET_STATUS = PSE_CODE;
;  2565  6                          DECODE ();
;  2566  6                          return;
;  2567  5                          end;
;  2568  5
```

```
;  2569  4                        end;                            !End of END$MSG processing
;  2570  4                    !
;  2571  4                    ! The  set  controller  characteristics command
;  2572  4                    ! disabled the reporting of attentions messages
;  2573  4                    ! so  treat   this  as  a  fatal  error and die.
;  2574  4                    !
;  2575  4
;  2576  4                    [ATT$MSG] :                         !Attention end message type
;  2577  5                        begin
;  2578  5                        RET_STATUS = ATN_CODE;
;  2579  5                        DECODE ();
;  2580  5                        return;
;  2581  4                        end;
;  2582  4                    !
;  2583  4                    ! It doesn't make any since for this end message
;  2584  4                    ! packet not to have the end message flag added
;  2585  4                    ! to this command opcode as treat it as a fatal
;  2586  4                    ! error and die.
;  2587  4                    !
;  2588  4
;  2589  4                    [CMD$MSG] :
;  2590  5                        begin
;  2591  5                        RET_STATUS = CMD_CODE;
;  2592  5                        DECODE ();
;  2593  5                        return;
;  2594  4                        end;
;  2595  4                    !
;  2596  4                    ! This end code type is of unknown origin so
;  2597  4                    ! treat is as a fatal error and die.
;  2598  4                    !
;  2599  4
;  2600  4                    [otherwise] :
;  2601  5                        begin
;  2602  5                        RET_STATUS = UEC_CODE;      !Unknown message type code received
;  2603  5                        DECODE ();
;  2604  5                        return;
;  2605  4                        end;
;  2606  4                    tes;
;  2607  4
;  2608  4                !
;  2609  4                ! Before  we  leave  put this receive envelope message length
;  2610  4                ! field  back to the  envelope  size,  in bytes (Per UQ Spec).
;  2611  4                ! This  size does not include the 2 UQ's words preceeding the
;  2612  4                ! command text  area.
;  2613  4                !
;  2614  4                RET_EN$AD [MSG_LENGTH] = RB_SIZE*2;
;  2615  4                !
;  2616  4                ! Return  this receive slot descriptor back to
;  2617  4                ! the port to fullfill my part of the protocol.
;  2618  4                !
;  2619  4                RECEIVE_RING [.NRD_SLOT, OWN_BIT] = PORT_OWNED;
;  2620  4                !
;  2621  4                ! Look at the  next response ring descriptor.  If its
```

```
;  2622  4                ! host owned then continue this process else exit the
;  2623  4                ! loop.  First see if the ring reference has  wrapped
;  2624  4                ! around to the top of the ring.
;  2625  4                !
;  2626  4
;  2627  4                if .NRD_SLOT + 1 gtru REC_ALLOCATE - 1      !Has the ring ref wrapped around
;  2628  4                then
;  2629  4                    TEMP = ZERO                             !Wrap it back to zero dsc slot
;  2630  4                else
;  2631  4                    TEMP = .NRD_SLOT + 1;                   !Look at the next seq dsc slot
;  2632  4
;  2633  4                !
;  2634  4                ! Now see if the next receive descriptor slot is
;  2635  4                ! host owned.
;  2636  4                !
;  2637  4
;  2638  4                if .RECEIVE_RING [.TEMP, OWN_BIT] eqlu HOST_OWNED   !Are we done yet
;  2639  4                then
;  2640  4                    GET_NRD ()                              !Get the next resp desc to process
;  2641  4                else
;  2642  4                    exitloop;                               !No more to do so exit
;  2643  4
;  2644  3                end;                                        !End of WHILE LOOP
;  2645  3
;  2646  3            !
;  2647  3            ! All response ring descriptors have been with out any
;  2648  3            ! detected errors so return to the main host code with
;  2649  3            ! an pass return code.
;  2650  3            !
;  2651  3            return  RET_STATUS = PAS_CODE;                  !Return a "pass code
;  2652  2            end;
;  2653  2
;  2654  2        !**********************END OF RESPONSE RING INTERRUPT PROCESSING**********************
;  2655  2
;  2656  2        !+
;  2657  2        ! A send ring transition interrupt could happen if at
;  2658  2        ! some date only one descriptor slot is allocated for
;  2659  2        ! commands.
;  2660  2        !
;  2661  2        ! Clear the interrupt indicator if this is true and do
;  2662  2        ! a return with no errors.
;  2663  2        !-
;  2664  2
;  2665  2        if .HEAD_AREA [CMD_INT]                             !Is this a com ring transition interrupt
;  2666  2        then
;  2667  3            begin
;  2668  3            HEAD_AREA [CMD_INT] = ZERO;                     !Clear out the indicator
;  2669  3            return;                                         !Continue on with the host code
;  2670  2            end;
;  2671  2
;  2672  2        !**************************END OF COMMAND RING INTERRUPT PROCESSING**********************
;  2673  2
;  2674  2        !+
```

```
;  2675  2          ! Check  to see if an adaptor purge is being requested
;  2676  2          ! by the port/controller in order to complete excution
;  2677  2          ! of a issued command.  Remember that this is  illegal
;  2678  2          ! during PDP-11  formatting and is concidered  to be a
;  2679  2          ! fatal error.
;  2680  2          !-
;  2681  2
;  2682  2          if .HEAD_AREA [ADP_CH] nequ ZERO            !Is the an adaptor purge request?
;  2683  2          then
;  2684  3              begin
;  2685  3              RET_STATUS = APR_CODE;                  !Indicate the error code
;  2686  3              DECODE ();                              !Report the error and kick the bucket
;  2687  3              return;                                 !Just for show. Decode kills it
;  2688  2              end;
;  2689  2
;  2690  2          !*************************END OF ADAPTOR PURGE INTERRUPT PROCESSING************************
;  2691  2
;  2692  2          !+
;  2693  2          ! The host program has been interrupted by an unknown interrupt
;  2694  2          ! source if the routine program flow reaches here.
;  2695  2          !
;  2696  2          ! Load the error code into return status and call decode to take
;  2697  2          ! appropriate action.
;  2698  2          !-
;  2699  2
;  2700  2          RET_STATUS = UIN_CODE;
;  2701  2          DECODE ();
;  2702  2          return;
;  2703  1          end;


                                        .SBTTL  DUP$I.SERVICE GLOBAL ROUTINE DECLARATIONS
000000  010046                    DUP$I.SERVICE::
                                    MOV     R0,-(SP)                ;                       2198
000002  010146                      MOV     R1,-(SP)
000004  010246                      MOV     R2,-(SP)
000006  010346                      MOV     R3,-(SP)
000010  010446                      MOV     R4,-(SP)
000012  010546                      MOV     R5,-(SP)
000014  013700  000000G             MOV     RDRX.ADDR,R0            ;                       2269
000020  016046  000002              MOV     2(R0),-(SP)             ; *,RC$S.REG
000024  100004                      BPL     1$
000026  012737  000021  000000G     MOV     #21,RET.STATUS          ;                       2272
000034  000557                      BR      14$                     ;                       2273
000036  013700  000000G     1$:     MOV     HEAD.AREA,R0            ;                       2300
000042  032760  000001  000006      BIT     #1,6(R0)
000050  001002                      BNE     2$
000052  000137  002020'             JMP     26$
000056  005060  000006      2$:     CLR     6(R0)                   ;                       2303
000062  004737  000026'             JSR     PC,GET.NRD              ;                       2304
000066  013700  000000G             MOV     NRD.SLOT,R0             ;                       2312
000072  006300                      ASL     R0
000074  006300                      ASL     R0
```

```
000076  063700  000000G                        ADD     RECEIVE.RING,R0
000102  032760  100000  000002                BIT     #100000,2(R0)
000110  001404                                BEQ     4$
000112  012737  000061  000000G       3$:     MOV     #61,RET.STATUS          ;                       2315
000120  000577                                BR      21$                     ;                       2316
000122  104422                        4$:     TRAP    22                      ;                       2329
000124  013700  000000G                        MOV     NRD.SLOT,R0             ;                       2337
000130  006300                                ASL     R0
000132  006300                                ASL     R0
000134  063700  000000G                        ADD     RECEIVE.RING,R0
000140  011037  000000G                        MOV     (R0),RET.EN$AD
000144  162737  000004  000000G               SUB     #4,RET.EN$AD
000152  013701  000000G                        MOV     RET.EN$AD,R1            ;                       2357
000156  116100  000014                        MOVB    14(R1),R0
000162  006200                                ASR     R0
000164  006200                                ASR     R0
000166  006200                                ASR     R0
000170  006200                                ASR     R0
000172  006200                                ASR     R0
000174  006200                                ASR     R0
000176  042700  177774                        BIC     #177774,R0
000202  020027  000002                        CMP     R0,#2                   ;                       2360
000206  001130                                BNE     19$
000210  005002                                CLR     R2                      ;                       2374
000212  156102  000014                        BISB    14(R1),R2
000216  020227  000205                        CMP     R2,#205                 ;                       2380
000222  001007                                BNE     6$
000224  021127  000060                        CMP     (R1),#60                ;                       2383
000230  101462                                BLOS    15$
000232  012737  000071  000000G       5$:     MOV     #71,RET.STATUS          ;                       2386
000240  000527                                BR      21$                     ;                       2387
000242  020227  000204                6$:     CMP     R2,#204                 ;                       2397
000246  001007                                BNE     8$
000250  021127  000060                        CMP     (R1),#60                ;                       2401
000254  101450                                BLOS    15$
000256  021127  000034                        CMP     (R1),#34                ;                       2402
000262  001445                        7$:     BEQ     15$
000264  000762                                BR      5$                      ;                       2405
000266  020227  000201                8$:     CMP     R2,#201                 ;                       2415
000272  001003                                BNE     9$
000274  021127  000026                        CMP     (R1),#26                ;                       2418
000300  000770                                BR      7$
000302  020227  000203                9$:     CMP     R2,#203                 ;                       2447
000306  001003                                BNE     10$
000310  021127  000020                        CMP     (R1),#20                ;                       2450
000314  000762                                BR      7$
000316  020227  000206                10$:    CMP     R2,#206                 ;                       2463
000322  001003                                BNE     11$
000324  021127  000014                        CMP     (R1),#14                ;                       2466
000330  000754                                BR      7$
000332  020227  000200                11$:    CMP     R2,#200                 ;                       2498
000336  001004                                BNE     12$
000340  012737  000701  000000G               MOV     #701,RET.STATUS         ;                       2500
```

```
000346  000557                          BR      29$                             ;                               2501
000350  020227  000207          12$:    CMP     R2,#207                         ;                               2509
000354  001004                          BNE     13$
000356  012737  000601  000000G        MOV     #601,RET.STATUS                 ;                               2511
000364  000550                          BR      29$                             ;                               2512
000366  012737  000101  000000G  13$:    MOV     #101,RET.STATUS                 ;                               2521
000374  000544                  14$:    BR      29$                             ;                               2522
000376  013700  000000G          15$:    MOV     RET.EN$AD,R0                    ;                               2533
000402  016005  000004                  MOV     4(R0),R5                        ; *,REF.NUM
000406  005004                          CLR     R4                              ; FOUND.CMD                     2542
000410  005000                          CLR     R0                              ; I                             2544
000412  005001                  16$:    CLR     R1                              ;                               2546
000414  156001  000000G                BISB    OUT$STD.BUF(R0),R1              ; *(I),*
000420  020105                          CMP     R1,R5                           ; *,REF.NUM
000422  001011                          BNE     17$
000424  052760  100000  000000G        BIS     #100000,OUT$STD.BUF(R0)         ; *,*(I)                        2549
000432  013760  000000G  000002G        MOV     RET.EN$AD,OUT$STD.BUF+2(R0)     ; *,*(I)                        2550
000440  012704  000001                  MOV     #1,R4                           ; *,FOUND.CMD                   2551
000444  000405                          BR      18$                             ;                               2548
000446  062700  000004          17$:    ADD     #4,R0                           ; *,I                           2544
000452  020027  000014                  CMP     R0,#14                          ; I,*
000456  101755                          BLOS    16$
000460  032704  000001          18$:    BIT     #1,R4                           ; *,FOUND.CMD                   2561
000464  001016                          BNE     22$
000466  000611                          BR      3$                              ;                               2564
000470  020027  000001          19$:    CMP     R0,#1                           ;                               2576
000474  001004                          BNE     20$
000476  012737  000401  000000G        MOV     #401,RET.STATUS                 ;                               2578
000504  000500                          BR      29$                             ;                               2579
000506  005700                  20$:    TST     R0                              ;                               2589
000510  001326                          BNE     13$
000512  012737  000501  000000G        MOV     #501,RET.STATUS                 ;                               2591
000520  000472                  21$:    BR      29$                             ;                               2592
000522  012777  000074  000000G  22$:    MOV     #74,@RET.EN$AD                  ;                               2614
000530  013700  000000G                MOV     NRD.SLOT,R0                     ;                               2619
000534  006300                          ASL     R0
000536  006300                          ASL     R0
000540  063700  000000G                ADD     RECEIVE.RING,R0
000544  052760  100000  000002          BIS     #100000,2(R0)
000552  013700  000000G                MOV     NRD.SLOT,R0                     ;                               2627
000556  005200                          INC     R0
000560  020027  000003                  CMP     R0,#3
000564  101402                          BLOS    23$
000566  005003                          CLR     R3                              ; TEMP                          2629
000570  000401                          BR      24$                             ;                               2627
000572  010003                  23$:    MOV     R0,R3                           ; *,TEMP                        2631
000574  010300                  24$:    MOV     R3,R0                           ; TEMP,*                        2638
000576  006300                          ASL     R0
000600  006300                          ASL     R0
000602  063700  000000G                ADD     RECEIVE.RING,R0
000606  032760  100000  000002          BIT     #100000,2(R0)
000614  001004                          BNE     25$
000616  004737  000026'                JSR     PC,GET.NRD                      ;                               2640
```

```
000622  000137  001306'                        JMP     4$                              ;                       2638
000626  005037  000000G                25$:    CLR     RET.STATUS                      ;                       2651
000632  000427                          26$:    BR      30$                             ;                       2302
000634  013700  000000G                        MOV     HEAD.AREA,R0                    ;                       2665
000640  032760  000001  000004                BIT     #1,4(R0)
000646  001403                                BEQ     27$
000650  005060  000004                        CLR     4(R0)                           ;                       2668
000654  000416                                BR      30$                             ;                       2667
000656  013700  000000G                27$:    MOV     HEAD.AREA,R0                    ;                       2682
000662  105760  000003                        TSTB    3(R0)
000666  001404                                BEQ     28$
000670  012737  000201  000000G               MOV     #201,RET.STATUS                 ;                       2685
000676  000403                                BR      29$                             ;                       2686
000700  012737  000301  000000G        28$:    MOV     #301,RET.STATUS                 ;                       2700
000706  004737  000250'                29$:    JSR     PC,DECODE                       ;                       2701
000712  005726                          30$:    TST     (SP)+                           ;                       2198
000714  012605                                MOV     (SP)+,R5
000716  012604                                MOV     (SP)+,R4
000720  012603                                MOV     (SP)+,R3
000722  012602                                MOV     (SP)+,R2
000724  012601                                MOV     (SP)+,R1
000726  012600                                MOV     (SP)+,R0
000730  000002                                RTI

; Routine Size:  237 words,     Routine Base:  $CODE$ + 1164
; Maximum stack depth per invocation:  13 words


;   2704  1
```

```
;  2705  1     global routine CTO_WAIT (TO_VALUE, REF_NUM, BUF$LOC) =  !Controller time out wait
;  2706  1
;  2707  1     !++
;  2708  1     ! Functional Description :
;  2709  1     !       This routine is called to  wait  for  the port/controller to either
;  2710  1     !       complete  the  queued command  or time out the command.
;  2711  1     !
;  2712  1     ! Formal Parameters :
;  2713  1     !       TO_VALUE        Indicate the time-out interval for this command.
;  2714  1     !
;  2715  1     !       REF_NUM         This argument contains the unique reference number
;  2716  1     !                       assigned to this command being timed out by this
;  2717  1     !                       routine.
;  2718  1     !
;  2719  1     !       BUF$LOC         This argument points to the out$std_buf location
;  2720  1     !                       where this command is saved.  At this location the
;  2721  1     !                       received flag "rec_flg" bit is examined within the
;  2722  1     !                       the timeout loop and when it equals true will
;  2723  1     !                       signal that this command has been received by the
;  2724  1     !                       interrupt service routine.
;  2725  1     !
;  2726  1     ! Implicit Inputs :
;  2727  1     !       none
;  2728  1     !
;  2729  1     ! Implicit Outputs :
;  2730  1     !       The command  word in the out$std_buffer 'word zero of a command
;  2731  1     !       slot' is cleared out with the value %o'100000' to indicate this
;  2732  1     !       is an unused out$std_buffer  slot and  that  it  can  be reused.
;  2733  1     !
;  2734  1     ! Completion Codes :
;  2735  1     !    There are  two  levels of return   status  returned   by   this  routine.
;  2736  1     !       1.  The   DUP interrupt service returns to  this routine a status code
;  2737  1     !           to indicate the success of the connection/communications mechanism
;  2738  1     !           to complete  the  queued command.  If the port/controller does not
;  2739  1     !           time  out  then  this  return status  is returned as this routines
;  2740  1     !           return status code.
;  2741  1     !
;  2742  1     !       2.  If  the  port/controller times out  then the SA Register error bit
;  2743  1     !           is  examined  for the  error  bit set.  If set  then an port fatal
;  2744  1     !           error  code is  returned to the calling  routine else a controller
;  2745  1     !           time out error code is returned.
;  2746  1     !
;  2747  1     !       In  all  cases, if   an error  code is  returned  (bit 0 = 1) then the
;  2748  1     !       routine  decode  is called   to decode   the  error code  and does the
;  2749  1     !       necessary recovery actions.
;  2750  1     !
;  2751  1     !       At  the next higher level of return from this routine is another level
;  2752  1     !       of  return status    returned.  This   level test  the  success of the
;  2753  1     !       connection   and  also test the status  field in the returned response
;  2754  1     !       envelope for  the  success of  the controller to successfully complete
;  2755  1     !       the requested command.
;  2756  1     !
;  2757  1     ! Side Effects :
```

```
;  2758  1     !       none
;  2759  1     !--
;  2760  1
;  2761  2         begin
;  2762  2
;  2763  2         !+
;  2764  2         ! Before doing the timeout wait make sure that this buffer location
;  2765  2         ! that we're suppose to time out actually  contains the command ref
;  2766  2         ! number  that was sent to us via  the formal argument.  Error  and
;  2767  2         ! kick the bucket if not the same.
;  2768  2         !-
;  2769  2
;  2770  2         if .OUT$STD_BUF [.BUF$LOC, CMD_REF] nequ .REF_NUM   !Is this the same ref_num
;  2771  2         then
;  2772  3             begin
;  2773  3             RET_STATUS = OSE_CODE;                  !Indicate the error code
;  2774  3             DECODE ();                              !Call decode to report the error
;  2775  2             end;
;  2776  2
;  2777  2         !+
;  2778  2         ! Loop   on  a  one  micro  second  delay for  the  number of times
;  2779  2         ! requested  by    the   caller.  After  each  delay see  if the flag
;  2780  2         ! "rec_flg" has  been set yet.  Return  "Ret_status" and clear  the
;  2781  2         ! command  word  to %o'100000' to   indicate  this command has been
;  2782  2         ! received  if  this  flag  gets  set  before  the  timer   expires.
;  2783  2         ! Return a error code if the timer expires before the flag gets set.
;  2784  2         !-
;  2785  2
;  2786  2         incru i from 0 to .TO_VALUE do              !Loop for time-out_value
;  2787  3             begin
;  2788  3             DELAY (C_US);                           !Do the one micro second delay
;  2789  3             !
;  2790  3             ! Exit  routine  with  the  DUP  interrupt   service
;  2791  3             ! routines "ret_status" if "rec_flg"  got set before
;  2792  3             ! the timer expires.
;  2793  3             !
;  2794  3
;  2795  3             if .OUT$STD_BUF [.BUF$LOC, REC_FLG] eqlu TRUE   !Is this command received yet
;  2796  3             then
;  2797  4                 begin
;  2798  4                 OUT$STD_BUF [.BUF$LOC, CMD_WRD] = %o'100000';       !Return the slot to the unused state
;  2799  4                 return .RET_STATUS;                 !Return the interrupt service status
;  2800  3                 end;
;  2801  3
;  2802  3             BREAK;                                  !Service any control C's
;  2803  2             end;
;  2804  2
;  2805  2     !
;  2806  2     ! The port/controller timed out if the code
;  2807  2     ! reached  here.  Return  an  error code to
;  2808  2     ! the caller and exit the routine.
;  2809  2     !
;  2810  2
```

```
;    2811  2          if .RDRX_ADDR [RCSA, ERR_BIT]                !Is the SA error bit set
;    2812  2          then
;    2813  2              return RET_STATUS = PFE_CODE             !Port timed out with fatal error
;    2814  2          else
;    2815  2              return RET_STATUS = CTO_CODE;            !Port just timed out
;    2816  2
;    2817  1          end;
```

```
                                .GLOBL  L$CPU, L$DLY

                                .SBTTL  CTO.WAIT GLOBAL ROUTINE DECLARATIONS
000000  004137  000000G    CTO.WAIT::
                                JSR     R1,$SAVE3                       ;                               2705
000004  024646                  CMP     -(SP),-(SP)
000006  016600  000016          MOV     16(SP),R0                       ; BUF$LOC,*                     2770
000012  006300                  ASL     R0
000014  006300                  ASL     R0
000016  012703  000000G         MOV     #OUT$STD.BUF,R3
000022  060003                  ADD     R0,R3
000024  005000                  CLR     R0
000026  151300                  BISB    (R3),R0
000030  020066  000020          CMP     R0,20(SP)                       ; *,REF.NUM
000034  001405                  BEQ     1$
000036  012737  003001  000000G MOV     #3001,RET.STATUS                ;                               2773
000044  004737  000250'        JSR     PC,DECODE                       ;                               2774
000050  005002          1$:     CLR     R2                              ; I                             2786
000052  000425                  BR      8$
000054  013701  000000G 2$:     MOV     L$CPU,R1                        ; *,$$TMP2                      2788
000060  001411          3$:     BEQ     6$
000062  013700  000000G         MOV     L$DLY,R0                        ; *,$$TMP1
000066  001404                  BEQ     5$
000070  005066  000002  4$:     CLR     2(SP)                           ; $$TMP
000074  005300                  DEC     R0                              ; $$TMP1
000076  001374                  BNE     4$
000100  005301          5$:     DEC     R1                              ; $$TMP2
000102  000766                  BR      3$
000104  005713          6$:     TST     (R3)                            ;                               2795
000106  100005                  BPL     7$
000110  012713  100000          MOV     #-100000,(R3)                   ;                               2798
000114  013700  000000G         MOV     RET.STATUS,R0                   ;                               2797
000120  000421                  BR      11$
000122  104422          7$:     TRAP    22                              ;                               2800
000124  005202                  INC     R2                              ; I                             2786
000126  020266  000022  8$:     CMP     R2,22(SP)                       ; I,TO.VALUE
000132  101750                  BLOS    2$
000134  013700  000000G         MOV     RDRX.ADDR,R0                    ;                               2811
000140  016016  000002          MOV     2(R0),(SP)                      ; *,RC$S.REG
000144  100003                  BPL     9$
000146  012700  000021          MOV     #21,R0                          ;                               2813
000152  000402                  BR      10$
000154  012700  000011  9$:     MOV     #11,R0                          ;                               2815
```

```
000160  010037  000000G                   10$:    MOV     R0,RET.STATUS
000164  022626                            11$:    CMP     (SP)+,(SP)+                     ;                       2705
000166  000207                                    RTS     PC

; Routine Size:  60 words,    Routine Base:  $CODE$ + 2116
; Maximum stack depth per invocation:  8 words


;  2818  1
```

```
;  2819  1     global routine ABORT =          !Aborts remote program
;  2820  1
;  2821  1     !++
;  2822  1     ! Functional Description :
;  2823  1     !       The  abort program  command is used to terminate the execution of a
;  2824  1     !       remote  program in an orderly  fashion.  When a successful response
;  2825  1     !       is   received to   this command the remote  program  has  stopped
;  2826  1     !       executing  and the  server is in idle state.  Note that the sending
;  2827  1     !       of this command does not preclude further send data or receive data
;  2828  1     !       exchanges: On the  contrary, the  remote program may be designed to
;  2829  1     !       send  out termination status and possibly even ask questions during
;  2830  1     !       its forced-exit sequence.  The time out for this command is a fixed
;  2831  1     !       10  seconds  and  if    a   response  is  not  received by then the
;  2832  1     !       connection to  the dust should be terminated.  This command is only
;  2833  1     !       legal if the dust is in active state.
;  2834  1     !
;  2835  1     ! Formal Parameters :
;  2836  1     !       none
;  2837  1     !
;  2838  1     ! Implicit Inputs :
;  2839  1     !       NSD_SLOT        This  global storage gets  loaded  by  the routine
;  2840  1     !                       'Get_nsd'  and  in it is stored the next send ring
;  2841  1     !                       descriptor  slot  where the port/controller should
;  2842  1     !                       be polling on and the  place  to put this commands
;  2843  1     !                       command packet.
;  2844  1     !
;  2845  1     ! Implicit Outputs :
;  2846  1     !       none
;  2847  1     !
;  2848  1     ! Completion Codes :
;  2849  1     !       RET_STATUS:     Return status passes back to the calling routine
;  2850  1     !                       the status of the just issued command.
;  2851  1     !
;  2852  1     ! Side Effects :
;  2853  1     !       Any remote program running in the controllers DM machine will
;  2854  1     !       be aborted.
;  2855  1     !--
;  2856  1
;  2857  2         begin
;  2858  2
;  2859  2         local
;  2860  2             REF_NUM,                                    !Stores unique cmd ref number
;  2861  2             ABO_BUF$LOC,                                !Stores out$standing cmd buffer location
;  2862  2             TEMP;                                       !A place to put the read IP register data
;  2863  2
;  2864  2         !
;  2865  2         ! Before we load up the command packet up
;  2866  2         ! with  all this good information get the
;  2867  2         ! next send  descriptor slot and a unique
;  2868  2         ! command reference number.
;  2869  2         !
;  2870  2         GET_NSD ();                                     !Get the next send desc slot
;  2871  2         REF_NUM = GET_CMD$REF ();                       !Get a unique command ref num
```

```
;  2872  2          !
;  2873  2          ! UQ Port command envelope Header field definition
;  2874  2          !
;  2875  2          SND_ENVELOPE [.NSD_SLOT, MSG_LENGTH] = SZ_ABT;      !Load the length of envelope
;  2876  2          SND_ENVELOPE [.NSD_SLOT, CREDITS] = ONE;    !Load credits
;  2877  2          SND_ENVELOPE [.NSD_SLOT, MSG_TYPE] = 0;     !Load message type
;  2878  2          SND_ENVELOPE [.NSD_SLOT, CONN_ID] = 2;      !Load connection ID
;  2879  2          !
;  2880  2          ! DUP command envelope field definition
;  2881  2          !
;  2882  2          SND_ENVELOPE [.NSD_SLOT, CMD_LREF] = .REF_NUM;      !Define referance number
;  2883  2          SND_ENVELOPE [.NSD_SLOT, CMD_HREF] = ZERO;  !Hi order ref number
;  2884  2          SND_ENVELOPE [.NSD_SLOT, UN_LUSED] = ZERO;  !Unused low order
;  2885  2          SND_ENVELOPE [.NSD_SLOT, UN_HUSED] = ZERO;  !Unused hi order
;  2886  2          SND_ENVELOPE [.NSD_SLOT, OPCODE] = OP_ABT;  !Load opcod
;  2887  2          SND_ENVELOPE [.NSD_SLOT, RSVD] = ZERO;      !Reserved field
;  2888  2          SND_ENVELOPE [.NSD_SLOT, MODIFIER] = ZERO;
;  2889  2          !
;  2890  2          ! Call   the  load  out$standing  command buffer routine
;  2891  2          ! and  load  this command  into  the buffer.  The return
;  2892  2          ! from this routine will point us to the buffer location
;  2893  2          ! where  this command  is  stored.  Later we can look at
;  2894  2          ! this  location to see if the interrupt service routine
;  2895  2          ! has received and process it.
;  2896  2          !
;  2897  2          ABO_BUF$LOC = LOAD_OUT$STD_BUF (.REF_NUM);  !Load the command
;  2898  2
;  2899  2          if .ABO_BUF$LOC eqlu OBF_CODE then DECODE ();       !Error if buffer is full
;  2900  2
;  2901  2          !
;  2902  2          ! Set the ownership bit to 1 giving this slot
;  2903  2          ! to the port/controller
;  2904  2          !
;  2905  2          SEND_RING [.NSD_SLOT, OWN_BIT] = PORT_OWNED;
;  2906  2          !
;  2907  2          ! Read the IP register to stimulate port polling
;  2908  2          !
;  2909  2          TEMP = .RDRX_ADDR [RCIP, RC_ALL];
;  2910  2          !
;  2911  2          ! Time  out the  port/controller processing  the command.
;  2912  2          !
;  2913  2          ! The   first   test   tests the connections  ability to
;  2914  2          ! respond to  this command without  any errors in the SA
;  2915  2          ! register and for the command not timing out.
;  2916  2          !
;  2917  2          ! The  second tests the  DUP server for good status.  If
;  2918  2          ! bad status is sent back then an error code is returned
;  2919  2          ! to the calling routine where the routine "decode" will
;  2920  2          ! decode  and take the  appropriate recovery.  The  time
;  2921  2          ! out  routine will loop on delaying and checking the hi
;  2922  2          ! bit of the  first word in  the  out$std_buf for a true.
;  2923  2          ! When true signals us that the interrupt service routine
;  2924  2          ! has received  the endpacket  and  no connection errors
```

```
;  2925  2          ! were detected.
;  2926  2          !
;  2927  2
;  2928  2          if CTO_WAIT (3000, .REF_NUM, .ABO_BUF$LOC) then DECODE ();  !Is return an error
;  2929  2
;  2930  2          !
;  2931  2          ! Get  the return envelope  address  from the  out$std_buf
;  2932  2          ! at  this commands  buffer location  and check the packet
;  2933  2          ! for good status error and die if bad status was returned
;  2934  2          !
;  2935  2          RET_EN$AD = .OUT$STD_BUF [.ABO_BUF$LOC, ENV_ADR];   !Get the ret env adr
;  2936  2          !
;  2937  2          ! Now test for good status
;  2938  2          !
;  2939  2
;  2940  2          if .RET_EN$AD [STATUS] nequ ZERO            !Test the status
;  2941  2          then
;  2942  2              return RET_STATUS = RSE_CODE            !Return a "Response status err" code
;  2943  2          else
;  2944  2              return .RET_STATUS;                     !This ret_status is good or bad
;  2945  2
;  2946  1          end;


                                .SBTTL  ABORT GLOBAL ROUTINE DECLARATIONS
000000  004137  000000G     ABORT:: JSR     R1,$SAVE2                       ;                       2819
000004  005746                     TST     -(SP)
000006  004737  000000'             JSR     PC,GET.NSD                      ;                       2870
000012  004737  000146'             JSR     PC,GET.CMD$REF                  ;                       2871
000016  010002                     MOV     R0,R2                           ; *,REF.NUM
000020  013746  000000G             MOV     NSD.SLOT,-(SP)                  ;                       2875
000024  012746  000054             MOV     #54,-(SP)
000030  004737  000000G             JSR     PC,BL$MUL
000034  012760  000014  000000G     MOV     #14,SND.ENVELOPE(R0)
000042  012701  000002G             MOV     #SND.ENVELOPE+2,R1              ;                       2876
000046  060001                     ADD     R0,R1
000050  112711  000001             MOVB    #1,(R1)                         ;                       2877
000054  112761  000002  000001     MOVB    #2,1(R1)                        ;                       2878
000062  010260  000004G             MOV     R2,SND.ENVELOPE+4(R0)           ; REF.NUM,*             2882
000066  005060  000006G             CLR     SND.ENVELOPE+6(R0)              ;                       2883
000072  005060  000010G             CLR     SND.ENVELOPE+10(R0)             ;                       2884
000076  005060  000012G             CLR     SND.ENVELOPE+12(R0)             ;                       2885
000102  112760  000006  000014G     MOVB    #6,SND.ENVELOPE+14(R0)          ;                       2886
000110  105060  000015G             CLRB    SND.ENVELOPE+15(R0)             ;                       2887
000114  005060  000016G             CLR     SND.ENVELOPE+16(R0)             ;                       2888
000120  010216                     MOV     R2,(SP)                         ; REF.NUM,*             2897
000122  004737  000054'             JSR     PC,LOAD.OUT$STD.BUF
000126  010001                     MOV     R0,R1                           ; *,ABO.BUF$LOC
000130  020127  002001             CMP     R1,#2001                        ; ABO.BUF$LOC,*         2899
000134  001002                     BNE     1$
000136  004737  000250'             JSR     PC,DECODE
000142  013700  000000G     1$:     MOV     NSD.SLOT,R0                     ;                       2905
000146  006300                     ASL     R0
```

```
000150  006300                          ASL     R0
000152  063700  000000G                 ADD     SEND.RING,R0
000156  052760  100000  000002          BIS     #100000,2(R0)
000164  017766  000000G 000004          MOV     @RDRX.ADDR,4(SP)                ; *,RC$S.REG                    2909
000172  016600  000004                  MOV     4(SP),R0                        ; RC$S.REG,TEMP
000176  012716  005670                  MOV     #5670,(SP)                      ;                               2928
000202  010246                          MOV     R2,-(SP)                        ; REF.NUM,*
000204  010146                          MOV     R1,-(SP)                        ; ABO.BUF$LOC,*
000206  004737  002116'                 JSR     PC,CTO.WAIT
000212  022626                          CMP     (SP)+,(SP)+
000214  006000                          ROR     R0
000216  103002                          BCC     2$
000220  004737  000250'                 JSR     PC,DECODE
000224  010100                  2$:     MOV     R1,R0                           ; ABO.BUF$LOC,*                 2935
000226  006300                          ASL     R0
000230  006300                          ASL     R0
000232  016037  000002G 000000G         MOV     OUT$STD.BUF+2(R0),RET.EN$AD
000240  016000  000002G                 MOV     OUT$STD.BUF+2(R0),R0            ;                               2940
000244  005760  000016                  TST     16(R0)
000250  001405                          BEQ     3$
000252  012700  000031                  MOV     #31,R0                          ;                               2942
000256  010037  000000G                 MOV     R0,RET.STATUS
000262  000402                          BR      4$                              ;                               2944
000264  013700  000000G          3$:    MOV     RET.STATUS,R0
000270  062706  000006          4$:     ADD     #6,SP                           ;                               2819
000274  000207                          RTS     PC

; Routine Size:  95 words,    Routine Base:  $CODE$ + 2306
; Maximum stack depth per invocation:  9 words


;  2947  1
```

```
;  2948  1     global routine GET_DUST_STATUS =        !Gets DUP server status
;  2949  1
;  2950  1     !++
;  2951  1     ! Functional Description :
;  2952  1     !       This  command allows  the host program to interrogate the DUP server
;  2953  1     !       to  determine its  caracteristics, its  state  and the  state of the
;  2954  1     !       program  currently running  (if any).  It is legal in either idle or
;  2955  1     !       active  state  and  does  not affect  the state of server.  It has a
;  2956  1     !       fixed timeout interval of 3 seconds.  If the response times out, the
;  2957  1     !       host should break the connection.
;  2958  1     !
;  2959  1     ! Formal Parameters :
;  2960  1     !       none
;  2961  1     !
;  2962  1     ! Implicit Inputs :
;  2963  1     !       NSD_SLOT        This  global storage gets  loaded  by  the routine
;  2964  1     !                       'Get_nsd'  and  in it is stored the next send ring
;  2965  1     !                       descriptor  slot  where the port/controller should
;  2966  1     !                       be polling on and the  place  to put this commands
;  2967  1     !                       command packet.
;  2968  1     !
;  2969  1     ! Implicit Outputs :
;  2970  1     !       none
;  2971  1     !
;  2972  1     ! Completion Codes :
;  2973  1     !       RET_STATUS:     Return status passes back to the calling routine
;  2974  1     !                       the status of the just issued command.
;  2975  1     !
;  2976  1     ! Side Effects :
;  2977  1     !--
;  2978  1
;  2979  2         begin
;  2980  2
;  2981  2     local
;  2982  2         REF_NUM,                                    !Stores unique cmd ref number
;  2983  2         GDS_BUF$LOC,                                !Stores out$standing cmd buffer location
;  2984  2         TEMP;                                       !A place to put the IP read data
;  2985  2
;  2986  2     !
;  2987  2     ! Before we load up the command packet up
;  2988  2     ! with  all this good information get the
;  2989  2     ! next send  descriptor slot and a unique
;  2990  2     ! command reference number.
;  2991  2     !
;  2992  2     GET_NSD ();                                     !Get the next send desc slot
;  2993  2     REF_NUM = GET_CMD$REF ();                       !Get a unique command ref num
;  2994  2     !
;  2995  2     ! UQ Port command envelope Header field definition
;  2996  2     !
;  2997  2     SND_ENVELOPE [.NSD_SLOT, MSG_LENGTH] = SZ_GDS;      !Load the envelope size
;  2998  2     SND_ENVELOPE [.NSD_SLOT, CREDITS] = ONE;        !Load the credit size
;  2999  2     SND_ENVELOPE [.NSD_SLOT, MSG_TYPE] = 0;         !Load the message type (Sequential)
;  3000  2     SND_ENVELOPE [.NSD_SLOT, CONN_ID] = 2;          !Load the connection ID (DUP)
```

```
;    3001  2          !
;    3002  2          ! DUP generic command envelope field definition
;    3003  2          !
;    3004  2          SND_ENVELOPE [.NSD_SLOT, CMD_LREF] = .REF_NUM;      !Load command reference number
;    3005  2          SND_ENVELOPE [.NSD_SLOT, CMD_HREF] = ZERO;  !Command reference low order
;    3006  2          SND_ENVELOPE [.NSD_SLOT, UN_LUSED] = ZERO;  !Low order unused
;    3007  2          SND_ENVELOPE [.NSD_SLOT, UN_HUSED] = ZERO;  !Hi order unused
;    3008  2          SND_ENVELOPE [.NSD_SLOT, OPCODE] = OP_GDS;  !Load opcode
;    3009  2          SND_ENVELOPE [.NSD_SLOT, RSVD] = ZERO;      !Reserved field
;    3010  2          SND_ENVELOPE [.NSD_SLOT, MODIFIER] = ZERO;  !Load modifier field
;    3011  2          !
;    3012  2          ! Call   the  load  out$standing  command buffer routine
;    3013  2          ! and  load  this command  into  the buffer.  The return
;    3014  2          ! from this routine will point us to the buffer location
;    3015  2          ! where  this command  is  stored.  Later we can look at
;    3016  2          ! this  location to see if the interrupt service routine
;    3017  2          ! has received and process it.
;    3018  2          !
;    3019  2          GDS_BUF$LOC = LOAD_OUT$STD_BUF (.REF_NUM);  !Load the command
;    3020  2
;    3021  2          if .GDS_BUF$LOC eqlu OBF_CODE then DECODE ();       !Error if buffer is full
;    3022  2
;    3023  2          !
;    3024  2          ! Set the ownership bit to 1 giving this slot
;    3025  2          ! to the port/controller
;    3026  2          !
;    3027  2          SEND_RING [.NSD_SLOT, OWN_BIT] = PORT_OWNED;
;    3028  2          !
;    3029  2          ! Read the IP register to stimulate port polling
;    3030  2          !
;    3031  2          TEMP = .RDRX_ADDR [RCIP, RC_ALL];
;    3032  2          !
;    3033  2          ! Time  out the  port/controller processing  the command.
;    3034  2          !
;    3035  2          ! The   first   test   tests the connections  ability to
;    3036  2          ! respond to  this command without  any errors in the SA
;    3037  2          ! register and for the command not timing out.
;    3038  2          !
;    3039  2          ! The  second tests the  DUP server for good status.  If
;    3040  2          ! bad status is sent back then an error code is returned
;    3041  2          ! to the calling routine where the routine "decode" will
;    3042  2          ! decode  and take the  appropriate recovery.  The  time
;    3043  2          ! out  routine will loop on delaying and checking the hi
;    3044  2          ! bit of the  first word in  the  out$std_buf for a true.
;    3045  2          ! When true signals us that the interrupt service routine
;    3046  2          ! has received  the endpacket  and  no connection errors
;    3047  2          ! were detected.
;    3048  2          !
;    3049  2
;    3050  2          if CTO_WAIT (%o'3000', .REF_NUM, .GDS_BUF$LOC) then DECODE ();      !Is return an error
;    3051  2
;    3052  2          !
;    3053  2          ! Get  the return envelope  address  from the  out$std_buf
```

```
;    3054  2          ! at  this commands  buffer location  and check the packet
;    3055  2          ! for good status error and die if bad status was returned
;    3056  2          !
;    3057  2          RET_EN$AD = .OUT$STD_BUF [.GDS_BUF$LOC, ENV_ADR];   !Get the ret env adr
;    3058  2          !
;    3059  2          ! Now test for good status
;    3060  2          !
;    3061  2
;    3062  2          if .RET_EN$AD [STATUS] nequ ZERO            !Test the status
;    3063  2          then
;    3064  2              return RET_STATUS = RSE_CODE            !Return a "Response status err" code
;    3065  2          else
;    3066  2              return .RET_STATUS;                     !This ret_status is good or bad
;    3067  2
;    3068  1          end;


                                        .SBTTL  GET.DUST.STATUS GLOBAL ROUTINE DECLARATIONS
000000  004137  000000G          GET.DUST.STATUS::
                                        JSR     R1,$SAVE2                       ;                       2948
000004  005746                          TST     -(SP)
000006  004737  000000'                 JSR     PC,GET.NSD                      ;                       2992
000012  004737  000146'                 JSR     PC,GET.CMD$REF                  ;                       2993
000016  010002                          MOV     R0,R2                           ; *,REF.NUM
000020  013746  000000G                 MOV     NSD.SLOT,-(SP)                  ;                       2997
000024  012746  000054                  MOV     #54,-(SP)
000030  004737  000000G                 JSR     PC,BL$MUL
000034  012760  000014  000000G         MOV     #14,SND.ENVELOPE(R0)
000042  012701  000002G                 MOV     #SND.ENVELOPE+2,R1              ;                       2998
000046  060001                          ADD     R0,R1
000050  112711  000001                  MOVB    #1,(R1)                         ;                       2999
000054  112761  000002  000001         MOVB    #2,1(R1)                        ;                       3000
000062  010260  000004G                 MOV     R2,SND.ENVELOPE+4(R0)           ; REF.NUM,*             3004
000066  005060  000006G                 CLR     SND.ENVELOPE+6(R0)              ;                       3005
000072  005060  000010G                 CLR     SND.ENVELOPE+10(R0)             ;                       3006
000076  005060  000012G                 CLR     SND.ENVELOPE+12(R0)             ;                       3007
000102  112760  000001  000014G         MOVB    #1,SND.ENVELOPE+14(R0)          ;                       3008
000110  105060  000015G                 CLRB    SND.ENVELOPE+15(R0)             ;                       3009
000114  005060  000016G                 CLR     SND.ENVELOPE+16(R0)             ;                       3010
000120  010216                          MOV     R2,(SP)                         ; REF.NUM,*             3019
000122  004737  000054'                 JSR     PC,LOAD.OUT$STD.BUF
000126  010001                          MOV     R0,R1                           ; *,GDS.BUF$LOC
000130  020127  002001                  CMP     R1,#2001                        ; GDS.BUF$LOC,*         3021
000134  001002                          BNE     1$
000136  004737  000250'                 JSR     PC,DECODE
000142  013700  000000G          1$:    MOV     NSD.SLOT,R0                     ;                       3027
000146  006300                          ASL     R0
000150  006300                          ASL     R0
000152  063700  000000G                 ADD     SEND.RING,R0
000156  052760  100000  000002         BIS     #100000,2(R0)
000164  017766  000000G 000004         MOV     @RDRX.ADDR,4(SP)                ; *,RC$S.REG            3031
000172  016600  000004                  MOV     4(SP),R0                        ; RC$S.REG,TEMP
000176  012716  003000                  MOV     #3000,(SP)                      ;                       3050
```

```
000202  010246                          MOV     R2,-(SP)                        ; REF.NUM,*
000204  010146                          MOV     R1,-(SP)                        ; GDS.BUF$LOC,*
000206  004737  002116'                 JSR     PC,CTO.WAIT
000212  022626                          CMP     (SP)+,(SP)+
000214  006000                          ROR     R0
000216  103002                          BCC     2$
000220  004737  000250'                 JSR     PC,DECODE
000224  010100                  2$:     MOV     R1,R0                           ; GDS.BUF$LOC,*                 3057
000226  006300                          ASL     R0
000230  006300                          ASL     R0
000232  016037  000002G 000000G         MOV     OUT$STD.BUF+2(R0),RET.EN$AD
000240  016000  000002G                 MOV     OUT$STD.BUF+2(R0),R0            ;                               3062
000244  005760  000016                  TST     16(R0)
000250  001405                          BEQ     3$
000252  012700  000031                  MOV     #31,R0                          ;                               3064
000256  010037  000000G                 MOV     R0,RET.STATUS
000262  000402                          BR      4$                              ;                               3066
000264  013700  000000G          3$:    MOV     RET.STATUS,R0
000270  062706  000006           4$:    ADD     #6,SP                           ;                               2948
000274  000207                          RTS     PC

; Routine Size:  95 words,      Routine Base:  $CODE$ + 2604
; Maximum stack depth per invocation:  9 words


;   3069  1
```

```
; C 3070  1     %(
; C 3071  1     global routine EX_SUP_PROG =    !Executes supplied program
; C 3072  1
; C 3073  1     !++
; C 3074  1     ! Functional Description :
; C 3075  1     !       This  command  causes  the  server  to  transfer the program from host
; C 3076  1     !       memory  to  an  area  in  the controller and start its execution.  The
; C 3077  1     !       host  supplies  the  address  and  length  (in  bytes) of a buffer
; C 3078  1     !       containing  the  program  header  and  initial  load;  the starting
; C 3079  1     !       of the program, its memory requirements and any relocation information
; C 3080  1     !       needed  to  run under the server are in the program header in a format
; C 3081  1     !       which  is none of the host business.  This commmand is only legal when
; C 3082  1     !       the  server is in the idle state and return of a successful end packet
; C 3083  1     !       puts the server into to active state.
; C 3084  1     !
; C 3085  1     !       The time out interval for this command is 30 seconds.
; C 3086  1     !
; C 3087  1     ! Formal Parameters :
; C 3088  1     !       none
; C 3089  1     !
; C 3090  1     ! Implicit Inputs :
; C 3091  1     !       NSD_SLOT        This  global storage gets  loaded  by  the routine
; C 3092  1     !                       'Get_nsd'  and  in it is stored the next send ring
; C 3093  1     !                       descriptor  slot  where the port/controller should
; C 3094  1     !                       be polling on and the  place  to put this commands
; C 3095  1     !                       command packet.
; C 3096  1     !
; C 3097  1     !
; C 3098  1     ! Implicit Outputs :
; C 3099  1     !       AZFMTR:         Azfmtr is the vector produced by DMCONV program and
; C 3100  1     !                       is decalared in module AZKEL6.
; C 3101  1     !       DMSA:           These three bound addresses point to specific area
; C 3102  1     !       OVSA:           in the DM code buffer 'azfmtr' and are used to
; C 3103  1     !       HDSA:           define the buffer descriptors within this command.
; C 3104  1     !
; C 3105  1     ! Completion Codes :
; C 3106  1     !       RET_STATUS:     Return status passes back to the calling routine
; C 3107  1     !                       the status of the just issued command.
; C 3108  1     !
; C 3109  1     ! Side Effects :
; C 3110  1     !       The DM machine in the controller goes from the idle state to the
; C 3111  1     !       active state on return of a successful return packet.
; C 3112  1     !--
; C 3113  1
; C 3114  1         begin
; C 3115  1
; C 3116  1         local
; C 3117  1             REF_NUM,                                !Stores unique cmd ref number
; C 3118  1             ESP_BUF$LOC,                            !Stores out$standing cmd buffer location
; C 3119  1             TEMP;                                   !A place to put the read IP register data
; C 3120  1
; C 3121  1         !
; C 3122  1         ! Before we load up the command packet up
```

```
; C 3123  1          ! with  all this good information get the
; C 3124  1          ! next send  descriptor slot and a unique
; C 3125  1          ! command reference number.
; C 3126  1          !
; C 3127  1          GET_NSD ();                                 !Get the next send desc slot
; C 3128  1          REF_NUM = GET_CMD$REF ();                   !Get a unique command ref num
; C 3129  1          !
; C 3130  1          ! UQ Port command envelope Header field definition
; C 3131  1          !
; C 3132  1          SND_ENVELOPE [.NSD_SLOT, MSG_LENGTH] = SZ_ESP;      !Load the message length
; C 3133  1          SND_ENVELOPE [.NSD_SLOT, CREDITS] = ONE;    !Load the credits field
; C 3134  1          SND_ENVELOPE [.NSD_SLOT, MSG_TYPE] = 0;     !Define the msg type 'Sequential'
; C 3135  1          SND_ENVELOPE [.NSD_SLOT, CONN_ID] = 2;      !Define the conection ID as DUP
; C 3136  1          !
; C 3137  1          ! DUP generic command envelope field definition
; C 3138  1          !
; C 3139  1          SND_ENVELOPE [.NSD_SLOT, CMD_LREF] = .REF_NUM;      !Load command ref number
; C 3140  1          SND_ENVELOPE [.NSD_SLOT, CMD_HREF] = ZERO;  !Zero Hi order word of cmd ref
; C 3141  1          SND_ENVELOPE [.NSD_SLOT, UN_LUSED] = ZERO;  !Not used in DUP implimentation
; C 3142  1          SND_ENVELOPE [.NSD_SLOT, UN_HUSED] = ZERO;  !Not used in DUP implimentation
; C 3143  1          SND_ENVELOPE [.NSD_SLOT, OPCODE] = OP_ESP;  !Load the command op-code
; C 3144  1          SND_ENVELOPE [.NSD_SLOT, RSVD] = ZERO;      !Not used
; C 3145  1          SND_ENVELOPE [.NSD_SLOT, MODIFIER] = ZERO;
; C 3146  1          !
; C 3147  1          ! Command specfic command envelope field definition
; C 3148  1          !
; C 3149  1          !
; C 3150  1          ! Byte count of initial transfer (from bytes 0-3
; C 3151  1          ! of the program header).
; C 3152  1          !
; C 3153  1          SND_ENVELOPE [.NSD_SLOT, BLO_CNT] = .AZFMTR [WRD0]; !Byte count low word
; C 3154  1          SND_ENVELOPE [.NSD_SLOT, BHI_CNT] = .AZFMTR [WRD1]; !Byte count high word
; C 3155  1          !
; C 3156  1          ! Buffer descriptor definition for initial load.  First
; C 3157  1          ! byte  of  this  buffer  is byte 0 of  program header.
; C 3158  1          !
; C 3159  1          SND_ENVELOPE [.NSD_SLOT, BPA_LO] = HDSA;    !Low unibus adrs <0-15>
; C 3160  1          SND_ENVELOPE [.NSD_SLOT, BPA_HI] = ZERO;    !Unibus adrs bits <16-17>
; C 3161  1          SND_ENVELOPE [.NSD_SLOT, QBUS_EXT] = ZERO;  !Q_bus extention adrs
; C 3162  1          SND_ENVELOPE [.NSD_SLOT, RSV] = ZERO;       !Reserved field
; C 3163  1          SND_ENVELOPE [.NSD_SLOT, UBA_CHAN] = ZERO;  !Unibus adaptor channel number
; C 3164  1          SND_ENVELOPE [.NSD_SLOT, RSV0] = ZERO;      !These next four words are not
; C 3165  1          SND_ENVELOPE [.NSD_SLOT, RSV1] = ZERO;      !used in the DUP implementation
; C 3166  1          SND_ENVELOPE [.NSD_SLOT, RSV2] = ZERO;      !
; C 3167  1          SND_ENVELOPE [.NSD_SLOT, RSV3] = ZERO;      !
; C 3168  1          !
; C 3169  1          ! These  next field definitions are the same
; C 3170  1          ! as  above  except they are for the overlay
; C 3171  1          ! buffer descriptors.  To make life easy for
; C 3172  1          ! me I'll use the same names and just prefix
; C 3173  1          ! them with a $ for uniqueness.
; C 3174  1          !
; C 3175  1          ! The overlay area immediately follows the
```

```
; C 3176  1          ! initial load image in the program image.
; C 3177  1          !
; C 3178  1          SND_ENVELOPE [.NSD_SLOT, $BPA_LO] = .OVSA;  !Low unibus adrs <0-15>
; C 3179  1          SND_ENVELOPE [.NSD_SLOT, $BPA_HI] = ZERO;   !Unibus adrs bits <16-17>
; C 3180  1          SND_ENVELOPE [.NSD_SLOT, $QBUS_EXT] = ZERO; !Q_bus extention adrs
; C 3181  1          SND_ENVELOPE [.NSD_SLOT, $RSV] = ZERO;      !Reserved field
; C 3182  1          SND_ENVELOPE [.NSD_SLOT, $UBA_CHAN] = ZERO; !Unibus adaptor channel number
; C 3183  1          SND_ENVELOPE [.NSD_SLOT, $RSV0] = ZERO;     !These next four words are not
; C 3184  1          SND_ENVELOPE [.NSD_SLOT, $RSV1] = ZERO;     !used in the DUP implementation
; C 3185  1          SND_ENVELOPE [.NSD_SLOT, $RSV2] = ZERO;     !
; C 3186  1          SND_ENVELOPE [.NSD_SLOT, $RSV3] = ZERO;     !
; C 3187  1          !
; C 3188  1          ! Call   the  load  out$standing  command buffer routine
; C 3189  1          ! and  load  this command  into  the buffer.  The return
; C 3190  1          ! from this routine will point us to the buffer location
; C 3191  1          ! where  this command  is  stored.  Later we can look at
; C 3192  1          ! this  location to see if the interrupt service routine
; C 3193  1          ! has received and process it.
; C 3194  1          !
; C 3195  1          ESP_BUF$LOC = LOAD_OUT$STD_BUF (.REF_NUM);  !Load the command
; C 3196  1
; C 3197  1          if .ESP_BUF$LOC eqlu OBF_CODE then DECODE ();       !Error if buffer is full
; C 3198  1
; C 3199  1          !
; C 3200  1          ! Set the ownership bit to 1 giving this slot
; C 3201  1          ! to the port/controller
; C 3202  1          !
; C 3203  1          SEND_RING [.NSD_SLOT, OWN_BIT] = PORT_OWNED;
; C 3204  1          !
; C 3205  1          ! Read the IP register to stimulate port polling
; C 3206  1          !
; C 3207  1          TEMP = .RDRX_ADDR [RCIP, RC_ALL];
; C 3208  1          !
; C 3209  1          ! Time  out the  port/controller processing  the command.
; C 3210  1          !
; C 3211  1          ! The   first   test   tests the connections  ability to
; C 3212  1          ! respond to  this command without  any errors in the SA
; C 3213  1          ! register and for the command not timing out.
; C 3214  1          !
; C 3215  1          ! The  second tests the  DUP server for good status.  If
; C 3216  1          ! bad status is sent back then an error code is returned
; C 3217  1          ! to the calling routine where the routine "decode" will
; C 3218  1          ! decode  and take the  appropriate recovery.  The  time
; C 3219  1          ! out  routine will loop on delaying and checking the hi
; C 3220  1          ! bit of the  first word in  the  out$std_buf for a true.
; C 3221  1          ! When true signals us that the interrupt service routine
; C 3222  1          ! has received  the endpacket  and  no connection errors
; C 3223  1          ! were detected.
; C 3224  1          !
; C 3225  1
; C 3226  1          if CTO_WAIT (%o'3000', .REF_NUM, .ESP_BUF$LOC) then DECODE ();      !Is return an error
; C 3227  1
; C 3228  1          !
```

```
; C 3229  1          ! Get  the return envelope  address  from the  out$std_buf
; C 3230  1          ! at  this commands  buffer location  and check the packet
; C 3231  1          ! for good status error and die if bad status was returned
; C 3232  1          !
; C 3233  1          RET_EN$AD = .OUT$STD_BUF [.ESP_BUF$LOC, ENV_ADR];   !Get the ret env adr
; C 3234  1          !
; C 3235  1          ! Now test for good status
; C 3236  1          !
; C 3237  1
; C 3238  1          if .RET_EN$AD [STATUS] nequ ZERO                    !Test the status
; C 3239  1          then
; C 3240  1              return RET_STATUS = RSE_CODE                    !Return a "Response status err" code
; C 3241  1          else
; C 3242  1              return .RET_STATUS;                             !This ret_status is good or bad
; C 3243  1
; C 3244  1          end;
;    3245  1      )%
```

```
;  3246  1     global routine EX_LOC_PROG =    !Executes local program
;  3247  1
;  3248  1     !++
;  3249  1     ! Functional Description :
;  3250  1     !       Receipt of this  command  causes the controller to search its local
;  3251  1     !       media  for  the named  program, load  and execute it.  Receipt of a
;  3252  1     !       successful response by the host means that the program is executing
;  3253  1     !       and the server is in the active state.  The time out value for this
;  3254  1     !       command is specified in the get dust response
;  3255  1     !
;  3256  1     ! Formal Parameters :
;  3257  1     !       none
;  3258  1     !
;  3259  1     ! Implicit Inputs :
;  3260  1     !       NSD_SLOT            This  global storage gets  loaded  by  the routine
;  3261  1     !                           'Get_nsd'  and  in it is stored the next send ring
;  3262  1     !                           descriptor  slot  where the port/controller should
;  3263  1     !                           be polling on and the  place  to put this commands
;  3264  1     !                           command packet.
;  3265  1     !
;  3266  1     ! Implicit Outputs :
;  3267  1     !       none
;  3268  1     !
;  3269  1     ! Completion Codes :
;  3270  1     !       RET_STATUS:         Return status passes back to the calling routine
;  3271  1     !                           the status of the just issued command.
;  3272  1     !
;  3273  1     ! Side Effects :
;  3274  1     !       The DM machine in the controller goes from the idle state to the
;  3275  1     !       active state on return of a successful return packet.
;  3276  1     !--
;  3277  1
;  3278  2         begin
;  3279  2
;  3280  2         local
;  3281  2             REF_NUM,                                !Stores unique cmd ref number
;  3282  2             ELP_BUF$LOC,                            !Stores outstanding cmd buffer location
;  3283  2             TEMP;                                   !A place to store the read IP register data
;  3284  2
;  3285  2         !
;  3286  2         ! Before we load up the command packet up
;  3287  2         ! with  all this good information get the
;  3288  2         ! next send  descriptor slot and a unique
;  3289  2         ! command reference number.
;  3290  2         !
;  3291  2         GET_NSD ();                                 !Get the next send desc slot
;  3292  2         REF_NUM = GET_CMD$REF ();                   !Get a unique command ref num
;  3293  2         !
;  3294  2         ! UQ Port command envelope Header field definition
;  3295  2         !
;  3296  2         SND_ENVELOPE [.NSD_SLOT, MSG_LENGTH] = SZ_ELP;      !Load the message size
;  3297  2         SND_ENVELOPE [.NSD_SLOT, CREDITS] = ONE;    !Load the credit size
;  3298  2         SND_ENVELOPE [.NSD_SLOT, MSG_TYPE] = 0;     !Define the msg typ 'Sequential'
```

```
;    3299  2          SND_ENVELOPE [.NSD_SLOT, CONN_ID] = 2;      !Define the connectio ID 'DUP'
;    3300  2          !
;    3301  2          ! DUP generic command envelope field definition
;    3302  2          !
;    3303  2          SND_ENVELOPE [.NSD_SLOT, CMD_LREF] = .REF_NUM;      !Load the command ref number
;    3304  2          SND_ENVELOPE [.NSD_SLOT, CMD_HREF] = ZERO;  !Zero the Hi order cmd ref num
;    3305  2          SND_ENVELOPE [.NSD_SLOT, UN_LUSED] = ZERO;  !Not used in DUP implimentation
;    3306  2          SND_ENVELOPE [.NSD_SLOT, UN_HUSED] = ZERO;  !Not used in DUP implimentation
;    3307  2          SND_ENVELOPE [.NSD_SLOT, OPCODE] = OP_ELP;  !Load this commands op-code
;    3308  2          SND_ENVELOPE [.NSD_SLOT, RSVD] = ZERO;      !Not used field
;    3309  2          SND_ENVELOPE [.NSD_SLOT, MODIFIER] = DUP_STND; !Define modifiers for this cmd
;    3310  2          !
;    3311  2          ! Command specfic command envelope field definition
;    3312  2          !
;    3313  2          SND_ENVELOPE [.NSD_SLOT, PN_0] = 'FO';      !Program name word 0
;    3314  2          SND_ENVELOPE [.NSD_SLOT, PN_1] = 'RM';      !Program name word 1
;    3315  2          SND_ENVELOPE [.NSD_SLOT, PN_2] = 'AT';      !Program name word 2
;    3316  2          !
;    3317  2          ! Call   the  load  out$standing  command buffer routine
;    3318  2          ! and  load  this command  into  the buffer.  The return
;    3319  2          ! from this routine will point us to the buffer location
;    3320  2          ! where  this command  is  stored.  Later we can look at
;    3321  2          ! this  location to see if the interrupt service routine
;    3322  2          ! has received and process it.
;    3323  2          !
;    3324  2          ELP_BUF$LOC = LOAD_OUT$STD_BUF (.REF_NUM);  !Load the command
;    3325  2
;    3326  2          if .ELP_BUF$LOC eqlu OBF_CODE then DECODE ();       !Error if buffer is full
;    3327  2
;    3328  2          !
;    3329  2          ! Set the ownership bit to 1 giving this slot
;    3330  2          ! to the port/controller
;    3331  2          !
;    3332  2          SEND_RING [.NSD_SLOT, OWN_BIT] = PORT_OWNED;
;    3333  2          !
;    3334  2          ! Read the IP register to stimulate port polling
;    3335  2          !
;    3336  2          TEMP = .RDRX_ADDR [RCIP, RC_ALL];
;    3337  2          !
;    3338  2          ! Time  out the  port/controller processing  the command.
;    3339  2          !
;    3340  2          ! The   first   test   tests the connections  ability to
;    3341  2          ! respond to  this command without  any errors in the SA
;    3342  2          ! register and for the command not timing out.
;    3343  2          !
;    3344  2          ! The  second tests the  DUP server for good status.  If
;    3345  2          ! bad status is sent back then an error code is returned
;    3346  2          ! to the calling routine where the routine "decode" will
;    3347  2          ! decode  and take the  appropriate recovery.  The  time
;    3348  2          ! out  routine will loop on delaying and checking the hi
;    3349  2          ! bit of the  first word in  the  out$std_buf for a true.
;    3350  2          ! When true signals us that the interrupt service routine
;    3351  2          ! has received  the endpacket  and  no connection errors
```

```
;    3352  2          ! were detected.
;    3353  2          !
;    3354  2
;    3355  2          if CTO_WAIT (3000, .REF_NUM, .ELP_BUF$LOC) then DECODE ();  !Is return an error
;    3356  2
;    3357  2          !
;    3358  2          ! Get  the return envelope  address  from the  out$std_buf
;    3359  2          ! at  this commands  buffer location  and check the packet
;    3360  2          ! for good status error and die if bad status was returned
;    3361  2          !
;    3362  2          RET_EN$AD = .OUT$STD_BUF [.ELP_BUF$LOC, ENV_ADR];   !Get the ret env adr
;    3363  2          !
;    3364  2          ! Now test for good status
;    3365  2          !
;    3366  2
;    3367  2          if .RET_EN$AD [STATUS] nequ ZERO            !Test the status
;    3368  2          then
;    3369  2              return RET_STATUS = RSE_CODE            !Return a "Response status err" code
;    3370  2          else
;    3371  2              return .RET_STATUS;                     !This ret_status is good or bad
;    3372  2
;    3373  1          end;


                                        .SBTTL  EX.LOC.PROG GLOBAL ROUTINE DECLARATIONS
000000  004137  000000G          EX.LOC.PROG::
                                        JSR     R1,$SAVE2                       ;                               3246
000004  005746                         TST     -(SP)
000006  004737  000000'                 JSR     PC,GET.NSD                      ;                               3291
000012  004737  000146'                 JSR     PC,GET.CMD$REF                  ;                               3292
000016  010002                         MOV     R0,R2                           ; *,REF.NUM
000020  013746  000000G                 MOV     NSD,SLOT,-(SP)                  ;                               3296
000024  012746  000054                 MOV     #54,-(SP)
000030  004737  000000G                 JSR     PC,BL$MUL
000034  012760  000060  000000G         MOV     #60,SND.ENVELOPE(R0)
000042  012701  000002G                 MOV     #SND.ENVELOPE+2,R1              ;                               3297
000046  060001                         ADD     R0,R1
000050  112711  000001                 MOVB    #1,(R1)                         ;                               3298
000054  112761  000002  000001         MOVB    #2,1(R1)                        ;                               3299
000062  010260  000004G                 MOV     R2,SND.ENVELOPE+4(R0)           ; REF.NUM,*                     3303
000066  005060  000006G                 CLR     SND.ENVELOPE+6(R0)              ;                               3304
000072  005060  000010G                 CLR     SND.ENVELOPE+10(R0)             ;                               3305
000076  005060  000012G                 CLR     SND.ENVELOPE+12(R0)             ;                               3306
000102  112760  000003  000014G         MOVB    #3,SND.ENVELOPE+14(R0)          ;                               3307
000110  105060  000015G                 CLRB    SND.ENVELOPE+15(R0)             ;                               3308
000114  012760  000001  000016G         MOV     #1,SND.ENVELOPE+16(R0)          ;                               3309
000122  012760  047506  000020G         MOV     #47506,SND.ENVELOPE+20(R0)      ;                               3313
000130  012760  046522  000022G         MOV     #46522,SND.ENVELOPE+22(R0)      ;                               3314
000136  012760  052101  000024G         MOV     #52101,SND.ENVELOPE+24(R0)      ;                               3315
000144  010216                         MOV     R2,(SP)                         ; REF.NUM,*                     3324
000146  004737  000054'                 JSR     PC,LOAD.OUT$STD.BUF
000152  010001                         MOV     R0,R1                           ; *,ELP.BUF$LOC
000154  020127  002001                 CMP     R1,#2001                        ; ELP.BUF$LOC,*                 3326
```

```
000160  001002                          BNE     1$
000162  004737  000250'                 JSR     PC,DECODE
000166  013700  000000G          1$:    MOV     NSD.SLOT,R0             ;                               3332
000172  006300                          ASL     R0
000174  006300                          ASL     R0
000176  063700  000000G                 ADD     SEND.RING,R0
000202  052760  100000  000002          BIS     #100000,2(R0)
000210  017766  000000G 000004          MOV     @RDRX.ADDR,4(SP)        ; *,RC$S.REG                    3336
000216  016600  000004                  MOV     4(SP),R0                ; RC$S.REG,TEMP
000222  012716  005670                  MOV     #5670,(SP)              ;                               3355
000226  010246                          MOV     R2,-(SP)                ; REF.NUM,*
000230  010146                          MOV     R1,-(SP)                ; ELP.BUF$LOC,*
000232  004737  002116'                 JSR     PC,CTO.WAIT
000236  022626                          CMP     (SP)+,(SP)+
000240  006000                          ROR     R0
000242  103002                          BCC     2$
000244  004737  000250'                 JSR     PC,DECODE
000250  010100                  2$:     MOV     R1,R0                   ; ELP.BUF$LOC,*                 3362
000252  006300                          ASL     R0
000254  006300                          ASL     R0
000256  016037  000002G 000000G         MOV     OUT$STD.BUF+2(R0),RET.EN$AD
000264  016000  000002G                 MOV     OUT$STD.BUF+2(R0),R0    ;                               3367
000270  005760  000016                  TST     16(R0)
000274  001405                          BEQ     3$
000276  012700  000031                  MOV     #31,R0                  ;                               3369
000302  010037  000000G                 MOV     R0,RET.STATUS
000306  000402                          BR      4$                      ;                               3371
000310  013700  000000G          3$:    MOV     RET.STATUS,R0
000314  062706  000006          4$:     ADD     #6,SP                   ;                               3246
000320  000207                          RTS     PC

; Routine Size:  105 words,    Routine Base:  $CODE$ + 3102
; Maximum stack depth per invocation:  9 words

;   3374  1
```

```
;  3375  1     global routine SEND_DATA (mslen) =      !Performs host-->port communications
;  3376  1
;  3377  1     !++
;  3378  1     ! Functional Description :
;  3379  1     !       These  commands are  used to  communicate between  the initiating host
;  3380  1     !       program  and   the  remote  program.  Both  send  and receive commands
;  3381  1     !       specify   a  h st buffer descriptor  and a byte count.  In the case of
;  3382  1     !       send data, the information in the buffer is read by the remote program
;  3383  1     !       and  a send data response sent back to the host to acknowledge receipt.
;  3384  1     !       In   the case  of receive data the remote program writes data into the
;  3385  1     !       buffer  up  to the amount specified by the byte count and then sends a
;  3386  1     !       receive data  response to  the host  to notify it of the transmission.
;  3387  1     !
;  3388  1     !       The  send data  and receive   data commands  are  only  legal when the
;  3389  1     !       server   is   in the  active state.  If  the remote program terminates
;  3390  1     !       abnormally, putting   the server   back  in the idle stae, outstanding
;  3391  1     !       send  data  and receive  data commands may be lost.  In the event that
;  3392  1     !       the  specified timeout  interval is  exceeded, the host program should
;  3393  1     !       issue   a get   dust  staus  command to  see if the  remote program is
;  3394  1     !       still    running  (is.  the  dup  server is  active);  if it  is, the
;  3395  1     !       progress   indicator should  be  remembered and  the  timeout interval
;  3396  1     !       should   be re-installed.  If   the second   timeout expires without a
;  3397  1     !       response and   a second   get  dust status shows  the  remote  program
;  3398  1     !       having  made   no progress   in the interim then the program should be
;  3399  1     !       considered broken and should be aborted.
;  3400  1     !
;  3401  1     ! Formal Parameters :
;  3402  1     !       none
;  3403  1     !
;  3404  1     ! Implicit Inputs :
;  3405  1     !       NSD_SLOT        This  global storage gets  loaded  by  the routine
;  3406  1     !                       'Get_nsd'  and  in it is stored the next send ring
;  3407  1     !                       descriptor  slot  where the port/controller should
;  3408  1     !                       be polling on and the  place  to put this commands
;  3409  1     !                       command packet.
;  3410  1     !
;  3411  1     ! Implicit Outputs :
;  3412  1     !       none
;  3413  1     !
;  3414  1     ! Completion Codes :
;  3415  1     !       RET_STATUS:     Return status passes back to the calling routine
;  3416  1     !                       the status of the just issued command.
;  3417  1     !
;  3418  1     ! Side Effects :
;  3419  1     !       none
;  3420  1     !--
;  3421  1
;  3422  2         begin
;  3423  2
;  3424  2         local
;  3425  2             REF_NUM,                            !Stores unique cmd ref number
;  3426  2             SND_BUF$LOC,                        !Stores out$standing cmd buffer location
;  3427  2             TEMP;                               !A place to put read IP register data
```

```
;  3428  2
;  3429  2          !
;  3430  2          ! Before we load up the command packet up
;  3431  2          ! with  all this good information get the
;  3432  2          ! next send  descriptor slot and a unique
;  3433  2          ! command reference number.
;  3434  2          !
;  3435  2          GET_NSD ();                                 !Get the next send desc slot
;  3436  2          REF_NUM = GET_CMD$REF ();                   !Get a unique command ref num
;  3437  2          !
;  3438  2          ! UQ Port command envelope Header field definition
;  3439  2          !
;  3440  2          SND_ENVELOPE [.NSD_SLOT, MSG_LENGTH] = SZ_SED;      !Load the message size
;  3441  2          SND_ENVELOPE [.NSD_SLOT, CREDITS] = ONE;    !Load the credit size
;  3442  2          SND_ENVELOPE [.NSD_SLOT, MSG_TYPE] = 0;     !Define the message typ 'Sequential
;  3443  2          SND_ENVELOPE [.NSD_SLOT, CONN_ID] = 2;      !Define the connection ID 'DUP'
;  3444  2          !
;  3445  2          ! DUP generic command envelope field definition
;  3446  2          !
;  3447  2          SND_ENVELOPE [.NSD_SLOT, CMD_LREF] = .REF_NUM;      !Load command reference number
;  3448  2          SND_ENVELOPE [.NSD_SLOT, CMD_HREF] = ZERO;  !Zero Hi order cmd ref number
;  3449  2          SND_ENVELOPE [.NSD_SLOT, UN_LUSED] = ZERO;  !Not used in DUP implimentation
;  3450  2          SND_ENVELOPE [.NSD_SLOT, UN_HUSED] = ZERO;  !Not used in DUP implimentation
;  3451  2          SND_ENVELOPE [.NSD_SLOT, OPCODE] = OP_SED;  !Load this commands op-code
;  3452  2          SND_ENVELOPE [.NSD_SLOT, RSVD] = ZERO;      !Not used field
;  3453  2          SND_ENVELOPE [.NSD_SLOT, MODIFIER] = ZERO;  !Define the commands modifiers
;  3454  2          !
;  3455  2          ! Command specfic command envelope field definition
;  3456  2          !
;  3457  2          !
;  3458  2          ! Byte count of transfer
;  3459  2          !
;  3460  2          SND_ENVELOPE [.NSD_SLOT, BLO_CNT] = .mslen; !Byte count low word
;  3461  2          SND_ENVELOPE [.NSD_SLOT, BHI_CNT] = ZERO;   !Byte count high word
;  3462  2          !
;  3463  2          ! Buffer descriptor definition                                                    C
;  3464  2          !
;  3465  2          SND_ENVELOPE [.NSD_SLOT, BPA_LO] = SND_BUF; !Buffer physical adrs <0-15>         C
;  3466  2          SND_ENVELOPE [.NSD_SLOT, BPA_HI] = ZERO;    !Buffer physical adrs bits <16-17>   C
;  3467  2          SND_ENVELOPE [.NSD_SLOT, QBUS_EXT] = ZERO;  !Q_bus extention adrs                 C
;  3468  2          SND_ENVELOPE [.NSD_SLOT, RSV] = ZERO;       !Reserved field                      C
;  3469  2          SND_ENVELOPE [.NSD_SLOT, UBA_CHAN] = ZERO;  !Unibus adaptor channel number       C
;  3470  2          SND_ENVELOPE [.NSD_SLOT, RSV0] = ZERO;      !These next four words are not       C
;  3471  2          SND_ENVELOPE [.NSD_SLOT, RSV1] = ZERO;      !used in the UQ Port implementation  C
;  3472  2          SND_ENVELOPE [.NSD_SLOT, RSV2] = ZERO;      !                                    C
;  3473  2          SND_ENVELOPE [.NSD_SLOT, RSV3] = ZERO;      !                                    C
;  3474  2          !                                                                                C
;  3475  2          ! Call   the  load  out$standing  command buffer routine                         C
;  3476  2          ! and  load  this command  into  the buffer.  The return                         C
;  3477  2          ! from this routine will point us to the buffer location                         C
;  3478  2          ! where  this command  is  stored.  Later we can look at                         C
;  3479  2          ! this  location to see if the interrupt service routine                         C
;  3480  2          ! has received and process it.                                                   C
```

```
;  3481  2          !
;  3482  2          SND_BUF$LOC = LOAD_OUT$STD_BUF (.REF_NUM);  !Load the command
;  3483  2
;  3484  2          if .SND_BUF$LOC eqlu OBF_CODE then DECODE ();       !Error if buffer is full
;  3485  2
;  3486  2          !
;  3487  2          ! Set the ownership bit to 1 giving this slot
;  3488  2          ! to the port/controller
;  3489  2          !
;  3490  2          SEND_RING [.NSD_SLOT, OWN_BIT] = PORT_OWNED;
;  3491  2          !
;  3492  2          ! Read the IP register to stimulate port polling
;  3493  2          !
;  3494  2          TEMP = .RDRX_ADDR [RCIP, RC_ALL];
;  3495  2
;  3496  2          !+
;  3497  2          ! Time out the port/controller for the response from this
;  3498  2          ! command.
;  3499  2          !
;  3500  2          ! If the controller times out then:
;  3501  2          !   1. See  what  kind  of  error was returned.  If the error
;  3502  2          !      is  a type other than a CTO_CODE (controller time out)
;  3503  2          !      then  call  routine  Decode which does the appropriate
;  3504  2          !      action based on the error.
;  3505  2          !
;  3506  2          !   2. If  the  returned  error  is an CTO_CODE then do a get
;  3507  2          !      dust  status  and check the progress indicator to look
;  3508  2          !      for an increase, indicating that the remote program is
;  3509  2          !      still running and is not dead.
;  3510  2          !
;  3511  2          !      If   the indicator hasn't changed then assume that the
;  3512  2          !      remote  program  is  dead  and return an error code of
;  3513  2          !      RPD_CODE (remote  program  dead  code)  and  exit.
;  3514  2          !
;  3515  2          !      If the  indicator  has  changed  then  assume that the
;  3516  2          !      remote  program  is  still running, save a copy of its
;  3517  2          !      value and  reinstate the controller time out delay and
;  3518  2          !      repeat the loop.
;  3519  2          !
;  3520  2          !      As long as the progress indicator in the remote program
;  3521  2          !      is still increasing this loop will be repeated for ever.
;  3522  2          !
;  3523  2          ! If  the  controller doesn't time then return with the return
;  3524  2          ! code returned from routine CTO_WAIT () which could be either
;  3525  2          ! a  success  or  error  code  by definition of this host code.
;  3526  2          !-
;  3527  2
;  3528  2          while TRUE do                                   !Repeat for ever
;  3529  3              begin
;  3530  3              BREAK;                                      !Flag control C's
;  3531  3              !
;  3532  3              ! Do a controller time out and  determine if the controller
;  3533  3              ! has processed the command or if a fatal error has occured.
```

```
;  3534  3              !
;  3535  3
;  3536  3              if CTO_WAIT (3000, .REF_NUM, .SND_BUF$LOC)      !Is return an error
;  3537  3              then
;  3538  4                  begin
;  3539  4                  !
;  3540  4                  ! If the return status code eql's a CTO_CODE
;  3541  4                  ! then  see  if the  remote program is still
;  3542  4                  ! running.  If  it is then save the progress
;  3543  4                  ! indicator  and repeat  the  loop else call
;  3544  4                  ! routine Decode ().
;  3545  4                  !
;  3546  4
;  3547  4                  if .RET_STATUS eqlu CTO_CODE        !Is this a controller time out
;  3548  4                  then
;  3549  5                      begin
;  3550  5
;  3551  5                      REF_NUM = .REF_NUM;
;  3552  5
;  3553  5
;  3554  5                      if GET_DUST_STATUS () then DECODE ();   !Get the dust status
;  3555  5
;  3556  5
;  3557  5
;  3558  5                      if .RET_EN$AD [PLO_IND] gtru .PID_SAVE  !Any progress been made
;  3559  5                      then
;  3560  5                          PID_SAVE = .RET_EN$AD [PLO_IND]     !Still running save Pid
;  3561  5                      else
;  3562  5                          return RET_STATUS = RPD_CODE;       !No progress so flag error
;  3563  5                      end
;  3564  4                  else
;  3565  4                  !
;  3566  4                  ! The return status code was not a controller time
;  3567  4                  ! out  code so  something else is wrong.  Call the
;  3568  4                  ! routine  Decode ()  to  find out what went wrong.
;  3569  4                  !
;  3570  4                      DECODE ()
;  3571  4
;  3572  4                  end
;  3573  3              else
;  3574  4                  begin
;  3575  4                  !
;  3576  4                  ! The command has been received by the interrupt service.
;  3577  4                  !
;  3578  4                  ! Get  this commands return  envelope address out of the
;  3579  4                  ! out$std_buf and check for good return status error and
;  3580  4                  ! die if bad status.
;  3581  4                  !
;  3582  4                  RET_EN$AD = .OUT$STD_BUF [.SND_BUF$LOC, ENV_ADR];   !Get the ret env adr
;  3583  4                  !
;  3584  4                  ! Test for good status
;  3585  4                  !
;  3586  4
```

```
;    3587  4                 if .RET_EN$AD [STATUS] nequ ZERO    !Test the status
;    3588  4                 then
;    3589  4                     return RET_STATUS = RSE_CODE    !Return a "Response status err" code
;    3590  4                 else
;    3591  4                     return .RET_STATUS;             !This ret_status is good or bad
;    3592  4
;    3593  3                 end;
;    3594  3
;    3595  2             end;
;    3596  2
;    3597  2         return .RET_STATUS;                         !It won't compile without this here
;    3598  1         end;


                                        .SBTTL  SEND.DATA GLOBAL ROUTINE DECLARATIONS
000000  004137  000000G           SEND.DATA::
                                        JSR     R1,$SAVE2                       ;                       3375
000004  005746                          TST     -(SP)
000006  004737  000000'                JSR     PC,GET.NSD                      ;                       3435
000012  004737  000146'                JSR     PC,GET.CMD$REF                  ;                       3436
000016  010002                          MOV     R0,R2                           ; ^,REF.NUM
000020  013746  000000G                MOV     NSD.SLOT,-(SP)                  ;                       3440
000024  012746  000054                 MOV     #54,-(SP)
000030  004737  000000G                JSR     PC,BL$MUL
000034  012760  000034  000000G       MOV     #34,SND.ENVELOPE(R0)
000042  012701  000002G                MOV     #SND.ENVELOPE+2,R1              ;                       3441
000046  060001                          ADD     R0,R1
000050  112711  000001                 MOVB    #1,(R1)                         ;                       3442
000054  112761  000002  000001        MOVB    #2,1(R1)                        ;                       3443
000062  010260  000004G               MOV     R2,SND.ENVELOPE+4(R0)           ; REF.NUM,^             3447
000066  005060  000006G               CLR     SND.ENVELOPE+6(R0)              ;                       3448
000072  005060  000010G               CLR     SND.ENVELOPE+10(R0)             ;                       3449
000076  005060  000012G               CLR     SND.ENVELOPE+12(R0)             ;                       3450
000102  112760  000004  000014G       MOVB    #4,SND.ENVELOPE+14(R0)          ;                       3451
000110  105060  000015G               CLRB    SND.ENVELOPE+15(R0)             ;                       3452
000114  005060  000016G               CLR     SND.ENVELOPE+16(R0)             ;                       3453
000120  016660  000016  000020G       MOV     16(SP),SND.ENVELOPE+20(R0)      ; MSLEN,^               3460
000126  005060  000022G               CLR     SND.ENVELOPE+22(R0)             ;                       3461
000132  012760  000000G 000024G       MOV     #SND.BUF,SND.ENVELOPE+24(R0)    ;                       3465
000140  012701  000026G               MOV     #SND.ENVELOPE+26,R1             ;                       3466
000144  060001                          ADD     R0,R1
000146  105011                          CLRB    (R1)                            ;                       3468
000150  105061  000001                 CLRB    1(R1)                           ;                       3469
000154  005060  000030G               CLR     SND.ENVELOPE+30(R0)             ;                       3470
000160  005060  000032G               CLR     SND.ENVELOPE+32(R0)             ;                       3471
000164  005060  000034G               CLR     SND.ENVELOPE+34(R0)             ;                       3472
000170  005060  000036G               CLR     SND.ENVELOPE+36(R0)             ;                       3473
000174  010216                          MOV     R2,(SP)                         ; REF.NUM,^             3482
000176  004737  000054'                JSR     PC,LOAD.OUT$STD.BUF
000202  010001                          MOV     R0,R1                           ; ^,SND.BUF$LOC
000204  020127  002001                 CMP     R1,#2001                        ; SND.BUF$LOC,^         3484
000210  001002                          BNE     1$
000212  004737  000250'                JSR     PC,DECODE
```

```
000216  013700  000000G                1$:     MOV     NSD.SLOT,R0                     ;                               3490
000222  006300                                  ASL     R0
000224  006300                                  ASL     R0
000226  063700  000000G                        ADD     SEND.RING,R0
000232  052760  100000  000002                BIS     #100000,2(R0)
000240  017766  000000G 000004                MOV     @RDRX.ADDR,4(SP)                ; *,RC$S.REG                    3494
000246  016600  000004                        MOV     4(SP),R0                        ; RC$S.REG,TEMP
000252  104422                        2$:     TRAP    22                              ;                               3529
000254  012716  005670                        MOV     #5670,(SP)                      ;                               3536
000260  010246                                MOV     R2,-(SP)                        ; REF.NUM,*
000262  010146                                MOV     R1,-(SP)                        ; SND.BUF$LOC,*
000264  004737  002116'                        JSR     PC,CTO.WAIT
000270  022626                                CMP     (SP)+,(SP)+
000272  006000                                ROR     R0
000274  103032                                BCC     6$
000276  023727  000000G 000011                CMP     RET.STATUS,#11                  ;                               3547
000304  001023                                BNE     5$
000306  004737  002604'                        JSR     PC,GET.DUST.STATUS              ;                               3554
000312  006000                                ROR     R0
000314  103002                                BCC     3$
000316  004737  000250'                        JSR     PC,DECODE
000322  013700  000000G                3$:     MOV     RET.EN$AD,R0                    ;                               3558
000326  026037  000024  000000G               CMP     24(R0),PID.SAVE
000334  101404                                BLOS    4$
000336  016037  000024  000000G               MOV     24(R0),PID.SAVE                 ;                               3560
000344  000742                                BR      2$                              ;                               3558
000346  012700  000051                4$:     MOV     #51,R0                          ;                               3562
000352  000420                                BR      7$
000354  004737  000250'                5$:     JSR     PC,DECODE                       ;                               3570
000360  000734                                BR      2$                              ;                               3536
000362  010100                        6$:     MOV     R1,R0                           ; SND.BUF$LOC,*                 3582
000364  006300                                ASL     R0
000366  006300                                ASL     R0
000370  016037  000002G 000000G               MOV     OUT$STD.BUF+2(R0),RET.EN$AD
000376  016000  000002G                        MOV     OUT$STD.BUF+2(R0),R0            ;                               3587
000402  005760  000016                        TST     16(R0)
000406  001405                                BEQ     8$
000410  012700  000031                        MOV     #31,R0                          ;                               3589
000414  010037  000000G                7$:     MOV     R0,RET.STATUS
000420  000402                                BR      9$                              ;                               3591
000422  013700  000000G                8$:     MOV     RET.STATUS,R0
000426  062706  000006                9$:     ADD     #6,SP                           ;                               3375
000432  000207                                RTS     PC

; Routine Size:  142 words,    Routine Base:  $CODE$ + 3424
; Maximum stack depth per invocation:  9 words

;   3599  1
```

```
;  3600  1     global routine REC_DATA =       !Performs host-->port communications
;  3601  1
;  3602  1     !++
;  3603  1     ! Functional Description :
;  3604  1     !       These   commands  are used  to communicate between the initiating host
;  3605  1     !       program  and the  remote   program.  Both   send  and receive commands
;  3606  1     !       specify a  host buffer   descriptor  and a byte count.  In the case of
;  3607  1     !       send data, the information in the buffer is read by the remote program
;  3608  1     !       and  a send data response sent back to the host to acknowledge receipt.
;  3609  1     !       In  the  case of  receive data the remote program writes data into the
;  3610  1     !       buffer  up  to the amount specified by the byte count and then sends a
;  3611  1     !       receive   data  response to the host to notify it of the transmission.
;  3612  1     !
;  3613  1     !       The   send data   and  receive data  commands  are only legal when the
;  3614  1     !       server  is  in the   active state.  If   the remote program terminates
;  3615  1     !       abnormally, putting   the server  back  in the  idle stae, outstanding
;  3616  1     !       send  data  and receive  data commands may be lost.  In the event that
;  3617  1     !       the  specified  timeout interval  is exceeded, the host program should
;  3618  1     !       issue  a get   dus' staus   command   to see if  the remote program is
;  3619  1     !       still   running   (is. the dup  server  is active); if   it  is, the
;  3620  1     !       progress   indicator   should be   remembered and the timeout interval
;  3621  1     !       should  be re-installed.  If   the second   timeout  expires without a
;  3622  1     !       response and a   second  get dust  status  shows the  remote program
;  3623  1     !       having   made no  progress   in the interim then the program should be
;  3624  1     !       considered broken and should be aborted.
;  3625  1     !
;  3626  1     ! Formal Parameters :
;  3627  1     !       none
;  3628  1     !
;  3629  1     ! Implicit Inputs :
;  3630  1     !       NSD_SLOT        This  global storage gets  loaded  by  the routine
;  3631  1     !                       'Get_nsd'  and  in it is stored the next send ring
;  3632  1     !                       descriptor  slot  where the port/controller should
;  3633  1     !                       be polling on and the  place  to put this commands
;  3634  1     !                       command packet.
;  3635  1     !
;  3636  1     ! Implicit Outputs :
;  3637  1     !       none
;  3638  1     !
;  3639  1     ! Completion Codes :
;  3640  1     !       RET_STATUS:     Return status passes back to the calling routine
;  3641  1     !                       the status of the just issued command.
;  3642  1     !
;  3643  1     ! Side Effects :
;  3644  1     !       none
;  3645  1     !--
;  3646  1
;  3647  2         begin
;  3648  2
;  3649  2         local
;  3650  2             REF_NUM,                            !Stores unique cmd ref number
;  3651  2             REC_BUF$LOC,                        !Stores out$standing cmd buffer location
;  3652  2             TEMP;                               !A place to put read IP register data
```

```
;  3653  2
;  3654  2          !
;  3655  2          ! Before we load up the command packet up
;  3656  2          ! with  all this good information get the
;  3657  2          ! next send  descriptor slot and a unique
;  3658  2          ! command reference number.
;  3659  2          !
;  3660  2          GET_NSD ();                                 !Get the next send desc slot
;  3661  2          REF_NUM = GET_CMD$REF ();                   !Get a unique command ref num
;  3662  2          !
;  3663  2          ! UQ Port command envelope Header field definition
;  3664  2          !
;  3665  2          SND_ENVELOPE [.NSD_SLOT, MSG_LENGTH] = SZ_RED;      !Load message length
;  3666  2          SND_ENVELOPE [.NSD_SLOT, CREDITS] = ONE;    !Load credit size
;  3667  2          SND_ENVELOPE [.NSD_SLOT, MSG_TYPE] = 0;     !Define message type 'Sequential'
;  3668  2          SND_ENVELOPE [.NSD_SLOT, CONN_ID] = 2;      !Define connection ID 'DUP'
;  3669  2          !
;  3670  2          ! DUP generic command envelope field definition
;  3671  2          !
;  3672  2          SND_ENVELOPE [.NSD_SLOT, CMD_LREF] = .REF_NUM;      !Load command reference number
;  3673  2          SND_ENVELOPE [.NSD_SLOT, CMD_HREF] = ZERO;  !Zero Hi order cmd ref num
;  3674  2          SND_ENVELOPE [.NSD_SLOT, UN_LUSED] = ZERO;  !Not used in DUP implimentation
;  3675  2          SND_ENVELOPE [.NSD_SLOT, UN_HUSED] = ZERO;  !Not used in DUP implimentation
;  3676  2          SND_ENVELOPE [.NSD_SLOT, OPCODE] = OP_RED;  !Load this commands op-code
;  3677  2          SND_ENVELOPE [.NSD_SLOT, RSVD] = ZERO;      !Not used field
;  3678  2          SND_ENVELOPE [.NSD_SLOT, MODIFIER] = ZERO;  !Define this commands modifiers
;  3679  2          !
;  3680  2          ! Command specfic command envelope field definition
;  3681  2          !
;  3682  2          !
;  3683  2          ! Byte count of transfer
;  3684  2          !
;  3685  2          SND_ENVELOPE [.NSD_SLOT, BLO_CNT] = RECB_SIZE;      !Byte count low word
;  3686  2          SND_ENVELOPE [.NSD_SLOT, BHI_CNT] = ZERO;   !Byte count high word
;  3687  2          !
;  3688  2          ! Buffer descriptor definition
;  3689  2          !
;  3690  2          SND_ENVELOPE [.NSD_SLOT, BPA_LO] = REC_BUF; !Low unibus adrs <0-15>
;  3691  2          SND_ENVELOPE [.NSD_SLOT, BPA_HI] = ZERO;    !Unibus adrs bits <16-17>
;  3692  2          SND_ENVELOPE [.NSD_SLOT, QBUS_EXT] = ZERO;  !Q_bus extention adrs
;  3693  2          SND_ENVELOPE [.NSD_SLOT, RSV] = ZERO;       !Reserved field
;  3694  2          SND_ENVELOPE [.NSD_SLOT, UBA_CHAN] = ZERO;  !Unibus adaptor channel nunber
;  3695  2          SND_ENVELOPE [.NSD_SLOT, RSV0] = ZERO;      !These next four words are not
;  3696  2          SND_ENVELOPE [.NSD_SLOT, RSV1] = ZERO;      !used in the UQ Port implementation
;  3697  2          SND_ENVELOPE [.NSD_SLOT, RSV2] = ZERO;      !
;  3698  2          SND_ENVELOPE [.NSD_SLOT, RSV3] = ZERO;      !
;  3699  2          !
;  3700  2          ! Call   the  load  out$standing  command buffer routine
;  3701  2          ! and  load  this command  into  the buffer.  The return
;  3702  2          ! from this routine will point us to the buffer location
;  3703  2          ! where  this command  is  stored.  Later we can look at
;  3704  2          ! this  location to see if the interrupt service routine
;  3705  2          ! has received and process it.
```

```
;  3706  2          !
;  3707  2          REC_BUF$LOC = LOAD_OUT$STD_BUF (.REF_NUM);  !Load the command
;  3708  2
;  3709  2          if .REC_BUF$LOC eqlu OBF_CODE then DECODE ();       !Error if buffer is full
;  3710  2
;  3711  2          !
;  3712  2          ! Set the ownership bit to 1 giving
;  3713  2          ! this slot to the port/controller
;  3714  2          !
;  3715  2          SEND_RING [.NSD_SLOT, OWN_BIT] = PORT_OWNED;
;  3716  2          !
;  3717  2          ! Read the IP register to stimulate port polling
;  3718  2          !
;  3719  2          TEMP = .RDRX_ADDR [RCIP, RC_ALL];
;  3720  2
;  3721  2          !+
;  3722  2          ! Time out the port/controller for the response from this command.
;  3723  2          !
;  3724  2          ! If the controller times out then:
;  3725  2          !   1. See  what  kind  of  error was returned.  If the error
;  3726  2          !      is  a type other than a CTO_CODE (controller time out)
;  3727  2          !      then  call  routine  Decode which does the appropriate
;  3728  2          !      action based on the error.
;  3729  2          !
;  3730  2          !   2. If  the  returned  error  is an CTO_CODE then do a get
;  3731  2          !      dust  status  and check the progress indicator to look
;  3732  2          !      for an increase, indicating that the remote program is
;  3733  2          !      still running and is not dead.
;  3734  2          !
;  3735  2          !      If  the  indicator hasn't changed then assume that the
;  3736  2          !      remote  program  is  dead  and return an error code of
;  3737  2          !      RPD_CODE (remote program dead code) and exit.
;  3738  2          !
;  3739  2          !      If  the  indicator  has  changed  then assume that the
;  3740  2          !      remote  program  is  still running, save a copy of its
;  3741  2          !      value and  reinstate the controller time out delay and
;  3742  2          !      repeat the loop.
;  3743  2          !
;  3744  2          !      As long as the progress indicator in the remote program
;  3745  2          !      is still increasing this loop will be repeated for ever.
;  3746  2          !
;  3747  2          ! If  the  controller doesn't time then return with the return code
;  3748  2          ! returned from routine CTO_WAIT () which could be either a success
;  3749  2          ! or error code by definition of this host code.
;  3750  2          !-
;  3751  2
;  3752  2          while TRUE do                                   !Repeat for ever
;  3753  3              begin
;  3754  3              BREAK;                                      !Flag control C's
;  3755  3              !
;  3756  3              ! Do a controller time out and  determine if the controller
;  3757  3              ! has processed the command or if a fatal error has occured.
;  3758  3              !
```

```
;  3759  3
;  3760  3            if CTO_WAIT (3000, .REF_NUM, .REC_BUF$LOC)      !Is return an error
;  3761  3            then
;  3762  4                begin
;  3763  4                !
;  3764  4                ! If the return status code eql's a CTO_CODE
;  3765  4                ! then  see  if the  remote program is still
;  3766  4                ! running.  If  it is then save the progress
;  3767  4                ! indicator  and repeat  the  loop else call
;  3768  4                ! routine Decode ().
;  3769  4                !
;  3770  4
;  3771  4                if .RET_STATUS eqlu CTO_CODE        !Is this a controller time out
;  3772  4                then
;  3773  5                    begin
;  3774  5
;  3775  5                    REF_NUM = .REF_NUM;
;  3776  5
;  3777  5
;  3778  5                    if GET_DUST_STATUS () then DECODE ();   !Get the dust status
;  3779  5
;  3780  5                    if .RET_ENVAD [PLO_IND] neq .PID_SAVE   !Any progress been made
;  3781  5                    then
;  3782  5                        PID_SAVE = .RET_ENVAD [PLO_IND]     !Still running save Pid
;  3783  5                    else
;  3784  5                        return RET_STATUS = RPD_CODE;       !No progress so flag error
;  3785  5
;  3786  5                    end
;  3787  4                else
;  3788  4                !
;  3789  4                ! The return status code was not a controller time
;  3790  4                ! out  code so  something else is wrong.  Call the
;  3791  4                ! routine  Decode ()  to find out  what went wrong.
;  3792  4                !
;  3793  4                    DECODE ()
;  3794  4
;  3795  4                end
;  3796  3            else
;  3797  4                begin
;  3798  4                !
;  3799  4                ! The command has been received by the interrupt service.
;  3800  4                !
;  3801  4                ! Get this  commands return  envelope address out of the
;  3802  4                ! outstd_buf and check for good return status error and
;  3803  4                ! die if bad status.
;  3804  4                !
;  3805  4                RET_ENVAD = .OUTSTD_BUF [.REC_BUF$LOC, ENV_ADR];   !Get the ret env adr
;  3806  4                !
;  3807  4                ! Test for good status
;  3808  4                !
;  3809  4
;  3810  4                if .RET_ENVAD [STATUS] nequ ZERO    !Test the status
;  3811  4                then
```

```
;    3812  4                    return RET_STATUS + RSE_CODE    !Return a "Response status err" code
;    3813  4                else
;    3814  4                    return .RET_STATUS;             !This ret_status is good or bad
;    3815  4
;    3816  3                end;
;    3817  3
;    3818  2            end;
;    3819  2
;    3820  2        return .RET_STATUS;                         !It won't compile without this here
;    3821  1        end;


                                        .SBTTL  REC.DATA GLOBAL ROUTINE DECLARATIONS
000000  004137  000000G           REC.DATA::
                                    JSR     R1,$SAVE2                       ;                       3600
000004  005746                      TST     -(SP)
000006  004737  000000'            JSR     PC,GET.NSD                      ;                       3660
000012  004737  000146'            JSR     PC,GET.CMD$REF                  ;                       3661
000016  010002                      MOV     R0,R2                           ; *,REF.NUM
000020  013746  000000G             MOV     NSD.SLOT,-(SP)                  ;                       3665
000024  012746  000054              MOV     #54,-(SP)
000030  004737  000000G             JSR     PC,BL$MUL
000034  012760  000034  000000G     MOV     #34,SND.ENVELOPE(R0)
000042  012701  000002G             MOV     #SND.ENVELOPE+2,R1              ;                       3666
000046  060001                      ADD     R0,R1
000050  112711  000001              MOVB    #1,(R1)                         ;                       3667
000054  112761  000002  000001     MOVB    #2,1(R1)                        ;                       3668
000062  010260  000004G             MOV     R2,SND.ENVELOPE+4(R0)           ; REF.NUM,*             3672
000066  005060  000006G             CLR     SND.ENVELOPE+6(R0)              ;                       3673
000072  005060  000010G             CLR     SND.ENVELOPE+10(R0)             ;                       3674
000076  005060  000012G             CLR     SND.ENVELOPE+12(R0)             ;                       3675
000102  112760  000005  000014G     MOVB    #5,SND.ENVELOPE+14(R0)          ;                       3676
000110  105060  000015G             CLRB    SND.ENVELOPE+15(R0)             ;                       3677
000114  005060  000016G             CLR     SND.ENVELOPE+16(R0)             ;                       3678
000120  012760  000170  000020G     MOV     #170,SND.ENVELOPE+20(R0)        ;                       3685
000126  005060  000022G             CLR     SND.ENVELOPE+22(R0)             ;                       3686
000132  012760  000000G  000024G     MOV     #REC.BUF,SND.ENVELOPE+24(R0)    ;                       3690
000140  012701  000026G             MOV     #SND.ENVELOPE+26,R1             ;                       3691
000144  060001                      ADD     R0,R1
000146  105011                      CLRB    (R1)                            ;                       3693
000150  105061  000001              CLRB    1(R1)                           ;                       3694
000154  005060  000030G             CLR     SND.ENVELOPE+30(R0)             ;                       3695
000160  005060  000032G             CLR     SND.ENVELOPE+32(R0)             ;                       3696
000164  005060  000034G             CLR     SND.ENVELOPE+34(R0)             ;                       3697
000170  005060  000036G             CLR     SND.ENVELOPE+36(R0)             ;                       3698
000174  010216                      MOV     R2,(SP)                         ; REF.NUM,*             3707
000176  004737  000054'            JSR     PC,LOAD.OUT$STD.BUF
000202  010001                      MOV     R0,R1                           ; *,REC.BUF$LOC
000204  020127  002001              CMP     R1,#2001                        ; REC.BUF$LOC,*         3709
000210  001002                      BNE     1$
000212  004737  000250'            JSR     PC,DECODE
000216  013700  000000G     1$:     MOV     NSD.SLOT,R0                     ;                       3715
000222  006300                      ASL     R0
```

```
000224  006300                          ASL     R0
000226  063700  000000G                 ADD     SEND.RING,R0
000232  052760  100000  000002          BIS     #100000,2(R0)
000240  017766  000000G 000004          MOV     @RDRX.ADDR,4(SP)        ; *,RC$S.REG                    3719
000246  016600  000004                  MOV     4(SP),R0                ; RC$S.REG,TEMP
000252  104422                  2$:     TRAP    22                      ;                               3753
000254  012716  005670                  MOV     #5670,(SP)              ;                               3760
000260  010246                          MOV     R2,-(SP)                ; REF.NUM,*
000262  010146                          MOV     R1,-(SP)                ; REC.BUF$LOC,*
000264  004737  002116'                  JSR     PC,CTO.WAIT
000270  022626                          CMP     (SP)+,(SP)+
000272  006000                          ROR     R0
000274  103032                          BCC     6$
000276  023727  000000G 000011          CMP     RET.STATUS,#11          ;                               3771
000304  001023                          BNE     5$
000306  004737  002604'                  JSR     PC,GET.DUST.STATUS      ;                               3778
000312  006000                          ROR     R0
000314  103002                          BCC     3$
000316  004737  000250'                  JSR     PC,DECODE
000322  013700  000000G         3$:     MOV     RET.EN$AD,R0            ;                               3780
000326  026037  000024  000000G         CMP     24(R0),PID.SAVE
000334  001404                          BEQ     4$
000336  016037  000024  000000G         MOV     24(R0),PID.SAVE         ;                               3782
000344  000742                          BR      2$                      ;                               3780
000346  012700  000051          4$:     MOV     #51,R0                  ;                               3784
000352  000420                          BR      7$
000354  004737  000250'          5$:     JSR     PC,DECODE               ;                               3793
000360  000734                          BR      2$                      ;                               3760
000362  010100                  6$:     MOV     R1,R0                   ; REC.BUF$LOC,*                 3805
000364  006300                          ASL     R0
000366  006300                          ASL     R0
000370  016037  000002G 000000G         MOV     OUT$STD.BUF+2(R0),RET.EN$AD
000376  016000  000002G                 MOV     OUT$STD.BUF+2(R0),R0    ;                               3810
000402  005760  000016                  TST     16(R0)
000406  001405                          BEQ     8$
000410  012700  000031                  MOV     #31,R0                  ;                               3812
000414  010037  000000G          7$:     MOV     R0,RET.STATUS
000420  000402                          BR      9$                      ;                               3814
000422  013700  000000G          8$:     MOV     RET.STATUS,R0
000426  062706  000006          9$:     ADD     #6,SP                   ;                               3600
000432  000207                          RTS     PC

; Routine Size:  142 words,    Routine Base:  $CODE$ + 4060
; Maximum stack depth per invocation:  9 words

;   3822  1
```

```
;  3823  1     global routine SET_CNTLR_CHAR =          !Sets control characteristics
;  3824  1
;  3825  1     !++
;  3826  1     ! Functional Description :
;  3827  1     !       The SET CONTROLLER CHARACTERISTICS command is used to set host
;  3828  1     !       settable   unit   characteristics  and   obtain  those   unit
;  3829  1     !       characteristics  that are  essential  for  proper class driver
;  3830  1     !       operation.  This    command  never  alters  the  unit's  state
;  3831  1     !       ("unit-online",   "unit-available",  "unit-offline").   It  is
;  3832  1     !       meaningless  to  set  host settable characteristics for a unit
;  3833  1     !       that is "unit-available" or "unit-offline".
;  3834  1     !
;  3835  1     ! Formal Parameters :
;  3836  1     !       none
;  3837  1     !
;  3838  1     ! Implicit Inputs :
;  3839  1     !       NSD_SLOT        This  global storage gets  loaded  by  the routine
;  3840  1     !                       'Get_nsd'  and  in it is stored the next send ring
;  3841  1     !                       descriptor  slot  where the port/controller should
;  3842  1     !                       be polling on and the  place  to put this commands
;  3843  1     !                       command packet.
;  3844  1     !
;  3845  1     !
;  3846  1     ! Implicit Outputs :
;  3847  1     !       none
;  3848  1     !
;  3849  1     ! Completion Codes :
;  3850  1     !       RET_STATUS:     Return status passes back to the calling routine
;  3851  1     !                       the status of the just issued command.
;  3852  1     !
;  3853  1     ! Side Effects :
;  3854  1     !       Any previously defined controller characteristics will possibly
;  3855  1     !       be altered after execution of this command.
;  3856  1     !--
;  3857  1
;  3858  2     begin
;  3859  2
;  3860  2     local
;  3861  2         REF_NUM,                                !Stores unique cmd ref number
;  3862  2         SCC_BUF$LOC,                            !Stores out$standing cmd buffer location
;  3863  2         TEMP;                                   !A place to put read IP register data
;  3864  2
;  3865  2     !
;  3866  2     ! Before we load up the command packet up
;  3867  2     ! with  all this good information get the
;  3868  2     ! next send  descriptor slot and a unique
;  3869  2     ! command reference number.
;  3870  2     !
;  3871  2     GET_NSD ();                                 !Get the next send desc slot
;  3872  2     REF_NUM = GET_CMD$REF ();                   !Get a unique command ref num
;  3873  2     !
;  3874  2     ! UQ Port command envelope Header field definition
;  3875  2     !
```

```
;    3876  2          SND_ENVELOPE [.NSD_SLOT, MSG_LENGTH] = SZ_SCC;      !Load message length
;    3877  2          SND_ENVELOPE [.NSD_SLOT, CREDITS] = ONE;     !Load credit size
;    3878  2          SND_ENVELOPE [.NSD_SLOT, MSG_TYPE] = 0;      !Define message type 'Sequential'
;    3879  2          SND_ENVELOPE [.NSD_SLOT, CONN_ID] = 0;       !Define connection ID 'DUP'
;    3880  2          !
;    3881  2          ! MSCP generic command envelope field definition
;    3882  2          !
;    3883  2          SND_ENVELOPE [.NSD_SLOT, CMD_LREF] = .REF_NUM;      !Load command reference number
;    3884  2          SND_ENVELOPE [.NSD_SLOT, CMD_HREF] = ZERO;  !Zero Hi order cmd ref num
;    3885  2          SND_ENVELOPE [.NSD_SLOT, UN_LUSED] = ZERO;  !Not used in DUP implimentation
;    3886  2          SND_ENVELOPE [.NSD_SLOT, UN_HUSED] = ZERO;  !Not used in DUP implimentation
;    3887  2          SND_ENVELOPE [.NSD_SLOT, OPCODE] = OP_SCC;  !Load this commands op-code
;    3888  2          SND_ENVELOPE [.NSD_SLOT, RSVD] = ZERO;      !Not used field
;    3889  2          SND_ENVELOPE [.NSD_SLOT, MODIFIER] = ZERO;  !Define this commands modifiers
;    3890  2          !
;    3891  2          ! Command specfic command envelope field definition
;    3892  2          !
;    3893  2          SND_ENVELOPE [.NSD_SLOT, MSCP_VER] = ZERO;  !MSCP version
;    3894  2          SND_ENVELOPE [.NSD_SLOT, CTL_FLAGS] = ZERO; !Controller flags
;    3895  2          SND_ENVELOPE [.NSD_SLOT, HOST_TOV] = ZERO;  !Host time out value
;    3896  2          SND_ENVELOPE [.NSD_SLOT, RS$VD] = ZERO;     !Reserved
;    3897  2          SND_ENVELOPE [.NSD_SLOT, T$D_0] = ZERO;     !Time and Date word 0
;    3898  2          SND_ENVELOPE [.NSD_SLOT, T$D_1] = ZERO;     !Time and Date word 1
;    3899  2          SND_ENVELOPE [.NSD_SLOT, T$D_2] = ZERO;     !Time and Date word 2
;    3900  2          SND_ENVELOPE [.NSD_SLOT, T$D_3] = ZERO;     !Time and Date word 3
;    3901  2          SND_ENVELOPE [.NSD_SLOT, CDP_LO] = ZERO;    !Cntlr dep parameter lo word
;    3902  2          SND_ENVELOPE [.NSD_SLOT, CDP_HI] = ZERO;    !Cntlr dep parameter hi wrd
;    3903  2          !
;    3904  2          ! Call   the  load  out$standing  command buffer routine
;    3905  2          ! and  load  this command  into  the buffer.  The return
;    3906  2          ! from this routine will point us to the buffer location
;    3907  2          ! where  this command  is  stored.  Later we can look at
;    3908  2          ! this  location to see if the interrupt service routine
;    3909  2          ! has received and process it.
;    3910  2          !
;    3911  2          SCC_BUF$LOC = LOAD_OUT$STD_BUF (.REF_NUM);  !Load the command
;    3912  2
;    3913  2          if .SCC_BUF$LOC eqlu OBF_CODE then DECODE ();       !Error if buffer is full
;    3914  2
;    3915  2          !
;    3916  2          ! Set the ownership bit to 1 giving this slot
;    3917  2          ! to the port/controller
;    3918  2          !
;    3919  2          SEND_RING [.NSD_SLOT, OWN_BIT] = PORT_OWNED;
;    3920  2          !
;    3921  2          ! Read the IP register to stimulate port polling
;    3922  2          !
;    3923  2          TEMP = .RDRX_ADDR [RCIP, RC_ALL];
;    3924  2          !
;    3925  2          ! Time  out the  port/controller processing  the command.
;    3926  2          !
;    3927  2          ! The   first   test   tests the connections  ability to
;    3928  2          ! respond to  this command without  any errors in the SA
```

```
;  3929  2          ! register and for the command not timing out.
;  3930  2          !
;  3931  2          ! The  second tests the  DUP server for good status.  If
;  3932  2          ! bad status is sent back then an error code is returned
;  3933  2          ! to the calling routine where the routine "decode" will
;  3934  2          ! decode  and take the  appropriate recovery.  The  time
;  3935  2          ! out  routine will loop on delaying and checking the hi
;  3936  2          ! bit of the  first word in  the  out$std_buf for a true.
;  3937  2          ! When true signals us that the interrupt service routine
;  3938  2          ! has received  the endpacket  and  no connection errors
;  3939  2          ! were detected.
;  3940  2          !
;  3941  2
;  3942  2          if CTO_WAIT (3000, .REF_NUM, .SCC_BUF$LOC) then DECODE ();  !Is return an error
;  3943  2
;  3944  2          !
;  3945  2          ! Get  the return envelope  address  from the  out$std_buf
;  3946  2          ! at  this commands  buffer location  and check the packet
;  3947  2          ! for good status error and die if bad status was returned
;  3948  2          !
;  3949  2          RET_EN$AD = .OUT$STD_BUF [.SCC_BUF$LOC, ENV_ADR];   !Get the ret env adr
;  3950  2          !
;  3951  2          ! Now test for good status
;  3952  2          !
;  3953  2
;  3954  2          if .RET_EN$AD [STATUS] nequ ZERO            !Test the status
;  3955  2          then
;  3956  2              return RET_STATUS = RSE_CODE            !Return a "Response status err" code
;  3957  2          else
;  3958  2              return .RET_STATUS;                     !This ret_status is good or bad
;  3959  2
;  3960  1          end;
```

```
                                .SBTTL  SET.CNTLR.CHAR GLOBAL ROUTINE DECLARATIONS
000000  004137  000000G      SET.CNTLR.CHAR::
                                JSR     R1,$SAVE2                       ;                       3823
000004  005746                  TST     -(SP)
000006  004737  000000'          JSR     PC,GET.NSD                      ;                       3871
000012  004737  000146'          JSR     PC,GET.CMD$REF                  ;                       3872
000016  010002                  MOV     R0,R2                           ; *,REF.NUM
000020  013746  000000G          MOV     NSD.SLOT,-(SP)                  ;                       3876
000024  012746  000054          MOV     #54,-(SP)
000030  004737  000000G          JSR     PC,BL$MUL
000034  012760  000040  000000G  MOV     #40,SND.ENVELOPE(R0)
000042  012701  000002G          MOV     #SND.ENVELOPE+2,R1              ;                       3877
000046  060001                  ADD     R0,R1
000050  112711  000001          MOVB    #1,(R1)                         ;                       3878
000054  105061  000001          CLRB    1(R1)                           ;                       3879
000060  010260  000004G          MOV     R2,SND.ENVELOPE+4(R0)           ; REF.NUM,*             3883
000064  005060  000006G          CLR     SND.ENVELOPE+6(R0)              ;                       3884
000070  005060  000010G          CLR     SND.ENVELOPE+10(R0)             ;                       3885
000074  005060  000012G          CLR     SND.ENVELOPE+12(R0)             ;                       3886
```

```
000100  112760  000004  000014G          MOVB    #4,SND.ENVELOPE+14(R0)          ;                               3887
000106  105060  000015G                  CLRB    SND.ENVELOPE+15(R0)             ;                               3888
000112  005060  000016G                  CLR     SND.ENVELOPE+16(R0)             ;                               3889
000116  005060  000020G                  CLR     SND.ENVELOPE+20(R0)             ;                               3893
000122  005060  000022G                  CLR     SND.ENVELOPE+22(R0)             ;                               3894
000126  005060  000024G                  CLR     SND.ENVELOPE+24(R0)             ;                               3895
000132  005060  000026G                  CLR     SND.ENVELOPE+26(R0)             ;                               3896
000136  005060  000030G                  CLR     SND.ENVELOPE+30(R0)             ;                               3897
000142  005060  000032G                  CLR     SND.ENVELOPE+32(R0)             ;                               3898
000146  005060  000034G                  CLR     SND.ENVELOPE+34(R0)             ;                               3899
000152  005060  000036G                  CLR     SND.ENVELOPE+36(R0)             ;                               3900
000156  005060  000040G                  CLR     SND.ENVELOPE+40(R0)             ;                               3901
000162  005060  000042G                  CLR     SND.ENVELOPE+42(R0)             ;                               3902
000166  010216                          MOV     R2,(SP)                         ; REF.NUM,*                     3911
000170  004737  000054'                  JSR     PC,LOAD.OUT$STD.BUF
000174  010001                          MOV     R0,R1                           ; *,SCC.BUF$LOC
000176  020127  002001                  CMP     R1,#2001                        ; SCC.BUF$LOC,*                 3913
000202  001002                          BNE     1$
000204  004737  000250'                  JSR     PC,DECODE
000210  013700  000000G          1$:     MOV     NSD.SLOT,R0                     ;                               3919
000214  006300                          ASL     R0
000216  006300                          ASL     R0
000220  063700  000000G                  ADD     SEND.RING,R0
000224  052760  100000  000002          BIS     #100000,2(R0)
000232  017766  000000G 000004          MOV     @RDRX.ADDR,4(SP)                ; *,RC$S.REG                    3923
000240  016600  000004                  MOV     4(SP),R0                        ; RC$S.REG,TEMP
000244  012716  005670                  MOV     #5670,(SP)                      ;                               3942
000250  010246                          MOV     R2,-(SP)                        ; REF.NUM,*
000252  010146                          MOV     R1,-(SP)                        ; SCC.BUF$LOC,*
000254  004737  002116'                  JSR     PC,CTO.WAIT
000260  022626                          CMP     (SP)+,(SP)+
000262  006000                          ROR     R0
000264  103002                          BCC     2$
000266  004737  000250'                  JSR     PC,DECODE
000272  010100                  2$:     MOV     R1,R0                           ; SCC.BUF$LOC,*                 3949
000274  006300                          ASL     R0
000276  006300                          ASL     R0
000300  016037  000002G 000000G          MOV     OUT$STD.BUF+2(R0),RET.EN$AD
000306  016000  000002G                  MOV     OUT$STD.BUF+2(R0),R0            ;                               3954
000312  005760  000016                  TST     16(R0)
000316  001405                          BEQ     3$
000320  012700  000031                  MOV     #31,R0                          ;                               3956
000324  010037  000000G                  MOV     R0,RET.STATUS
000330  000402                          BR      4$                              ;                               3958
000332  013700  000000G          3$:     MOV     RET.STATUS,R0
000336  062706  000006          4$:     ADD     #6,SP                           ;                               3823
000342  000207                          RTS     PC

; Routine Size:  114 words,    Routine Base:  $CODE$ + 4514
; Maximum stack depth per invocation:  9 words

;   3961  1
```

```
; C 3962  1     %(
; C 3963  1     global routine ON_LINE =        !Makes a unit come online to a host
; C 3964  1
; C 3965  1     !++
; C 3966  1     ! Functional Description :
; C 3967  1     !       The online command is used to bring a unit "unit-online, set
; C 3968  1     !       host settable unit characteristics and obtain those unit
; C 3969  1     !       characteristics that are essential for proper class driver
; C 3970  1     !       operation. The unit is spun-up, if necessary, and is heads
; C 3971  1     !       are loaded prior to returning the online command's end
; C 3972  1     !       message. Host settable characteristics are set exactly as if
; C 3973  1     !       a set unit characteristics command were issued. Host settable
; C 3974  1     !       characteristics are set after the unit has been successfully spun-up
; C 3975  1     !       and any other validity checks have succeded. Note that the unit's
; C 3976  1     !       host settable characteristics are not altered if the unit is already
; C 3977  1     !       "unit-online".
; C 3978  1     !
; C 3979  1     ! Formal Parameters :
; C 3980  1     !       none
; C 3981  1     !
; C 3982  1     ! Implicit Inputs :
; C 3983  1     !       NSD_SLOT        This global storage gets loaded by the routine
; C 3984  1     !                       'Get_nsd' and in it is stored the next send ring
; C 3985  1     !                       descriptor slot where the port/controller should
; C 3986  1     !                       be polling on and the place to put this commands
; C 3987  1     !                       command packet.
; C 3988  1     !
; C 3989  1     ! Implicit Outputs :
; C 3990  1     !       none
; C 3991  1     !
; C 3992  1     ! Completion Codes :
; C 3993  1     !       RET_STATUS:     Return status passes back to the calling routine
; C 3994  1     !                       the status of the just issued command.
; C 3995  1     !
; C 3996  1     ! Side Effects :
; C 3997  1     !       Any previously defined controller characteristics will possibly
; C 3998  1     !       be altered after execution of this command.
; C 3999  1     !--
; C 4000  1
; C 4001  1         begin
; C 4002  1
; C 4003  1         local
; C 4004  1             REF_NUM,                                  !Stores unique cmd ref number
; C 4005  1             ONL_BUF$LOC,                              !Stores out$standing cmd buffer location
; C 4006  1             TEMP;                                     !A place to put read IP register data
; C 4007  1
; C 4008  1         !
; C 4009  1         ! Before we load up the command packet up
; C 4010  1         ! with  all this good information get the
; C 4011  1         ! next send  descriptor slot and a unique
; C 4012  1         ! command reference number.
; C 4013  1         !
; C 4014  1         GET_NSD ();                                   !Get the next send desc slot
```

```
; C 4015  1          REF_NUM = GET_CMD$REF ();                    !Get a unique command ref num
; C 4016  1          !
; C 4017  1          ! UQ Port command envelope Header field definition
; C 4018  1          !
; C 4019  1          SND_ENVELOPE [.NSD_SLOT, MSG_LENGTH] = SZ_ONL;      !Load message length
; C 4020  1          SND_ENVELOPE [.NSD_SLOT, CREDITS] = ONE;    !Load credit size
; C 4021  1          SND_ENVELOPE [.NSD_SLOT, MSG_TYPE] = 0;     !Define message type 'Sequential'
; C 4022  1          SND_ENVELOPE [.NSD_SLOT, CONN_ID] = 0;      !Define connection ID
; C 4023  1          !
; C 4024  1          ! MSCP generic command envelope field definition
; C 4025  1          !
; C 4026  1          SND_ENVELOPE [.NSD_SLOT, CMD_LREF] = .REF_NUM;      !Load command reference number
; C 4027  1          SND_ENVELOPE [.NSD_SLOT, CMD_HREF] = ZERO;  !Zero Hi order cmd ref num
; C 4028  1          SND_ENVELOPE [.NSD_SLOT, UNIT_NUM] = .UNIT_NO;      !Select unit to bring online
; C 4029  1          SND_ENVELOPE [.NSD_SLOT, UN_HUSED] = ZERO;  !Not used in DUP implimentation
; C 4030  1          SND_ENVELOPE [.NSD_SLOT, OPCODE] = OP_ONL;  !Load this commands op-code
; C 4031  1          SND_ENVELOPE [.NSD_SLOT, RSVD] = ZERO;      !Not used field
; C 4032  1          SND_ENVELOPE [.NSD_SLOT, MODIFIER] = ZERO;  !Define this commands modifiers
; C 4033  1          !
; C 4034  1          ! Command specfic command envelope field definition
; C 4035  1          !
; C 4036  1          SND_ENVELOPE [.NSD_SLOT, RSV$D] = ZERO;     !Reserved
; C 4037  1          SND_ENVELOPE [.NSD_SLOT, UNT_FLAGS] = ZERO; !Unit flag field
; C 4038  1          SND_ENVELOPE [.NSD_SLOT, RSVD$0] = ZERO;    !Reserved field
; C 4039  1          SND_ENVELOPE [.NSD_SLOT, RSVD$1] = ZERO;    !Reserved field
; C 4040  1          SND_ENVELOPE [.NSD_SLOT, RSVD$2] = ZERO;    !Reserved field
; C 4041  1          SND_ENVELOPE [.NSD_SLOT, RSVD$3] = ZERO;    !Reserved field
; C 4042  1          SND_ENVELOPE [.NSD_SLOT, RSVD$4] = ZERO;    !Reserved field
; C 4043  1          SND_ENVELOPE [.NSD_SLOT, RSVD$5] = ZERO;    !Reserved field
; C 4044  1          SND_ENVELOPE [.NSD_SLOT, DDP_LO] = ZERO;    !Device dependent parameter
; C 4045  1          SND_ENVELOPE [.NSD_SLOT, DDP_HI] = ZERO;    !Device dependent parameter
; C 4046  1          SND_ENVELOPE [.NSD_SLOT, SHADOW_UNIT] = ZERO;       !Shadow unit
; C 4047  1          SND_ENVELOPE [.NSD_SLOT, COPY_SPEED] = ZERO;        !Copy speed
; C 4048  1          !
; C 4049  1          ! Call   the  load  out$standing  command buffer routine
; C 4050  1          ! and  load  this command  into  the buffer.  The return
; C 4051  1          ! from this routine will point us to the buffer location
; C 4052  1          ! where  this command  is  stored.  Later we can look at
; C 4053  1          ! this  location to see if the interrupt service routine
; C 4054  1          ! has received and process it.
; C 4055  1          !
; C 4056  1          ONL_BUF$LOC = LOAD_OUT$STD_BUF (.REF_NUM);  !Load the command
; C 4057  1
; C 4058  1          if .ONL_BUF$LOC eqlu OBF_CODE then DECODE ();       !Error if buffer is full
; C 4059  1
; C 4060  1          !
; C 4061  1          ! Set the ownership bit to 1 giving this slot
; C 4062  1          ! to the port/controller
; C 4063  1          !
; C 4064  1          SEND_RING [.NSD_SLOT, OWN_BIT] = PORT_OWNED;
; C 4065  1          !
; C 4066  1          ! Read the IP register to stimulate port polling
; C 4067  1          !
```

```
; C 4068  1          TEMP = .RDRX_ADDR [RCIP, RC_ALL];
; C 4069  1          !
; C 4070  1          ! Time  out the  port/controller processing  the command.
; C 4071  1          !
; C 4072  1          ! The   first   test   tests the connections  ability to
; C 4073  1          ! respond to  this command without  any errors in the SA
; C 4074  1          ! register and for the command not timing out.
; C 4075  1          !
; C 4076  1          ! The  second tests the  DUP server for good status.  If
; C 4077  1          ! bad status is sent back then an error code is returned
; C 4078  1          ! to the calling routine where the routine "decode" will
; C 4079  1          ! decode  and take the  appropriate recovery.  The  time
; C 4080  1          ! out  routine will loop on delaying and checking the hi
; C 4081  1          ! bit of the  first word in  the  out$std_buf for a true.
; C 4082  1          ! When true signals us that the interrupt service routine
; C 4083  1          ! has received  the endpacket  and  no connection errors
; C 4084  1          ! were detected.
; C 4085  1          !
; C 4086  1
; C 4087  1          if CTO_WAIT (3000, .REF_NUM, .ONL_BUF$LOC) then DECODE ();  !Is return an error
; C 4088  1
; C 4089  1          !
; C 4090  1          ! Get  the return envelope  address  from the  out$std_buf
; C 4091  1          ! at  this commands  buffer location  and check the packet
; C 4092  1          ! for good status error and die if bad status was returned
; C 4093  1          !
; C 4094  1          RET_EN$AD = .OUT$STD_BUF [.ONL_BUF$LOC, ENV_ADR];   !Get the ret env adr
; C 4095  1          !
; C 4096  1          ! Now test for good status
; C 4097  1          !
; C 4098  1
; C 4099  1          if .RET_EN$AD [STATUS] nequ ZERO            !Test the status
; C 4100  1          then
; C 4101  1              return RET_STATUS = RSE_CODE            !Return a "Response status err" code
; C 4102  1          else
; C 4103  1              return .RET_STATUS;                     !This ret_status is good or bad
; C 4104  1
; C 4105  1          end;
;    4106  1      )%
```

```
;   4107  1     global routine INT$I_SERVICE : INT_LNK$TYP novalue =    !Init sequence interrupt catcher
;   4108  1
;   4109  1     !++
;   4110  1     ! Functional Description :
;   4111  1     !       During the initialization sequence the IE bit is defined to be
;   4112  1     !       a zero.  This means that the host is not requesting interrupts at
;   4113  1     !       the completion of steps 1-3.
;   4114  1     !
;   4115  1     !       Note that no initerrupt will be generated at the completion of
;   4116  1     !       step  4  since  this step rquires only a small number of time.
;   4117  1     !
;   4118  1     !       This interrupt service routine serves to catch any interrupts that the
;   4119  1     !       controller  might  issue   during   the  initialization sequence.  The
;   4120  1     !       interrupt is ignored and control is returned.
;   4121  1     !
;   4122  1     ! Formal Parameters :
;   4123  1     !       none
;   4124  1     !
;   4125  1     ! Implicit Inputs :
;   4126  1     !       none
;   4127  1     !
;   4128  1     ! Implicit Outputs :
;   4129  1     !       none
;   4130  1     !
;   4131  1     ! Completion Codes :
;   4132  1     !       none
;   4133  1     !
;   4134  1     ! Side Effects :
;   4135  1     !       none
;   4136  1     !--
;   4137  1
;   4138  2         begin
;   4139  2         return;
;   4140  1         end;

                                .SBTTL  INT$I.SERVICE GLOBAL ROUTINE DECLARATIONS
000000  000002          INT$I.SERVICE::
                                RTI                                                     ;  4107

; Routine Size:  1 word,        Routine Base:  $CODE$ + 5060
; Maximum stack depth per invocation:  0 words

;   4141  1
```

```
;  4142  1     global routine IS_TIMER (SEQ_NO) =      !Init sequence time out
;  4143  1
;  4144  1     !++
;  4145  1     ! Functional Description :
;  4146  1     !       Steps 1-3 of the init sequence, each are required to
;  4147  1     !       complete  within 10  seconds.  If any of these steps
;  4148  1     !       fails  to complete within that period, this is to be
;  4149  1     !       treated as a host detected fatal error.
;  4150  1     !
;  4151  1     !       This routine will do one us delays for a total  of 10
;  4152  1     !       seconds.  After each delay the step field is examined
;  4153  1     !       to see if this init sequence has completed.
;  4154  1     !
;  4155  1     ! Formal Parameters :
;  4156  1     !       SEQ_NO: Indicated which init step is presently being
;  4157  1     !               performed within the RDRX init sequence.
;  4158  1     !
;  4159  1     ! Implicit Inputs :
;  4160  1     !       none
;  4161  1     !
;  4162  1     ! Implicit Outputs :
;  4163  1     !       none
;  4164  1     !
;  4165  1     ! Completion Codes :
;  4166  1     !       TRUE:   Indicates to the calling routine  that the
;  4167  1     !               indicated init sequence step has timed out.
;  4168  1     !
;  4169  1     !       FALSE:  Indicates to the calling routine that the
;  4170  1     !               indicated init sequence step has not timed
;  4171  1     !               out.
;  4172  1     !
;  4173  1     ! Side Effects :
;  4174  1     !       If the init sequence step times out and an error
;  4175  1     !       is  posted  in  the sa register then the routine
;  4176  1     !       decode will be call.
;  4177  1     !--
;  4178  1
;  4179  2         begin
;  4180  2
;  4181  2         local
;  4182  2             STEP_VAL : word;                    !Temp storage of step value
;  4183  2
;  4184  2         STEP_VAL = ZERO;                        !Make sure the loc is zeroed out
;  4185  2
;  4186  2         !+
;  4187  2         ! Select the step value expected from
;  4188  2         ! this init sequence step.
;  4189  2         !-
;  4190  2
;  4191  2         selectoneu .SEQ_NO of                   !Select the binary step value
;  4192  2             set
;  4193  2
;  4194  2             [0] :
```

```
;  4195  2                 STEP_VAL = %b'0001';                !Step 1 binary value
;  4196  2
;  4197  2             [1] :
;  4198  2                 STEP_VAL = %b'0010';                !Step 2 binary value
;  4199  2
;  4200  2             [2] :
;  4201  2                 STEP_VAL = %b'0100';                !Step 3 binary value
;  4202  2
;  4203  2             [3] :
;  4204  2                 STEP_VAL = %b'1000';                !Step 4 binary value
;  4205  2             tes;
;  4206  2
;  4207  2         !+
;  4208  2         ! Loop  on the 100 micro second delay until
;  4209  2         ! either the expected step field is read in
;  4210  2         ! the SA  register  or  the  step times out.
;  4211  2         !-
;  4212  2
;  4213  2         incru TIM_OUT from 0 to 15000 do            !Delay for 10 seconds
;  4214  3             begin
;  4215  3             DELAY (C_US);                           !Do the delay
;  4216  3             !
;  4217  3             ! Check the step bit to see if it is set yet.
;  4218  3             ! If it is set then return a false indicating
;  4219  3             ! the completion else continue delaying.
;  4220  3             !
;  4221  3
;  4222  3             if .RDRX_ADDR [RCSA, STP_FIELD] eqlu .STEP_VAL then return FALSE;
;  4223  3
;  4224  3             BREAK;                                  !Service any control C's
;  4225  2             end;
;  4226  2
;  4227  2         !+
;  4228  2         ! This step has not completed within the specified
;  4229  2         ! 10  second time interval.  Test the sa register
;  4230  2         ! for any errors posted and report errors if any.
;  4231  2         ! Return  a true to the caller indicating the error.
;  4232  2         !-
;  4233  2
;  4234  2         if .RDRX_ADDR [RCSA, ERR_BIT]               !Is the error bit set
;  4235  2         then
;  4236  3             begin
;  4237  3             RET_STATUS = PFE_CODE;                  !Indicate the port/fatal error code
;  4238  3             DECODE ();                              !Report the error
;  4239  2             end;
;  4240  2
;  4241  2         return TRUE;                                !Return a failure to the caller
;  4242  1         end;


                                        .SBTTL  IS.TIMER GLOBAL ROUTINE DECLARATIONS
000000  004137  000000G          IS.TIMER::
                                        JSR     R1,$SAVE3                       ;                                4142
```

```
000004  162706  000006                          SUB     #6,SP
000010  005002                                  CLR     R2                      ; STEP.VAL                      4184
000012  016600  000020                          MOV     20(SP),R0               ; SEQ.NO,*                      4191
000016  001003                                  BNE     1$                      ;                               4194
000020  012702  000001                          MOV     #1,R2                   ; *,STEP.VAL                    4195
000024  000421                                  BR      4$                      ;                               4191
000026  020027  000001          1$:     CMP     R0,#1                           ;                               4197
000032  001003                                  BNE     2$
000034  012702  000002                          MOV     #2,R2                   ; *,STEP.VAL                    4198
000040  000413                                  BR      4$                      ;                               4191
000042  020027  000002          2$:     CMP     R0,#2                           ;                               4200
000046  001003                                  BNE     3$
000050  012702  000004                          MOV     #4,R2                   ; *,STEP.VAL                    4201
000054  000405                                  BR      4$                      ;                               4191
000056  020027  000003          3$:     CMP     R0,#3                           ;                               4203
000062  001002                                  BNE     4$
000064  012702  000010                          MOV     #10,R2                  ; *,STEP.VAL                    4204
000070  005003          4$:     CLR     R3                      ; TIM.OUT                       4213
000072  013701  000000G         5$:     MOV     L$CPU,R1                ; *,$$TMP2                      4215
000076  001411                 6$:     BEQ     9$
000100  013700  000000G                 MOV     L$DLY,R0                ; *,$$TMP1
000104  001404                                  BEQ     8$
000106  005066  000004          7$:     CLR     4(SP)                   ; $$TMP
000112  005300                                  DEC     R0                      ; $$TMP1
000114  001374                                  BNE     7$
000116  005301                  8$:     DEC     R1                      ; $$TMP2
000120  000766                                  BR      6$
000122  013700  000000G         9$:     MOV     RDRx.ADDR,R0            ;                               4222
000126  016016  000002                  MOV     2(R0),(SP)              ; *,RC$S.REG
000132  010201                                  MOV     R2,R1                   ; STEP.VAL,*
000134  011600                                  MOV     (SP),R0                 ; RC$S.REG,*
000136  006200                                  ASR     R0
000140  006200                                  ASR     R0
000142  006200                                  ASR     R0
000144  000300                                  SWAB    R0
000146  042700  177760                        BIC     #177760,R0
000152  020001                                  CMP     R0,R1
000154  001423                                  BEQ     11$
000156  104422                                  TRAP    22
000160  005203                                  INC     R3                      ; TIM.OUT                       4213
000162  020327  035230                        CMP     R3,#35230               ; TIM.OUT,*
000166  101741                                  BLOS    5$
000170  013700  000000G                 MOV     RDRx.ADDR,R0            ;                               4234
000174  016066  000002  000002          MOV     2(R0),2(SP)             ; *,RC$S.REG
000202  100005                                  BPL     10$
000204  012737  000021  000000G                 MOV     #21,RET.STATUS          ;                               4237
000212  004737  000250'                 JSR     PC,DECODE               ;                               4238
000216  012700  000001          10$:    MOV     #1,R0                   ;                               4179
000222  000401                                  BR      12$
000224  005000          11$:    CLR     R0                      ;                               4142
000226  062706  000006          12$:    ADD     #6,SP
000232  000207                                  RTS     PC
```

```
; Routine Size:  78 words,    Routine Base:  $CODE$ + 5062
; Maximum stack depth per invocation:  9 words

;  4243  1
```

```
;  4244  1     global routine BOOT_RDRX =      !Performs RDRX init sequence
;  4245  1
;  4246  1     !++
;  4247  1     ! Functional Description :
;  4248  1     !       This   routine performs the  initialization  sequence  of the RDRX
;  4249  1     !       RDRX controller.
;  4250  1     !
;  4251  1     !       The initialization procedure serves to:
;  4252  1     !
;  4253  1     !       1.  Identify   the parameters   of the host-resident communications
;  4254  1     !           region to the port.
;  4255  1     !
;  4256  1     !       2.  Provide    a   confidence   check of port/controller  integrity.
;  4257  1     !
;  4258  1     !       3.  Bring  the  port/controller online  to the  host (note that the
;  4259  1     !           devices  attached  to the controller  are  not  thereby brought
;  4260  1     !           online to the class driver.)
;  4261  1     !
;  4262  1     ! Formal Parameters :
;  4263  1     !       none
;  4264  1     !
;  4265  1     ! Implicit Inputs :
;  4266  1     !       ISD_STRUCT      Stores the init sequence read and write data defined
;  4267  1     !       for this program and controller.
;  4268  1     !
;  4269  1     ! Implicit Outputs :
;  4270  1     !       none
;  4271  1     !
;  4272  1     ! Completion Codes :
;  4273  1     !       Success:  Is returned to the calling routine if this initialization
;  4274  1     !                 sequence was executed successfully.
;  4275  1     !
;  4276  1     !       Failure:  Is  returned to the callng routine if this initialization
;  4277  1     !                 sequence was not executed successfully.
;  4278  1     !
;  4279  1     ! Side Effects :
;  4280  1     !       Any DM code that might have been running in the DM machine will be
;  4281  1     !       aborted.
;  4282  1     !
;  4283  1     !       Any outstanding commands or response pertaining to a process using
;  4284  1     !       the controller will be lost.
;  4285  1     !--
;  4286  1
;  4287  2         begin
;  4288  2
;  4289  2         local
;  4290  2             TEMP : word;                            !Temporary storage location
;  4291  2
;  4292  2         !+
;  4293  2         ! The  host begins  the  initialization  sequence
;  4294  2         ! either  by  issuing  a  bus init  or by writing
;  4295  2         ! any  value into the IP register;  the port must
;  4296  2         ! guarantee  that the  host will read zeros in SA
```

```
;  4297  2          ! on  the  next bus  cycle.  Initialization  then
;  4298  2          ! sequences   through  steps 1-4 as per UQSSP.DOC
;  4299  2          ! Version 1.5.
;  4300  2          !
;  4301  2          ! Write  to  the  IP register and  start the init
;  4302  2          ! sequence going.
;  4303  2          !-
;  4304  2
;  4305  2          WRT_RDRX (RCIP, RC_ALL, ONES);              !Begin init sequence
;  4306  2
;  4307  2          !+
;  4308  2          ! This incr loop performs all four steps of the
;  4309  2          ! initialization sequence described above.  The
;  4310  2          ! SA  write  and  read  data is preset into the
;  4311  2          ! structure     ISD_STRUCT    and  stands   for
;  4312  2          ! "Initialization Sequence Data_STRUCT".
;  4313  2          !  -                 -       -
;  4314  2          !
;  4315  2          ! If    a step  time  out error occures the test
;  4316  2          ! invoking  this routine will take the necessary
;  4317  2          ! retry procedure. A  return code  of failure is
;  4318  2          ! returned.
;  4319  2          !
;  4320  2          ! If  any SA register  compare error is detected after
;  4321  2          ! a step completion the routine Decode will decode the
;  4322  2          ! error  and load statistical tables up pertanate data.
;  4323  2          !
;  4324  2          !-
;  4325  2
;  4326  2          incru SEQ_NO from STEP1 to STEP4 do         !Do the four init seq steps
;  4327  3              begin
;  4328  3              !
;  4329  3              ! Wait for the controller to load the SA reg
;  4330  3              ! up with the step data.
;  4331  3              !
;  4332  3
;  4333  3              if IS_TIMER (.SEQ_NO)                   !Did the Controller time out
;  4334  3              then
;  4335  4                  begin
;  4336  4                  !
;  4337  4                  ! DO SOME STAT TABLE UP DATA TO SHOW
;  4338  4                  ! THE TIME OUT
;  4339  4                  !
;  4340  4                  PRINTF (.EMSG_STRUCT [MSG10]);
;  4341  4                  return FAILURE;                     !Notify DRS> init of the failure
;  4342  3                  end;
;  4343  3
;  4344  3          !
;  4345  3          ! The controller did not time out so read the SA register
;  4346  3          ! for  the expected step data and compare it to the good
;  4347  3          ! data stored in ISD_STRUCT.
;  4348  3          !
;  4349  3          ! If the read SA data is not what we expect then  return a
```

```
;  4350  3            ! failure code.
;  4351  3            !
;  4352  3            ! Note that the reserved fields read in the SA register are
;  4353  3            ! or'ed with all ones to mask out the field before compared
;  4354  3            ! to the expected data stored in the structure "ISD_STRUCT".
;  4355  3            !
;  4356  3            TEMP = ((.RDRX_ADDR [RCSA, RC_ALL]) or (.RSVD_STRUCT [.SEQ_NO]));
;  4357  3
;  4358  3            if .TEMP nequ .ISD_STRUCT [.SEQ_NO, ISRD, ISR_ALL]      !Compare read to expected
;  4359  3            then
;  4360  4                begin
;  4361  4                !
;  4362  4                ! Load some satistical table up with
;  4363  4                ! some data to indicate that the init
;  4364  4                ! sequence had some trouble.
;  4365  4                !
;  4366  4                PRINTF (.EMSG_STRUCT [MSG11]);
;  4367  4                return FAILURE;                         !Return a failure code
;  4368  3                end;
;  4369  3
;  4370  3            !
;  4371  3            ! We need to save the micro code
;  4372  3            ! version number if this is step4.
;  4373  3            ! Save the ucode version in UC_VER
;  4374  3            ! if this is step4.
;  4375  3            !
;  4376  3
;  4377  3            if .SEQ_NO eqlu STEP4                       !Is this seq step 4
;  4378  3            then
;  4379  4                begin
;  4380  4                UC_VER = .RDRX_ADDR [RCSA, S4R_VER];        !Save the ucode version
;  4381  3                end;
;  4382  3
;  4383  3            !
;  4384  3            ! This step read data is what we expected so
;  4385  3            ! write the SA register with this steps write
;  4386  3            ! data stored in ISD_STRUCT.
;  4387  3            !
;P 4388  3            WRT_RDRX (RCSA, RC_ALL,                     !Write the SA register
;  4389  3                .ISD_STRUCT [.SEQ_NO, ISWRT, ISW_ALL]);     !Locate the data
;  4390  2            end;
;  4391  2
;  4392  2        !+
;  4393  2        ! The controller initialization sequence was
;  4394  2        ! done successfully so return a success code.
;  4395  2        !-
;  4396  2
;  4397  2        return SUCCESS;
;  4398  1        end;


                                        .SBTTL  BOOT.RDRx GLOBAL ROUTINE DECLARATIONS
00000  004137  000000G                BOOT.RDRX::
```

```
                                        JSR     R1,$SAVE4                       ;                               4244
000004  162706  000006                SUB     #6,SP
000010  017746  000000G               MOV     @RDRX.ADDR,-(SP)                ; *,RC$S.REG                    4305
000014  012700  177777                MOV     #-1,R0                          ; *,RC$M.REG
000020  010077  000000G               MOV     R0,@RDRX.ADDR                   ; RC$M.REG,*
000024  005003                        CLR     R3                              ; SEQ.NO                        4326
000026  010346                1$:     MOV     R3,-(SP)                        ; SEQ.NO,*                      4333
000030  004737  005062'              JSR     PC,IS.TIMER
000034  005726                        TST     (SP)+
000036  006000                        ROR     R0
000040  103007                        BCC     2$
000042  013746  000024G               MOV     EMSG.STRUCT+24,-(SP)            ;                               4340
000046  012746  000001                MOV     #1,-(SP)
000052  010600                        MOV     SP,R0                           ; SP,*
000054  104417                        TRAP    17
000056  000427                        BR      3$                              ;                               4341
000060  013700  000000G       2$:     MOV     RDRX.ADDR,R0                    ;                               4356
000064  016066  000002  000002        MOV     2(R0),2(SP)                     ; *,RC$S.REG
000072  010300                        MOV     R3,R0                           ; SEQ.NO,*
000074  006300                        ASL     R0
000076  016604  000002                MOV     2(SP),R4                        ; RC$S.REG,TEMP
000102  056004  000000G               BIS     RSVD.STRUCT(R0),R4              ; *,TEMP
000106  010302                        MOV     R3,R2                           ; SEQ.NO,*                      4358
000110  006302                        ASL     R2
000112  006302                        ASL     R2
000114  020462  000000G               CMP     R4,ISD.STRUCT(R2)               ; TEMP,*
000120  001410                        BEQ     4$
000122  013746  000026G               MOV     EMSG.STRUCT+26,-(SP)            ;                               4366
000126  012746  000001                MOV     #1,-(SP)
000132  010600                        MOV     SP,R0                           ; SP,*
000134  104417                        TRAP    17
000136  022626                3$:     CMP     (SP)+,(SP)+                     ;                               4367
000140  000440                        BR      6$                              ;                               4360
000142  020327  000003        4$:     CMP     R3,#3                           ; SEQ.NO,*                      4377
000146  001015                        BNE     5$
000150  013700  000000G               MOV     RDRX.ADDR,R0                    ;                               4380
000154  016066  000002  000004        MOV     2(R0),4(SP)                     ; *,RC$S.REG
000162  016600  000004                MOV     4(SP),R0                        ; RC$S.REG,*
000166  042700  177760                BIC     #177760,R0
000172  105037  000000G               CLRB    UC.VER
000176  150037  000000G               BISB    R0,UC.VER
000202  013700  000000G       5$:     MOV     RDRX.ADDR,R0                    ;                               4389
000206  016066  000002  000006        MOV     2(R0),6(SP)                     ; *,RC$S.REG
000214  016201  000002G               MOV     ISD.STRUCT+2(R2),R1             ; *,RC$M.REG
000220  010160  000002                MOV     R1,2(R0)                        ; RC$M.REG,*
000224  005203                        INC     R3                              ; SEQ.NO                        4326
000226  020327  000003                CMP     R3,#3                           ; SEQ.NO,*
000232  101675                        BLOS    1$
000234  012700  000001                MOV     #1,R0                           ;                               4287
000240  000401                        BR      7$
000242  005000                6$:     CLR     R0                              ;                               4244
000244  062706  000010        7$:     ADD     #10,SP
000250  000207                        RTS     PC
```

; Routine Size: 85 words, Routine Base: $CODE$ + 5316
; Maximum stack depth per invocation: 13 words

; 4399 1

```
4400  1     global routine INIT_COM_AREA =  !Inits DUP Protocol commumication area
4401  1
4402  1     !++
4403  1     ! Functional Description :
4404  1     !       After initialization step 3 the port controller clears out
4405  1     !       the communication area's ring buffers.
4406  1     !
4407  1     !       This routine first makes sure that this protocol is accom-
4408  1     !       plished by the port before proceeding.
4409  1     !
4410  1     !       If the port did its part of the protocol then the communic-
4411  1     !       ations  area is initialized as follows:
4412  1     !
4413  1     !         1. Defines from  the  contigious  data storage structure
4414  1     !            "COM_AREA"  the  header  area  address,  receive ring
4415  1     !            address  and  the send ring address (these structures
4416  1     !            are  initially  declared  as reference structures and
4417  1     !            require  an address  to be  defined  as its value per
4418  1     !            BLISS language conventions).
4419  1     !
4420  1     !         2. Clears  the  interrupt  indicators and adaptor  purge
4421  1     !            (ring  base  -1, -2, -3, -4)  defined as  "HEAD_AREA".
4422  1     !
4423  1     !         3. Loads the receive and send descriptors with the values:
4424  1     !               a. Envelope low, high and Q_bus address
4425  1     !               b. Reserved field
4426  1     !               c. Flag bit
4427  1     !               d. Ownership bit
4428  1     !
4429  1     !         4. Load the receive envelope message length field with the
4430  1     !            buffer size in bytes.
4431  1     !
4432  1     !         5. Initialize the Out$standing command buffer to reflect
4433  1     !            that all slots are unused.
4434  1     !
4435  1     ! Formal Parameters :
4436  1     !       none
4437  1     !
4438  1     ! Implicit Inputs :
4439  1     !       HEAD_AREA, RECEIVE_RING, SEND_RING, COM_AREA
4440  1     !
4441  1     ! Implicit Outputs :
4442  1     !                 The  communication area as  a result of this
4443  1     !                 routine will be initialized for host program
4444  1     !                 to remote program communications per DUP and
4445  1     !                 UQSSP specifications.
4446  1     !
4447  1     ! Completion Codes :
4448  1     !       TRUE:   Error code to indicate the port controller has
4449  1     !                 not  fullfilled  its  part of the DUP protocol.
4450  1     !
4451  1     !       FALSE:  An error  code to  indicate the port controller
4452  1     !                 has fullfilled its  part of  the DUP  protocol.
```

```
4453  1     !
4454  1     ! Side Effects :
4455  1     !       none
4456  1     !--
4457  1
4458  2         begin
4459  2
4460  2         !+
4461  2         ! Make sure that the controller has done its part of
4462  2         ! the DUP protocol by clearing out the ring buffers.
4463  2         ! If  the rings are not cleared out then return with
4464  2         ! an error code of true.
4465  2         !-
4466  2
4467  2         incru i from 2 to RING_SIZE - 1 do          !Test all blocks for zeros
4468  2
4469  2             incru j from WRD0 to WRD1 do            !Test all words for zeros
4470  2                 !
4471  2                 ! Test this word for zeros.  If not zeros then exit
4472  2                 ! this routine  with  an "communication area  init"
4473  2                 ! error  code  to  indicate the Protocol violation.
4474  2                 !
4475  2
4476  2                 if .COM_AREA [.i, .j, WORD_REF] nequ ZERO then return CIE_CODF;
4477  2
4478  2         !+
4479  2         ! The  port did its  part of the protocol so now
4480  2         ! define  the address locations of the HEAD_AREA,
4481  2         ! RECEIVE_RING and SEND_RING from the contigious
4482  2         ! storage declared by COM_AREA.
4483  2         !-
4484  2
4485  2         HEAD_AREA = COM_AREA;                       !Define the Header area
4486  2         RECEIVE_RING = COM_AREA [REC_BASE];         !Define the receive ring area
4487  2         SEND_RING = COM_AREA [SND_BASE];            !Define the send ring area
4488  2
4489  2         !+
4490  2         ! Not quite sure if the port has to clear out
4491  2         ! the  header area of the communications area
4492  2         ! so  I'll  clear it  out  here  just in case.
4493  2         !-
4494  2
4495  2         incru i from WRD0 to WRD3 do
4496  2             HEAD_AREA [.i, WORD_REF] = ZEROS;
4497  2
4498  2         !+
4499  2         ! Load up the Send Ring descriptors with an envelope address,
4500  2         ! define  the "Flag bit" to = 1 (interrupt requested), define
4501  2         ! the  "Ownership bit"  to  = 0 (owned by host)  and load the
4502  2         ! Reserved field with zeros (per DUP spec).
4503  2         !-
4504  2
4505  2         incru i from 0 to SND_ALLOCATE - 1 do
```

```
4506  3          begin
4507  3          SEND_RING [.i, LO_EN$AD] = SND_ENVELOPE [.i, CMD_LREF]; !Low-order envelope adress for all sys
4508  3          SEND_RING [.i, HI_EN$AD] = ZERO;        !High-order portion of an 18-bit U/Q bus adrs
4509  3          SEND_RING [.i, QB_EXT] = ZERO;          !Q_bus extention
4510  3          SEND_RING [.i, D$RSVD] = ZERO;          !Reserved field
4511  3          SEND_RING [.i, FLAG_BIT] = SET_FLG;     !Flag bit whose meaning varies depending on dsc state
4512  3          SEND_RING [.i, OWN_BIT] = HOST_OWNED;   !Indicates whether dsc is host or port owned
4513  2          end;
4514  2
4515  2      !+
4516  2      ! Load  up the  Receive Ring descriptors with an envelope
4517  2      ! address, define the "Ownership bit" = 1 (owned by port),
4518  2      ! define the "Flag bit" to = 1 (Interrupts requested) and
4519  2      ! the reserved field set to zeros (per DUP spec).
4520  2      !-
4521  2
4522  2      incru i from 0 to REC_ALLOCATE - 1 do
4523  3          begin
4524  3          RECEIVE_RING [.i, LO_EN$AD] = REC_ENVELOPE [.i, CMD_LREF];
4525  3          RECEIVE_RING [.i, HI_EN$AD] = ZEROS;
4526  3          RECEIVE_RING [.i, QB_EXT] = ZEROS;
4527  3          RECEIVE_RING [.i, D$RSVD] = ZEROS;
4528  3          RECEIVE_RING [.i, FLAG_BIT] = SET_FLG;
4529  3          RECEIVE_RING [.i, OWN_BIT] = PORT_OWNED;
4530  2          end;
4531  2
4532  2      !+
4533  2      ! Reset the communications area pointer to
4534  2      ! their initial state.
4535  2      !-
4536  2
4537  2      NRD_SLOT = -1;                              !Start ring pointer at zero
4538  2      NSD_SLOT = -1;                              !Start ring pointer at zero
4539  2      NXT_CRN = ZERO;                             !Start unique cmd ref num at one
4540  2
4541  2      !+
4542  2      ! Set the response envelope message length size equal
4543  2      ! to  the buffer  size in bytes starting at text + 0.
4544  2      !-
4545  2
4546  2      incru i from 0 to REC_ALLOCATE - 1 do
4547  2          REC_ENVELOPE [.i, MSG_LENGTH] = RB_SIZE*2;      !Convert to bytes before loading
4548  2
4549  2      !+
4550  2      ! Init the outstanding command buffer as follows:
4551  2      !   1.  Indicate that all slots are unused by loading
4552  2      !       the unique value %o'100000'.
4553  2      !   2.  Clear the envelope adrs words to zero.
4554  2      !-
4555  2
4556  2      incru i from 0 to REC_ALLOCATE - 1 do
4557  3          begin
4558  3          OUT$STD_BUF [.i, CMD_WRD] = %o'100000'; !Define the slot as unused
```

```
4559  3          OUT$STD_BUF [.i, ENV_ADR] = ZERO;       !Clear out the envelope adrs field
4560  2          end;
4561  2
4562  2      !+
4563  2      ! No  errors detected by this routine so
4564  2      ! return with an non-error code of false.
4565  2      !-
4566  2
4567  2      return PAS_CODE;
4568  1      end;


                                .SBTTL  INIT.COM.AREA GLOBAL ROUTINE DECLARATIONS
00000  004137  000000G          INIT.COM.AREA::
                                JSR     R1,$SAVE3                       ;                       4400
00004  012701  000004           MOV     #4,R1                           ; *,I                   4467
00010  005002                   1$:     CLR     R2                      ; J                     4469
00012  010100                   2$:     MOV     R1,R0                   ; I,*                   4476
00014  060200                           ADD     R2,R0                   ; J,*
00016  006300                           ASL     R0
00020  005760  000000G                  TST     COM.AREA(R0)
00024  001403                           BEQ     3$
00026  012700  000001                  MOV     #1,R0
00032  000207                           RTS     PC
00034  005202                   3$:     INC     R2                      ; J                     4469
00036  020227  000001                  CMP     R2,#1                   ; J,*
00042  101763                           BLOS    2$
00044  062701  000002                  ADD     #2,R1                   ; *,I                   4467
00050  020127  000022                  CMP     R1,#22                  ; I,*
00054  101755                           BLOS    1$
00056  012737  000000G 000000G        MOV     #COM.AREA,HEAD.AREA             ;               4485
00064  012737  000010G 000000G        MOV     #COM.AREA+10,RECEIVE.RING       ;               4486
00072  012737  000030G 000000G        MOV     #COM.AREA+30,SEND.RING          ;               4487
00100  005000                           CLR     R0                      ; I                     4495
00102  010001                   4$:     MOV     R0,R1                   ; I,*                   4496
00104  063701  000000G                  ADD     HEAD.AREA,R1
00110  005011                           CLR     (R1)
00112  062700  000002                  ADD     #2,R0                   ; *,I                   4495
00116  020027  000006                  CMP     R0,#6                   ; I,*
00122  101767                           BLOS    4$
00124  005003                           CLR     R3                      ; I                     4505
00126  010301                   5$:     MOV     R3,R1                   ; I,*                   4507
00130  006301                           ASL     R1
00132  006301                           ASL     R1
00134  010102                           MOV     R1,R2
00136  063702  000000G                  ADD     SEND.RING,R2
00142  010346                           MOV     R3,-(SP)                ; I,*
00144  012746  000054                  MOV     #54,-(SP)
00150  004737  000000G                  JSR     PC,BL$MUL
00154  062700  000004G                  ADD     #SND.ENVELOPE+4,R0
00160  010012                           MOV     R0,(R2)
00162  010100                           MOV     R1,R0                   ;                       4508
00164  063700  000000G                  ADD     SEND.RING,R0
```

```
00170  062700  000002                    ADD     #2,R0
00174  012710  040000                    MOV     #40000,(R0)             ;                       4512
00200  022626                            CMP     (SP)+,(SP)+             ;                       4506
00202  005203                            INC     R3                      ; I                     4505
00204  020327  000003                    CMP     R3,#3                   ; I,*
00210  101746                            BLOS    5$
00212  005003                            CLR     R3                      ; I                     4522
00214  010301                    6$:     MOV     R3,R1                   ; I,*                   4524
00216  006301                            ASL     R1
00220  006301                            ASL     R1
00222  010102                            MOV     R1,R2
00224  063702  000000G                   ADD     RECEIVE.RING,R2
00230  010300                            MOV     R3,R0                   ; I,*
00232  000300                            SWAB    R0
00234  106000                            RORB    R0
00236  006000                            ROR     R0
00240  006000                            ROR     R0
00242  142700  000077                    BICB    #77,R0
00246  062700  000004G                   ADD     #REC.ENVELOPE+4,R0
00252  010012                            MOV     R0,(R2)
00254  010100                            MOV     R1,R0                   ;                       4525
00256  063700  000000G                   ADD     RECEIVE.RING,R0
00262  062700  000002                    ADD     #2,R0
00266  012710  140000                    MOV     #140000,(R0)            ;                       4529
00272  005203                            INC     R3                      ; I                     4522
00274  020327  000003                    CMP     R3,#3                   ; I,*
00300  101745                            BLOS    6$
00302  012737  177777  000000G           MOV     #-1,NRD.SLOT            ;                       4537
00310  012737  177777  000000G           MOV     #-1,NSD.SLOT            ;                       4538
00316  105037  000000G                   CLRB    NXT.CRN                 ;                       4539
00322  005000                            CLR     R0                      ; I                     4546
00324  012760  000074  000000G    7$:     MOV     #74,REC.ENVELOPE(R0)    ; *,*(I)                4547
00332  062700  000100                    ADD     #100,R0                 ; *,I                   4546
00336  020027  000300                    CMP     R0,#300                 ; I,*
00342  101770                            BLOS    7$
00344  005000                            CLR     R0                      ; I                     4556
00346  012760  100000  000000G    8$:     MOV     #-100000,OUT$STD.BUF(R0)  ; *,*(I)              4558
00354  005060  000002G                   CLR     OUT$STD.BUF+2(R0)       ; *(I)                  4559
00360  062700  000004                    ADD     #4,R0                   ; *,I                   4556
00364  020027  000014                    CMP     R0,#14                  ; I,*
00370  101766                            BLOS    8$
00372  005000                            CLR     R0                      ;                       4458
00374  000207                            RTS     PC                      ;                       4400

Routine Size:  127 words,    Routine Base:  $CODE$ + 5570
Maximum stack depth per invocation:  7 words
```

```
4569  1        global routine copy (fff, ttt, length) = !copy a string
4570  2            begin
4571  2                incru i from 0 to .length by 2 do
4572  3                    .ttt + .i = .(.fff + .i)
4573  1            end;
```

```
                                        .SBTTL  COPY GLOBAL ROUTINE DECLARATIONS
0000  004137  000000G             COPY::  JSR     R1,$SAVE2                       ;                       4569
0004  005001                             CLR     R1                              ; I                     4570
0006  000411                             BR      2$
0010  010102                         1$: MOV     R1,R2                           ; I,*                   4572
0012  066602  000012                     ADD     12(SP),R2                       ; TTT,*
0016  010100                             MOV     R1,R0                           ; I,*
0020  066600  000014                     ADD     14(SP),R0                       ; FFF,*
0024  011012                             MOV     (R0),(R2)
0026  062701  000002                     ADD     #2,R1                           ; *,I                   4570
0032  020166  000010                 2$: CMP     R1,10(SP)                       ; I,LENGTH
0036  101764                             BLOS    1$
0040  012700  177777                     MOV     #-1,R0                          ;                       4569
0044  000207                             RTS     PC

Routine Size:  19 words,    Routine Base:  $CODE$ + 6166
Maximum stack depth per invocation:  4 words

  4574  1
  4575  1     end
  4576  1
  4577  0     eludom


                                OTS external references
                                        .GLOBL  $SAVE4, $SAVE3, $SAVE2, BL$MUL


                                PSECT SUMMARY

  Psect Name                    Words     Attributes
   $CODE$                        1614       RO , I  , LCL,  REL,  CON


                        Library Statistics

                                ---------- Symbols ----------      Pages       Processing
  File                          Total     Loaded    Percent      Mapped      Time

DISK$USER2:[YOUNG.FMT]ZRQBB0.L16;4      358        313         87          18        00:00.1
```

COMMAND QUALIFIERS

```
BLISS/PDP11/LIST ZRQB4.B16/EN:NOEIS/SOURCE-PAGE:53

Size:          1614 code + 0 data words
Run Time:        01:09.3
Elapsed Time:    03:21.1
Lines/CPU Min:    3964
Lexemes/CPU-Min: 19086
Memory Used:  319 pages
Compilation Complete
```

```
0001  0     MODULE ZRQB5    (%TITLE 'RDRx DISK FORMATTER'
0002  0                     IDENT = 'REV C PATCH 0',
0003  0                     ADDRESSING_MODE (ABSOLUTE),
0004  0                     ENVIRONMENT (NOEIS)) =
0005  1     BEGIN
0006  1
0007  1     LIBRARY 'ZRQBB0.L16';           !Define RDRx Formatter library
0008  1
0009  1     REQUIRE 'BLSMAC.REQ';           !Define Bliss macro require file
1545  1
1546  1     %SBTTL 'LAST ADDRESS AND SETUP SECTION'
1547  1
1548  1
```

```
1549  1
1550  1     !+
1551  1     ! The LASTAD macro must be the final statement (except .end) in  a  pro-
1552  1     ! gram.  The call generates an even address reflecting the first word of
1553  1     ! memory unused by the program.
1554  1     !-
1555  1
1556  2     LASTAD
1557  2
1558  2     !+
1559  2     ! Hardcoded P-TABLES
1560  2     !
1561  2     ! These optional hardware P-TABLES are located  (when  present)  between
1562  2     ! the  "LASTAD" macro and the ".END" statement.  These hardware P-TABLES
1563  2     ! are above and beyond the default hardware P-TABLE located in the  main
1564  2     ! body  of the program.  These P-TABLES wind up appended to the BIN file
1565  2     ! of the diagnostic, just as though the supervisor or the  "SETUP"  utility
1566  2     ! had  built them there.  Thus the diagnostic can be "pre-parameterized"
1567  2     ! by the programmer.
1568  2     !
1569  2     ! If this hardcoded P_TABLE section is not wanted then define "number" in
1570  2     ! the BGNSETUP macro to zero and omitt BGNTAB and ENDTAB macros.
1571  2     !
1572  2     ! Coding sample is as follows:
1573  2     !
1574  2     !     LASTAD
1575  2     !
1576  2     !     BGNSETUP (Number)          !Number of P-TABLES
1577  2     !
1578  2     !     BGNPTAB
1579  2     !     (DATA)
1580  2     !     (DATA)
1581  2     !     (DATA)
1582  2     !     ENDPTAB
1583  2     !
1584  2     !     BGNPTAB
1585  2     !     (DATA)
1586  2     !     (DATA)
1587  2     !     (DATA)
1588  2     !     ENDPTAB
1589  2     !
1590  2     !     ENDSETUP
1591  2     !     .END
1592  2     !-
1593  2
1594  2     BGNSETUP (1);
P 1595  2          BGNPTAB
P 1596  2             %O'172150',     %O'154',        4,      0
1597  2          ENDPTAB
1598  1     ENDSETUP

                                .TITLE  ZRQB5 RDRx DISK FORMATTER
```

```
                                        .IDENT  /REV C /
                                        .ENABL  AMA


00000                                   .PSECT  $XYZ$,  RO
00000  000020'                    BL$LAS::.WORD   T$FREE
00002  00000C                             .WORD   <<T$FREE-<BL$LAS+4>>/2>
00004  000000                    P.AAA:   .WORD   0
00006  000004                             .WORD   4                              ; Plit count word
00010  172150                    P.AAB:   .WORD   -5630
00012  000154                             .WORD   154
00014  000004                             .WORD   4
00016  000000                             .WORD   0
00020  000000                    T$FREE::.WORD    0


       000004'                  L$LAST=          BL$LAS+4
       000001                   T$PTHV=          1
       000004'                  $$LAS1=          P.AAA
       000010'                  $REM2=           P.AAB


                                        .SBTTL  $END.LINK LAST ADDRESS AND SETUP SECTION
00000  000207                    $END.LINK::
                                        RTS     PC                             ;                                       1544

Routine Size:  1 word,        Routine Base:  $XYZ$ + 0022
Maximum stack depth per invocation:  0 words

 1599  1     END
 1600  0     ELUDOM

                                PSECT SUMMARY

Psect Name                      Words     Attributes
  $XYZ$                            10       RO ,  I  ,  LCL,  REL,  CON


                        Library Statistics

                                ------- Symbols -------           Pages       Processing
File                            Total   Loaded   Percent          Mapped      Time

DISK$USER2:[YOUNG.FMT]ZRQBB0.L16;4   358      53        14          18        00:00.1
```

COMMAND QUALIFIERS

BLISS/PDP11/LIST ZRQB5.B16/EN:NOEIS/SOURCE=PAGE:53

```
ize:          1 code + 9 data words
un Time:        00:07.3
lapsed Time:    00:22.9
ines/CPU Min:   13186
exemes/CPU-Min: 66585
emory Used:  106 pages
ompilation Complete
```

```
                                ; THIS SHORT SUBROUTINE ALLOWS THE FORMATTER TO WORK ON EITHER
                                ; AN F-11 OR J-11 BY LOADING EITHER A 1 OR A 5 INTO GLOBAL LOCATION
                                ; L$CPU. THE DELAY COUNTER IS MULTIPLIED BY THIS VALUE, WHICH IS
                                ; THE RATIO OF THE CPU SPEEDS.

            000005              ORION   =       5               ;RELATIVE CPU SPEEDS
            000001              LCP     =       1

            000003              LCPTYP  =       3               ;KDF11 PROCESSOR TYPE CODE

000000                          .PSECT  $CPU$

000000                          SETCPU::
000000  010046                          MOV     R0,-(SP)
000002  012767  000001  000000G         MOV     #LCP,L$CPU
000010  000007                          MFPT                    ;
000012  020027  000003                  CMP     R0,#LCPTYP      ;11-23?
000016  001403                          BEQ     10$             ;YES, MULTIPLIER OF 1
000020  012767  000005  000000G         MOV     #ORION,L$CPU    ;NO, SET MULTIPLIER OF 5
000026  012600                  10$:    MOV     (SP)+,R0
000030  000207                          RTS     PC
            000001              .END
```

IN. MACRO M1200 05-MAR-84 14:44 PAGE 1-1

SEQ 0202

BOL TABLE

= 000001 LCPTYP= 000003 L$CPU = ****** GX ORION = 000005 SETCPU 000000G 002

3S. 000000 000
000000 001
)$ 000032 002
)RS DETECTED: 0

TUAL MEMORY USED: 45 WORDS (1 PAGES)
AMIC MEMORY: 20060 WORDS (77 PAGES)
'SED TIME: 00:00:06
'PU.OBJ,SETCPU.LIS/-SP=SETCPU.MAC

```
TLE 'RDRX FORMATTER LIBRARY MODULE'                                    SEQ 0203

ttl 'MACRO DEFINITIONS'

ro
 WRT_RDRX (O, FIELDNAM, IMAGE) =

 begin
 local
     RC$M_REG;
 RC$M_REG = .RDRX_ADDR [0, RC_ALL];
 RC$M_REG <%fieldexpand (FIELDNAM)> = IMAGE;
 (.RDRX_ADDR + %upval*0) = .RC$M_REG;
 end%,

up Protocol Macros Declarations

 REC_BASE =
     HDR_SIZ, 0, 0, 16, 0%,
 SND_BASE =
     HDR_SIZ + REC_ALLOCATE , 0, 0, 16, 0%,

acro to clear out the DUP send data text buffer
fore requesting input form operator.  This  by
fault puts a null byte at the end of the ascii
put.

CLR_SBUF =
     incru i from 0 to SNDB_SIZE - 1 do
         SND_BUF [.i] = ZEROS;%,

acro to clear out the DUP send data text buffer
fore requesting input form operator.  This  by
fault puts a null byte at the end of the ascii
put.

CLR_RBUF =
     incru i from 0 to ESZ_RED- 1 do
         msgadr+i = ZEROS;%,

neral purpose word reference field select

 WORD_REF =
     0, 16, 0%;
```

```
:tl 'SUPERVISOR DEFINED LITERALS'                                    SEQ 0204

:ral

:T DIFINITIONS

BIT15 = %o'100000',
BIT14 = %o'40000',
BIT13 = %o'20000',
BIT12 = %o'10000',
BIT11 = %o'4000',
BIT10 = %o'2000',
BIT09 = %o'1000',
BIT08 = %o'400',
BIT07 = %o'200',
BIT06 = %o'100',
BIT05 = %o'40',
BIT04 = %o'20',
BIT03 = %o'10',
BIT02 = %o'4',
BIT01 = %o'2',
BIT00 = %o'1',
!
BIT9 = BIT09,
BIT8 = BIT08,
BIT7 = BIT07,
BIT6 = BIT06,
BIT5 = BIT05,
BIT4 = BIT04,
BIT3 = BIT03,
BIT2 = BIT02,
BIT1 = BIT01,
BIT0 = BIT00,

ENT FLAG DEFINITIONS
EF32:EF17 RESERVED FOR SUPERVISOR TO PROGRAM COMMUNICATION
EF_START = 32,                  ! START COMMAND WAS ISSUED
EF_RESTART = 31,                ! RESTART COMMAND WAS ISSUED
EF_CONTINUE = 30,               ! CONTINUE COMMAND WAS ISSUED
EF_NEW = 29,                    ! A NEW PASS HAS BEEN STARTED
EF_PWR = 28,                    ! A POWER-FAIL/POWER-UP OCCURRED

IORITY LEVEL DEFINITIONS

PRI07 = %o'340',
PRI06 = %o'300',
PRI05 = %o'240',
PRI04 = %o'200',
PRI03 = %o'140',
PRI02 = %o'100',
PRI01 = %o'40',
PRI00 = %o'0',
```

SEQ 0205

RATOR FLAG BITS

```
EVL  = %o'4'.
.OT  = %o'10'.
ADR  = %o'20'.
IDU  = %o'40'.
ISR  = %o'100'.
JAM  = %o'200'.
BOE  = %o'400'.
PNT  = %o'1000'.
PRI  = %o'2000'.
IXE  = %o'4000'.
IBE  = %o'10000'.
IER  = %o'20000'.
.OE  = %o'40000'.
HOE  = %o'100000'.
```

```
!tl 'FORMATTER DEFINED LITERALS'

!ssage selection literials

!MSG0 = 0.                              !Select message 0
!MSG1 = 1.                              !Select message 1
!MSG2 = 2.                              !Select message 2
!MSG3 = 3.                              !Select message 3
!MSG4 = 4.                              !Select message 4
!MSG5 = 5.                              !Select message 5
!MSG6 = 6.                              !Select message 6
!MSG7 = 7.                              !Select message 7
!MSG8 = 8.                              !Select message 8
!MSG9 = 9.                              !Select message 9
!MSG10 = 10.                            !Select message 10
!MSG11 = 11.                            !Select message 11
!MSG12 = 12.                            !Select message 12
!MSG13 = 13.                            !Select message 13
!MSG14 = 14.                            !Select message 14
!MSG15 = 15.                            !Select message 15
!MSG16 = 16.                            !Select message 16
!MSG17 = 17.                            !Select message 17
!MSG18 = 18.                            !Select message 18
!MSG19 = 19.                            !Select message 19
!MSG20 = 20.                            !Select message 20
!MSG21 = 21.                            !Select message 21
!MSG22 = 22.                            !Select message 22
!MSG23 = 23.                            !Select message 23
!MSG24 = 24.                            !Select message 24
!MSG25 = 25.                            !Select message 25
!MSG26 = 26.                            !Select message 26
!MSG27 = 27.                            !Select message 27
!MSG28 = 28.                            !Select message 28
!MSG29 = 29.                            !Select message 29
!MSG30 = 30.                            !Select message 30
```

```
!cellanious literals

FAILURE = 0,                            !Failure return code
!SUCCESS = 1,                           !Success return code
!BLK0 = 0,                              !Selects block 0 of struct
!BLK1 = 1,                              !Selects block 1 of struct
!BLK2 = 2,                              !Selects block 2 of struct
!BLK3 = 3,                              !Selects block 3 of struct
!WRD0 = 0,                              !Selects word 0 of block
!WRD1 = 1,                              !Selects word 1 of block
!WRD2 = 2,                              !Selects word 2 of block
!WRD3 = 3,                              !Selects word 3 of block
!WRD4 = 4,                              !Selects word 4 of block
!WRD5 = 5,                              !Selects word 5 of block
!WRD6 = 6,                              !Selects word 6 of block
!ISRD = 0,                              !Select the read word (1st) in ISD_STRUCTURE
!ISWRT = 1,                             !Select the write word (2nd) in ISD_STRUCTURE
!STEP1 = 0,                             !Initialization sequence step one
!STEP4 = 3,                             !Initialization sequence step two
TRUE = 1,                               !True indicator
FALSE = 0,                              !False indicator
SET_FLG = 1,                            !Set the indicated flag to one
CLR_FLG = 0,                            !Clear the indicated flag to zero
ONE = 1,                                !Ones data type
ZERO = 0,                               !Zero data type
ONES = %o'177777',                      !All ones data type
ZEROS = 0,                              !All zeros data type
%LIMIT = 16,                            !Allowable number of units to format
DM_SIZE = 8192,                         !Size of DM buffer size
FCT_SIZE = 4096,                        !Size of fct buffer size

Command opcode bits 6 and 7 indicate the type of message
(command, end or attention message).  The following literials
define these field values.

CMD$MSG = %b'00',                       !Command opcode message type
ATT$MSG = %b'01',                       !Attention opcode msg type
END$MSG = %b'10',                       !End opcode message type
```

```
!
! Error code literials
!
PAS_CODE = %o'00',                      !Pass code
CIE_CODE = %o'01',                      !Communication area init error
CTO_CODE = %o'11',                      !Controller time out error
PFE_CODE = %o'21',                      !Port fatal error
RSE_CODE = %o'31',                      !response status error
PVE_CODE = %o'41',                      !Host/Controller out of sequence
RPD_CODE = %o'51',                      !Remote program died error code
PSE_CODE = %O'61',                      !Port/host synchronious error
MLE_CODE = %O'71',                      !Message length error code
UEC_CODE = %O'101',                     !Unknown end code error
APR_CODE = %O'201',                     !Adaptor purge reguest error
UIN_CODE = %O'301',                     !Unknown interrupt error code
ATN_CODE = %O'401',                     !Attention msg received
CMD_CODE = %O'501',                     !Command msg received
SEX_CODE = %O'601',                     !Serious exception error received
IVC_CODE = %O'701',                     !Invalid command error received
UMT_CODE = %O'1001',                    !Unknown message type
OBF_CODE = %O'2001',                    !Out standing buffer full error
OSE_CODE = %O'3001',                    !Out$std_buffer out of sync error
UMN_CODE = %O'4001',                    !Unknown message number
FRE_CODE = %O'5001',                    !File read error code from load media
!
! Dup Protocol literials
!
! Note:
!    The values assigned to the literials
!    REC_SIZ, SND_SIZ are represented  in
!    powers of 2 notation per DUP spec.
!
!    By redefining these two literials the
!    communications area allocation and init
!    sequence is automatically handled and
!    no other parameter modification is needed.
!    Further more the send and receive rings
!    can be of different lengths.  However the
!    maximum value allowed is 7 (2**7 = 128 slots)
!    per DUP spec.
!
REC_SIZ = 2,                            !Define number of receive slots
SND_SIZ = 2,                            !Define number of send slots
```

```
!
! This one is tricky: (Hdr_siz)
!       The communication area is defined to be a
!       contigious Blockvector (two words per block)
!       data segment of size, Rec_allocate + Snd_allocate +
!       Hdr_siz.  The two words per block coming from the
!       two words needed to represent the ring descriptors.
!
!       Hdr_siz is then really * 2 or 4 words of storage
!       to represent the interrupt and purge indicators.
!
    HDR_SIZ = 2,                                !Define com area header size 4 words
!
!
    REC_ALLOCATE = 1+REC_SIZ,                   !Define receive ring allocation
    SND_ALLOCATE = 1+SND_SIZ,                   !Define send ring allocation
    !
    ! Ring_size equals the total number of ring descriptors
    ! (number  of  2  word  blocks) allocated   within  the
    ! communitcations area.
    !
    RING_SIZE = REC_ALLOCATE + SND_ALLOCATE + HDR_SIZ,
!
! The  RB_SIZ  by  definition, DUP spec, must be
! a minimum of 60 bytes long (30 words) 64 bytes
! overall.  The  additional  2  words for the UQ
! port  information  is  accounted for in azkel2
! when the storage is allocated.
!
    RB_SIZE = 30,                               !Number of "words" in response buffer
!
! The biggest command I'll ever send will
! be (I hope) will be 40 bytes, 42 overall,
! and again the UQ port 2 words is allowed
! for in azkel2.
!
    SB_SIZE = 20,                               !Number of "words" in send buffer
                                                !
    SNDB_SIZE = 37,                             !Send DUP cmd text buffer size
    SNDB_COUNT = 16,                            !Byte count to send to RDRX Formatter
    RECB_SIZE = 120,                            !Receive DUP cmd text buffer size
    PORT_OWNED = 1,                             !Descriptor owned by port
    HOST_OWNED = 0,                             !Descriptor owned by host
```

```
!
! Delay literal values
!
! An argument of one results in a 100us
! delay.
!
! A roman numeral notation is used to
! denote values of delay arguments.
! The notation is as follows:
!
! I (1), V (5), X (10), L (50), C (100),
! D (500), M (1000)
!
! Any symbol following another of equal or greater
! value adds to its value, as II = 2, XI = 11.
!
! Any symbol proceeding one of greater value subtracts
! from the second and the remainder added to the first
! as XIV = 14, LIX = 59.
!
    ONE_SEC = 10000,                        !One second delay argument
    C_US = 1,                               !100 micro sec delay argument
    CC_US = 2,                              !200 micro sec delay argument
    CCC_US = 3,                             !300 micro sec delay argument
    XC_US = 4,                              !400 micro sec delay argument
    D_US = 5,                               !500 micro sec delay argument
!
! RDRX register offsets
!
    RCIP = 0,
    RCSA = 1,
!
! Command packet opcodes
! (See note following)
!
!     OP_ABO = %o'1',                       !Abort command
!     OP_ACC = %o'20',                      !Access command
!     OP_AVL = %o'10',                      !Available command
!     OP_CCD = %o'21',                      !Compare controller data command
!     OP_CMP = %o'40',                      !Compare host data command
!     OP_DAP = %o'13',                      !Determine access path
!     OP_ERS = %o'22',                      !Erase command
!     OP_FLU = %o'23',                      !Flush command
!     OP_GCS = %o'02',                      !Get command status command
!     OP_GUS = %o'03',                      !Get unit status command
!     OP_ONL = %o'11',                      !Online command
!     OP_RD = %o'41',                       !Read command
!     OP_RPL = %o'24',                      !Replace command
      OP_SCC = %o'04',                      !Set controller characteristics command
!     OP_SUC = %o'12',                      !Set unit characteristics command
!     OP_WR = %o'42',                       !Write command
!     OP_MRD = %o'30',                      !Maintenance read command
!     OP_MWR = %o'31',                      !Maintenance write command
```

```
!
! End message and serious exception encodes
! (see note following)
!
OP_END = %o'200',                       !End packet flag
OP_SEX = %o'7',                         !Serious exception end packet
!
! MSCP  Attention message endcodes
!
OP_AVA = %o'100',                       !Available attention message
OP_DUP = %o'101',                       !Duplicate unit number attention message
OP_ACP = %o'102',                       !Access path attention message
OP_RLC = %o'103',                       !Reset command limit attention message
!
!  The following are the dup op-code commands
!
OP_GDS = %o'1',                         !Get dust status
OP_ESP = %o'2',                         !Execute supplied program
OP_ELP = %o'3',                         !Execute local program
OP_SED = %o'4',                         !Send data
OP_RED = %o'5',                         !Receive data
OP_ABT = %o'6',                         !Abort program
!+
!  NOTE:
!  -----
!
!  End message opcodes  (also called endcodes) are formed by adding  the end
!  message  flag  to  the  command opcode.  For  example, a READ commands end
!  message contains the value OP.RED + OP.END in its opcode field. The Invalid
!  command  end  message contains just the end message flag (i.e., OP.END) in
!  its opcode field.  The serious exception opcode shown above (i.e. OP.SEX +
!  OP.END) in its opcode field.
!
!  Commands opcode bits 6 and 7 indicate the type of message (command, end or
!  attention  message.  Command opcodes bits 3 through 5 indicate the command
!  catagory  (immediate, sequential  or no-sequential) and whether or not the
!  command includes a buffer descriptor.
!
!  See MSCP document appendix "A-1 NOTE:" for more information on this topic.
!-
!
! DUP endcode message types
!
EOP_GDS = OP_GDS + OP_END,              !Get dust status
EOP_ESP = OP_ESP + OP_END,              !Execute supplied program
EOP_ELP = OP_ELP + OP_END,              !Execute local program
EOP_RED = OP_RED + OP_END,              !Receive data
EOP_SED = OP_SED + OP_END,              !Send data
EOP_ABT = OP_ABT + OP_END,              !Abort program
```

```
!
! MSCP endcode messag types
!
!    EOP_ABO = OP_ABO + OP_END,                    !Abort command
!    EOP_ACC = OP_ACC + OP_END,                    !Access command
!    EOP_AVL = OP_AVL + OP_END,                    !Available command
!    EOP_CCD = OP_CCD + OP_END,                    !Compare controller data command
!    EOP_CMP = OP_CMP + OP_END,                    !Compare host data command
!    EOP_ERS = OP_ERS + OP_END,                    !Erase command
!    EOP_FLU = OP_FLU + OP_END,                    !Flush command
!    EOP_GCS = OP_GCS + OP_END,                    !Get command status command
!    EOP_GUS = OP_GUS + OP_END,                    !Get unit status command
!    EOP_ONL = OP_ONL + OP_END,                    !Online command
!    EOP_RD = OP_RD + OP_END,                      !Read command
!    EOP_RPL = OP_RPL + OP_END,                    !Replace command
    EOP_SCC = OP_SCC + OP_END,                    !Set controller characteristics command
!    EOP_SUC = OP_SUC + OP_END,                    !Set unit characteristics command
!    EOP_WR = OP_WR + OP_END,                      !Write command
!    EOP_MRD = OP_MRD + OP_END,                    !Maintenance read command
!    EOP_MWR = OP_MWR + OP_END,                    !Maintenance write command
    EOP_SEX = OP_SEX + OP_END,                    !Serious exception

!+
! Dup Command message envelope
! byte sizes beginning at text+0
! of the command envelope.
!-
    SZ_GDS = 12,                                  !Get dust status size
    SZ_ESP = 40,                                  !Execute supplied program size
    SZ_ELP = 48,                                  !Execute local program size
    SZ_RED = 28,                                  !Receive data size
    SZ_SED = 28,                                  !Send data size
    SZ_ABT = 12,                                  !Abort program size
!+
! MSCP Command message envelope
! byte sizes beginning at text+0
! of the command envelope.
!-
    SZ_SCC = 32,                                  !Set Controller characteristics
    SZ_ONL = 36,                                  !On-line command
```

```
!+
! The following are the expected number of bytes
! in a commands  end  packet  transmitted by the
! communications mechanism to the host.
!-
    !
    ! DUP command end message sizes
    !
    ESZ_GDS = %DECIMAL '22',                    !Get dust status end packet size
    ESZ_ESP = %DECIMAL '12',                    !Execute supplied prog end packet size
    ESZ_ELP = %DECIMAL '16',                    !Execute local prog end packet size
    ESZ_RED = %DECIMAL '48',                    !Receive data end packet size
    ESZ_SED = %DECIMAL '48',                    !Send data end packet size
    ESZ_ABT = %DECIMAL '12',                    !Abort program end packet size
    !
    ! MSCP command end message sizes
    !
    ESZ_SCC = %DECIMAL '28',                    !Set controller characteristics
    ESZ_ONL = %DECIMAL '44',                    !On line command
    !
    ! DUP STANDALONE FLAG MODIFIER
    !
    DUP_STND = 1,
    !
    ! JDT Trap Vector
    !
    O_TVEC = %O'14';
```

```
*sbttl  FIELD DECLARATIONS'
field
    !+
    ! Definitions:
    !
    !    ISD_FIELD = Initialization Sequence Data_Field
    !
    !    ISRD_ = Initialization Sequence Read = 0
    !
    !    ISWRT_ = Initialization Sequence Write = 1
    !
    !    S1R_ = Step One Read
    !
    !    S1W_ = Step One Write
    !
    !    etc.
    !-
```

```
ISD FIELD -
    set
    !
    ! Miscellanious status register field
    ! referance declarations.
    !
    RC_ALL = [0, 16, 0],                    !RDRX word access
    ERR_BIT = [15, 1, 0],                   !Error bit
    ISR_ALL = [0, 16, 0],                   !Initialize sequence read word
    ISW_ALL = [0, 16, 0],                   !Initialize sequence write word
    STP_FIELD = [11, 4, 0],                 !All step bit fields
    SA_GO = [0, 1, 0],                      !Status register GO bit
    ERR_CODE = [0, 11, 0],                  !SA register fatal error code
    !
    ! Step one read SA register field reference
    !
    S1R_STEP = [11, 1, 0],                  !Step one step bit
    S1R_NV = [10, 1, 0],                    !No host inter vec settable adrs
    S1R_QB = [9, 1, 0],                     !22-bit addressing support
    S1R_DI = [8, 1, 0],                     !Enhanced diag implementation
    S1R_RSVD = [0, 8, 0],                   !Reserved field
    !
    ! Step one write SA register field reference
    !
    S1W_WR = [14, 1, 0],                    !Diag wrap around
    S1W_CRING = [11, 3, 0],                 !Number of C-ring slots 'pwrs of 2'
    S1W_RRING = [8, 3, 0],                  !Number of R-ring slots 'pwrs of 2'
    S1W_IE = [7, 1, 0],                     !Init Sequence interrupt request
    S1W_VADR = [0, 7, 0],                   !Interrupt vector address
    !
    ! Step two read SA register field reference
    !
    S2R_STEP = [12, 1, 0],                  !Step two step bit
    S2R_PTYP = [8, 3, 0],                   !Port type number
    S2R_BIT7 = [7, 1, 0],                   !Echoed IE bit from step one write
    S2R_WR = [6, 1, 0],                     !Echoed bit 14 from step one write
    S2R_CRING = [3, 3, 0],                  !Echoed bits 3-5 from step one write
    S2R_RRING = [0, 3, 0],                  !Echoed bits 0-2 from step one write
    !
    ! Step two write SA register field reference
    !
    S2W_LRBASE = [0, 16, 0],                !Ring base lower address
    S2W_PI = [0, 1, 0],                     !Adapter purge interrupt request
    !
    ! Step three read SA register field reference
    !
    S3R_STEP = [13, 1, 0],                  !Step three step bit
    S3R_RSVD = [8, 3, 0],                   !Reserved
    S3R_IE = [7, 1, 0],                     !Echoed IE bit from step one write
    S3R_VADR = [0, 7, 0],                   !Echoed VADR from step one write
    !
    ! Step three write SA register field reference
    !
    S3W_PP = [15, 1, 0],                    !Purge & Poll test request
    S3W_HRBASE = [0, 15, 0],                !Ring base high address
```

```
!
! Step four read SA register field reference
!
S4R_STEP = [14, 1, 0],                  !Step four step bit
S4R_RSVD = [8, 3, 0],                   !Reserved
S4R_MOD = [4, 4, 0],                    !Controller u-CODE version
S4R_VER = [0, 4, 0],                    !Controller u-CODE version
!
! Step four write SA register field reference
!
S4W_RSVD = [8, 8, 0],                   !Reserved
S4W_BURST = [2, 6, 0],                  !Max number longwords per NPR xfer
S4W_LF = [1, 1, 0],                     !Last fail request
S4W_GO = [0, 1, 0]                      !Go bit
tes,
```

```
!+
! Field declaration to define the interrupt
! indicator and purge words of the communications
! area.
!-

HDR_FIELD =
    set
    !
    ! Header Word Ringbase -4
    !
!   RESERVED = [0, 0, 16, 0],
    !
    ! Header Word Ringbase -3
    !
!   RSVD = [1, 0, 8, 0],
    ADP_CH = [1, 8, 8, 0],
    !
    ! Header Word Ringbase -2
    !
    CMD_INT = [2, 0, 16, 0],
    !
    ! Header Word Ringbase -1
    !
    RSP_INT = [3, 0, 16, 0]
    tes,
```

```
!+
! Field declaration to define the fields within
! the send and receive ring descriptors.
!-

DSC_FIELD =
    set
    !
    ! Low order envelope address
    !
    LO_EN$AD = [0, 0, 16, 0],
    !
    ! High order 18 bit unibus or Qbus address
    !
    HI_EN$AD = [1, 0, 2, 0],
    !
    ! Q_Bus extention address
    !
    QB_EXT = [1, 2, 4, 0],
    !
    ! Reserver field
    !
    D$RSVD = [1, 6, 8, 0],
    !
    ! Flag bit
    !
    FLAG_BIT = [1, 14, 1, 0],
    !
    ! Ownership bit
    !
    OWN_BIT = [1, 15, 1, 0]
    tes,
```

```
!+
! Field declaration to define the fields within
! message envelope buffers.
!-

ENV_FIELD =
    set
    !
    ! UQ Port envelope header field declaration
    !
    MSG_LENGTH = [0, 0, 16, 0],                 !Message length
    CREDITS = [1, 0, 4, 0],                     !Credits
    MSG_TYPE = [1, 4, 4, 0],                    !Message type
    CONN_ID = [1, 8, 8, 0],                     !Connection ID
```

```
!
! DUP/MSCP command and response envelope header
! field declaration
!
CMD_LREF = [2, 0, 16, 0],                   !Command Ref number low word
CMD_HREF = [3, 0, 16, 0],                   !Command Ref number high word
UNIT_NUM = [4, 0, 16, 0],                   !Unit selection field
UN_LUSED = [4, 0, 16, 0],                   !Unused low word
UN_HUSED = [5, 0, 16, 0],                   !Unused high word
TYP$MSG = [6, 6, 2, 0],                     !End code message type
OPCODE = [6, 0, 8, 0],                      !Opcode
ENDCODE = [6, 0, 8, 0],                     !Opcode
E$FLAG = [6, 8, 8, 0],                      !End message flag field
RSVD = [6, 8, 8, 0],                        !Reserved
STATUS = [7, 0, 16, 0],                     !Status
MODIFIER = [7, 0, 16, 0],                   !Modifier

!++
! DUP command and response envelope parameter
! field declarations
!--

!+
! ABORT command and response envelope parameter
! field declartion
!-
!
! No parameters declared
!
!       response FIELD
! No parameters declared

!+
! GET DUST STATUS command and response envelope
! parameter field declaration
!-
!       COMMAND FIELD
! No parameters declared
!
!       response FIELD
!
PLO_EXT = [8, 0, 16, 0],                    !Program Extension low word
PHI_EXT = [9, 0, 8, 0],                     !Program Extension high word
F$LAGS = [9, 8, 8, 0],                      !Flags
FLG_B0 = [9, 8, 1, 0],                      !Flag field bit 0
FLG_B1 = [9, 9, 1, 0],                      !Flag field bit 1
FLG_B2 = [9, 10, 1, 0],                     !Flag field bit 2
FLG_B3 = [9, 11, 1, 0],                     !Flag field bit 3
PLO_IND = [10, 0, 16, 0],                   !Progress indicator low word
PHI_IND = [11, 0, 16, 0],                   !Progress indicator high word
TIM_OUT = [12, 0, 16, 0],                   !Time out
```

```
!.
! EXECUTE SUPPLIED PROGRAM command and response
! envelope parameter field declartion
!-
!
!       COMMAND FIELD
!
BLO_CNT = [8, 0, 16, 0],                !Byte count low word
BHI_CNT = [9, 0, 16, 0],                !Byte count high word
BPA_LO = [10, 0, 16, 0],                !Buffer physical adrs bits <0-15>
BPA_HI = [11, 0, 2, 0],                 !Buffer physical adrs bits <16-17>
QBUS_EXT = [11, 2, 4, 0],               !Q_bus extention
RSV = [11, 6, 2, 0],                    !Reserved field
UBA_CHAN = [11, 8, 8, 0],               !Unibus adaptor channel number
RSV0 = [ 12, 0, 16, 0],                 !These next four words are not
RSV1 = [ 13, 0, 16, 0],                 !in the UQ port implementation.
RSV2 = [ 14, 0, 16, 0],                 !
RSV3 = [ 15, 0, 16, 0],                 !
!
! These next field definitions are the same
! as above except they are for the overlay
! buffer descriptors.  To make life easy for
! me I'll use the same names and just prefix
! the names with a $ for uniqueness.
!
$BPA_LO = [16, 0, 16, 0],               !Buffer physical adrs bits <0-15>
$BPA_HI = [17, 0, 2, 0],                !Buffer physical adrs bits <16-17>
$QBUS_EXT = [17, 2, 4, 0],              !Q_bus extention
$RSV = [17, 6, 2, 0],                   !Reserved field
$UBA_CHAN = [17, 8, 8, 0],              !Unibus adaptor channel number
$RSV0 = [ 18, 0, 16, 0],                !These next four words are not
$RSV1 = [ 19, 0, 16, 0],                !in the UQ port implementation.
$RSV2 = [ 20, 0, 16, 0],                !
$RSV3 = [ 21, 0, 16, 0],                !
!
!       response FIELD
! No parameters declared

!.
! EXECUTE LOCAL PROGRAM command and response
! parameter field declartion
!-
!
!       COMMAND FIELD
!
PN_0 = [8, 0, 16, 0],                   !Program name word 0
PN_1 = [9, 0, 16, 0],                   !Program name word 1
PN_2 = [10, 0, 16, 0],                  !Program name word 2
!
!       response FIELD
!
version = [8, 0, 16, 0],                !Version
TIME_OUT = [9, 0, 8, 0],                !Time out
FLAGS = [9, 8, 8, 0],                   !Flags
```

```
!+
! SEND DATA/RECEIVE DATA command and response
! parameter field declartion
!-
!       COMMAND FIELD
! byte count, buffer descriptor are the same
! as Execute Supplied Program parameters
!
!       response FIELD
! byte count is the same
! as Execute Supplied Program parameters


!++
! MSCP command and response envelope parameter
! field declarations
!--

!+
! SET CONTROLLER CHARACTERISTICS command and response
! parameter field declartion
!-
!
!       COMMAND FIELD
!
MSCP_VER = [ 8, 0, 16, 0],                      !MSCP version
CTL_FLAGS = [ 9, 0, 16, 0],                     !Controller flags
HOST_TOV = [ 10, 0, 16, 0],                     !Host time out value
RS$VD = [ 11, 0, 16, 0],                            !Reserved
T$D_0 = [ 12, 0, 16, 0],                            !Time and Date word 0
T$D_1 = [ 13, 0, 16, 0],                            !Time and Date word 1
T$D_2 = [ 14, 0, 16, 0],                            !Time and Date word 2
T$D_3 = [ 15, 0, 16, 0],                            !Time and Date word 3
CDP_LO = [ 16, 0, 16, 0],                       !Cntlr dep parameter lo word
CDP_HI = [ 17, 0, 16, 0],                       !Cntlr dep parameter hi wrd
!
!       RESPONSE FIELD
!
!MSCP_VER = [ 8, 0, 16, 0],                     !Same as cmd field
!CTL_FLAGS = [ 9, 0, 16, 0],                    !Same as cmd field
CTL_TOV = [ 10, 0, 16, 0],                      !Cntlr time out value
CSVRSN = [ 11, 0, 8, 0],                            !Cntlr S/W, F/W, u-code rev num
CHVRSN = [ 11, 8, 8, 0],                            !Cntlr H/W rev num
CID_0 = [ 12, 0, 16, 0],                            !Cntlr identifier wrd 0
CID_1 = [ 13, 0, 16, 0],                            !Cntlr identifier wrd 1
CID_2 = [ 14, 0, 16, 0],                            !Cntlr identifier wrd 2
CID_3 = [ 15, 0, 16, 0],                            !Cntlr identifier wrd 3
```

```
!.
! ONLINE COMMAND command and response
! parameter field declartion
!.
!
!       COMMAND FIELD
!
RSV$D= [ 8, 0, 16, 0],                  !Reserved
UNT_FLAGS = [ 9, 0, 16, 0],             !Unit flag field
RSVD$0 = [ 10, 0, 16, 0],               !Reserved field
RSVD$1 = [ 11, 0, 16, 0],               !Reserved field
RSVD$2 = [ 12, 0, 16, 0],               !Reserved field
RSVD$3 = [ 13, 0, 16, 0],               !Reserved field
RSVD$4 = [ 14, 0, 16, 0],               !Reserved field
RSVD$5 = [ 15, 0, 16, 0],               !Reserved field
DDP_LO = [ 16, 0, 16, 0],               !Device dependent parameter
DDP_HI = [ 17, 0, 16, 0],               !Device dependent parameter
SHADOW_UNIT = [ 18, 0, 16, 0],          !Shadow unit
COPY_SPEED  = [ 19, 0, 16, 0],          !Copy speed
!
!       RESPONSE FIELD
!
MUNT_CODE = [ 8, 0, 16, 0],             !Multi-unit code
!       UNT_FLAGS = [ 9, 0, 16, 0],     !Same as cmd field
!       RSVD$0 = [ 10, 0, 16, 0],       !Same as cmd field
!       RSVD$1 = [ 11, 0, 16, 0],       !Same as cmd field
UID_0 = [ 12, 0, 16, 0],                !Unit ident word 0
UID_1 = [ 13, 0, 16, 0],                !Unit ident word 1
UID_2 = [ 14, 0, 16, 0],                !Unit ident word 2
UID_3 = [ 15, 0, 16, 0],                !Unit ident word 3
MTID_LO = [ 16, 0, 16, 0],              !Media type ident word 0
MTID_HI = [ 17, 0, 16, 0],              !Media type ident word
!       SHADOW_UNIT = [ 18, 0, 16, 0],  !Same as cmd
SHA_STATE = [ 19, 0, 16, 0],            !Shadow state
USZ_LO = [ 20, 0, 16, 0],               !Unit size lo word
USZ_HI = [ 21, 0, 16, 0],               !Unit size hi word
VSN_LO = [ 22, 0, 16, 0],               !Volume serial num lo word
VSN_HI = [ 23, 0, 16, 0]                !Volume serial num hi word
tes.
```

```
!+
! Send Receive command buffer field definition
!-
RECB_FIELD =
    set
    MSG_NUM = [0, 0, 12, 0],
    MSG_TYP = [0, 12, 4, 0],
    MSG_TXT = [1, 0, 16, 0]
    tes,

!+
! Outstanding command buffer field declarations
!-
OUT$FIELD =
    set
    CMD_WRD = [0, 0, 16, 0],                !Command word ref "word 0 of slot"
    REC_FLG = [0, 15, 1, 0],                !Command received indicator flag
    CMD_REF = [0, 0, 8, 0],                 !Command reference field
    ENV_ADR = [1, 0, 16, 0]                 !Envelope adrs field
    tes;
```

```
%sbttl 'LINKAGE DELCARATIONS'

linkage
    !
    ! Call_lnk$typ
    !
    ! This  spcifies that  the PDP-11  JSR and RTS instructions
    ! are used by  the compiled  code,  and that the parameters
    ! with   standard  parameters  locations are   passed using
    ! register 5 (R5) as the argument pointer with the register
    ! usage as follows:
    !
    ! Register          Usage
    ! --------          -----
    !    0              Value return register, non-preserved
    !    1-4            Preserved
    !    5              Argument pointer
    !    6              Stack pointer
    !    7              Program counter
    !
    !
   CALL_LNK$TYP = call (standard),
    !
    !
    ! Int_lnk$typ
    !
    ! Specifies that a routine will be called only by a PDP-11
    ! hardware or  software interrupt  and will be returned via
    ! the RTI instruction. Register usage is as follows:
    !
    ! Register          Default usage
    ! --------          -------------
    !   0-5             Preserved
    !   6               Stack pointer
    !   7               Program pointer
    !
    !
   INT_LNK$TYP = interrupt (standard);
```

```
literal
!+
! CODES FOR SUPERVISOR SERVICE CALLS
!-
    C$RESERV = 0,                   !RESERVED FOR THE SUPERVISOR.
    C$ETST   = 54,                  !BLISS ENDTST--SIGNIFY END OF TEST.
    C$BSUB   = 2,                   !BLISS BGNSUB--SIGNIFY START OF SUBTEST.
    C$ESUB   = 55,                  !BLISS ENDSUB--SIGNIFY END OF SUBTEST.
    C$BSEG   = 4,                   !BGNSEG--SIGNIFY START OF SEGMENT.
    C$ESEG   = 56,                  !BLISS ENDSEG--SIGNIFY END OF SEGMENT.
    C$CLP1   = 53,                  !BLISS CKLOOP--CHECK IF LOOP ON ERROR,  WITHOUT A LABEL.
    C$RDBU   = 7,                   !READBUS--READ TYPE OF BUS
    C$ESCAPE = 52,                  !BLISS ESCAPE--ESCAPE INNER BLOCK
    C$INIT   = 9,                   !ENDINIT--SIGNIFY END OF INITIALIZE CODE.
    C$CLEAN  = 10,                  !ENDCLN--SIGNIFY END OF CLEAN-UP CODE.
    C$TPRI   = 11,                  !TRAPPRI--SET TRAP PRIORITY
    C$PNTB   = 12,                  !PRINTB--PRINT BASIC EXPANSION OF ERROR INFO.
    C$PNTX   = 13,                  !PRINTX--PRINT NON-BASIC EXPANSION OF ERROR INFO.
    C$PNTS   = 14,                  !PRINTS--PRINT A STATISTICAL REPORT.
    C$PNTF   = 15,                  !PRINTF--PRINT A FORCED MESSAGE.
    C$INLP   = 16,                  !INLOOP--TEST IF PROGRAM IS IN AN ERROR LOOP.
    C$RFLA   = 17,                  !RFLAGS--RETRIEVE FLAG SETTINGS OF THE DRS
    C$BRK    = 18,                  !BREAK--BREAK TO THE SUPERVISOR.
    C$MSG    = 19,                  !ENDMSG--END A BLOCK OF ERROR REPORTING CODE.
    C$DRPT   = 20,                  !DORPT--CALL STATISTICAL REPORT CODE.
    C$RPT    = 21,                  !ENDRPT--SIGNIFY END OF STATISTICAL REPORT.
    C$GETB   = 22,                  !GETBYTE--GET A BYTE FROM AN XXDP+ DATA FILE
    C$GETW   = 23,                  !GETWORD--GET A WORD FROM AN XXDP+ DATA FILE
    C$GPLO   = 24,                  !GPLOAD--GET PARAMETERS OF THE LOAD DEVICE
    C$MEM    = 25,                  !MEMORY--GET POINTER TO FREE MEMORY.
    C$EXIT   = 51,                  !BLISS EXIT--UNCONDITIONALLY EXIT TEST, SUBTEST, OR SEGMENT.
    C$RESET  = 27,                  !BRESET--BUS RESET.
    C$OPEN   = 28,                  !OPEN AN XXDP+ DATA FILE FOR READ
    C$CLOS   = 29,                  !CLOSE THE OPEN XXDP+ DATA FILE
    C$CVEC   = 30,                  !CLRVEC--RESTORE AN INTERRUPT VECTOR.
    C$SVEC   = 31,                  !SETVEC--SETUP AN INTERRUPT VECTOR.
    C$GPRI   = 32,                  !GETPRI--GET PRIORITY LEVEL.
    C$SPRI   = 33,                  !SETPRI--SET PRIORITY LEVEL.
    C$GPHRD  = 34,                  !GPHARD--GET H.W. P-TABLE ADR. SPECIFIED
    C$GMAN   = 35,                  !GMANI--GET MANUAL INTERVENTION.
    C$DCLN   = 36,                  !DOCLN--CALL CLEANUP CODE.
    C$CEFG   = 37,                  !CLREF--CLEAR EVENT FLAG.
    C$SEFG   = 38,                  !SETEF--SET EVENT FLAG.
    C$REFG   = 39,                  !READEF--READ EVENT FLAG.
    C$MANI   = 40,                  !MANUAL--DETERMINE IF MANUAL INTERVENTION USED
    C$DODU   = 41,                  !DODO--DROP UNITS
    C$AU     = 42,                  !ENDAU--SIGNIFY END OF ADD UNITS CODE
    C$DU     = 43,                  !ENDDU--SIGNIFY END OF DROP UNITS CODE
    C$ERSF   = 44,                  !ERRSF--SYSTEM FATAL ERROR.
    C$ERDF   = 45,                  !ERRDF--DEVICE FATAL ERROR.
    C$ERHRD  = 46,                  !ERRHRD--MEDIA HARD ERROR.
    C$ERSOFT = 47,                  !ERRSOFT--MEDIA SOFT ERROR.
    C$ERROR  = 48,                  !ERROR--DIAG. MODIFIABLE ERROR CALL
    C$AUTO   = 49,                  !ENDAUTO--SIGNIFY END OF AUTO CODE
    C$CLCK   = 50,                  !CLOCK--GET ADDRESS OF CLOCK TABLE
    C$QIO    = 255;                 !QIO CALLS
```

```
literal
!
! GET PATAMETER CODES
!-
    ! REQUEST TYPES (BITS 0, 1, 2)
    G$PRML = 0,                         !LOGICAL
    G$PRMA = 1,                         !ADDRESS
    G$PRMD = 2,                         !DATA
    G$DISP = 3,                         !DISPLAY
    G$XFER = 4,                         !TRANSFER

    ! DEFAULT (BIT 3)
    G$NO  = 0 * 8,                      !NO
    G$YES = 1 * 8,                      !YES

    ! RADIX (BITS 4, 5, 6)
    G$RADB = 0 * 16,                    !BINARY
    G$RADO = 1 * 16,                    !OCTAL
    G$RADD = 2 * 16,                    !DECIMAL
    G$RADL = 5 * 16,                    !LOGICAL
    G$RADA = 6 * 16,                    !ASCIZ

    ! OFFSET (BITS 9-15)
    G$OFSIZE = 254,                     !OFFSET MAXIMUM SIZE
    G$OFFSET = 1 * 256,                 !MULTIPLIER FOR GPRM'S AND GMANI
    X$OFFSET = 1 * 256,                 !MULTIPLIER FOR XFER'S.

    ! COUNT (BIT 7)
    G$CNTOP = 1 * 128,                  !COUNT OPTION INDICATOR

    ! EXCEPTION (BIT 8)
    G$EXCP  = 1 * 256,                  !EXCEPTION FLAG
    G$LOLIM = 1,                        !LOW LIMIT EXCEPTION
    G$HILIM = 2,                        !HIGH LIMIT EXCEPTION

    ! TRANSFER CONDITIONS
    X$ALWAYS = 0 * 16,                  !XFER
    X$TRUE   = 1 * 16,                  !XFERT
    X$FALSE  = 2 * 16;                  !XFERF

macro
    ERROR =
    L$ERROR (C$ERROR)
%;

linkage
    L$ERROR = trap;
```

```
macro
    ERRDF (ERRNUM, MSGPTR, ROUT) =
    begin
        builtin DECX;
        %if %length gtr 3
            %then %errormacro ('TOO MANY ARGUMENTS SPECIFIED')
        %fi;
        DECX (C$ERDF, ERRNUM, MSGPTR, ROUT);
    end
%;

macro
    ERRHRD (ERRNUM, MSGPTR, ROUT) =
    begin
        builtin DECX;
        %if %length gtr 3
            %then %errormacro ('TOO MANY ARGUMENTS SPECIFIED')
        %fi;
        DECX (C$ERHRD, ERRNUM, MSGPTR, ROUT);
    end
%;

macro
    ERRSF (ERRNUM, MSGPTR, ROUT) =
    begin
        builtin DECX;
        %if %length gtr 3
            %then %errormacro ('TOO MANY ARGUMENTS SPECIFIED')
        %fi;
        DECX (C$ERSF, ERRNUM, MSGPTR, ROUT);
    end
%;

macro
    ERRSOFT (ERRNUM, MSGPTR, ROUT) =
    begin
        builtin DECX;
        %if %length gtr 3
            %then %errormacro ('TOO MANY ARGUMENTS SPECIFIED')
        %fi;
        DECX (C$ERSOFT, ERRNUM, MSGPTR, ROUT);
    end
%;
```

```
macro
    FEQUAL =
    begin
    global literal
!+
! FUNCTION-LEVEL I/O DEFINITIONS
!-
    Q.IOFN = 2,                                 !DIRECTIVE FUNCTION
    Q.IOLN = 4,                                 !DIRECTIVE LOG UNIT NO.
    Q.IOEF = 6,                                 !DIRECTIVE EVENT FLAG
    Q.IOSB = 10,                                !DIRECTIVE STATUS BLOCK
    Q.IOAE = 12,                                !DIRECTIVE AST ENTRY
    Q.IOPL = 14,                                !DIRECTIVE PARAMETER LIST
    IS.SUC = 1;                                 !REQUEST WAS SUCCESSFUL
    end
%;

macro
    EQUALS =
    global literal
!+
! BIT DIFINITIONS
!-
    BIT15 = %o'100000',
    BIT14 = %o'40000',
    BIT13 = %o'20000',
    BIT12 = %o'10000',
    BIT11 = %o'4000',
    BIT10 = %o'2000',
    BIT09 = %o'1000',
    BIT08 = %o'400',
    BIT07 = %o'200',
    BIT06 = %o'100',
    BIT05 = %o'40',
    BIT04 = %o'20',
    BIT03 = %o'10',
    BIT02 = %o'4',
    BIT01 = %o'2',
    BIT00 = %o'1',
!
    BIT9 = BIT09,
    BIT8 = BIT08,
    BIT7 = BIT07,
    BIT6 = BIT06,
    BIT5 = BIT05,
    BIT4 = BIT04,
    BIT3 = BIT03,
    BIT2 = BIT02,
    BIT1 = BIT01,
    BIT0 = BIT00,
```

```
!+
! EVENT FLAG DEFINITIONS
!   EF32:EF17 RESERVED FOR SUPERVISOR TO PROGRAM COMMUNICATION
!-
    EF_START    = 32,                       !START COMMAND WAS ISSUED
    EF_RESTART  = 31,                       !RESTART COMMAND WAS ISSUED
    EF_CONTINUE = 30,                       !CONTINUE COMMAND WAS ISSUED
    EF_NEW      = 29,                       !A NEW PASS HAS BEEN STARTED
    EF_PWR      = 28,                       !A POWER-FAIL/POWER-UP OCCURRED
!+
! PRIORITY LEVEL DEFINITIONS
!-
    PRI07 = %o'340',
    PRI06 = %o'300',
    PRI05 = %o'240',
    PRI04 = %o'200',
    PRI03 = %o'140',
    PRI02 = %o'100',
    PRI01 = %o'40',
    PRI00 = %o'0',
!+
!OPERATOR FLAG BITS
!-
    EVL = %o'4',
    LOT = %o'10',
    ADR = %o'20',
    IDU = %o'40',
    ISR = %o'100',
    UAM = %o'200',
    BOE = %o'400',
    PNT = %o'1000',
    PRI = %o'2000',
    IXE = %o'4000',
    IBE = %o'10000',
    IER = %o'20000',
    LOE = %o'40000',
    HOE = %o'100000';
%;

macro
    BREAK =
    T$RAP (C$BRK)
%;

macro
    BRESET =
    T$RAP (C$RESET)
%;
```

```
macro
    DELAY (MULT) =
    begin
        external L$CPU, L$DLY;
        local $$TMP2;
        $$TMP2 = .L$CPU * MULT;
        while .$$TMP2 neq 0 do
        begin
            begin
            decru $$TMP1 from .L$DLY to 1 do
                begin
                local $$TMP : volatile;
                $$TMP = 0;
                end;
            end;
        $$TMP2 = .$$TMP2 - 1;
        end;
    end;
%;

macro
    RFLAGS (LOC) =
    LOC = L$RFLAGS (C$RFLA)
%;

linkage
    L$RFLAGS = trap (register = 0);

macro
    READBUS =
    (L$READBUS (C$RDBU))
%;

linkage
    L$READBUS = trap : VALUECBIT clearstack;

macro
    MEMORY (MEMLOC) =
    MEMLOC = L$MEMORY (C$MEM)
%;

linkage
    L$MEMORY = trap (register = 0);

macro
    INLOOP =
    (L$INLOOP (C$INLP))
%;

linkage
    L$INLOOP = trap : VALUECBIT clearstack;
```

```
macro
    MANUAL =
    (L$MANUAL (C$MANI))
%;

linkage
    L$MANUAL = trap : VALUECBIT clearstack;

macro
    CKLOOP =
    if (T$RAPRO (C$CLP1)) then
        leave  T$$TAG ;
%;

macro
    ESCAPE =
    if (T$RAPRO (C$ESCAPE)) then
        leave T$$TAG ;
%;

macro
    EXIT =
    begin
        T$RAPRO (C$EXIT);
        leave T$$TAG ;
    end;
%;

macro
    EXIT_TST =
    begin
        T$RAPRO (C$EXIT);
        return;
    end;
%;

compiletime
    $BGNTST   = 0,                  !THIS FLAG IS SET AT BGNTST AND CLEARED AT ENDTST
    $BGNSUB   = 0,                  !THIS FLAG IS SET AT BGNSUB AND CLEARED AT ENDSUB
    $BGNSEG   = 0,                  !THIS FLAG IS INCREMENTED AT BGNSEG AND DECREMENTED AT ENDSEG
    $TSTNUM   = 1,                  !THIS VARIABLE IS USED TO GENERATE THE TEST LABEL
    $SUBNUM   = 1,                  !THIS VARIABLE IS USED TO GENERATE THE SUBROUTINE LABEL
    $SEGNUM   = 1,                  !THIS VARIABLE IS USED TO GENERATE THE SEGMENT LABEL
    $PUSHLEV  = 0,                  !THIS VARIABLE IS USED WITHIN SEGMENT NESTINGS
    $POPLEV   = 8,                  !THIS VVARIABLE IS USED WITHIN SEGMENT NESTINGS
    T$TEMP    = 0,                  !THIS VARIABLE IS USED TO GENERATE GPRM'S/GMANI'S
    T$CODE    = 0,                  !THIS VARIABLE IS USED TO GENERATE GPRM'S/GMANI'S
    C$COUNT   = 0;                  !COUNTER USED IN VARIOUS MACROS
```

```
macro
    BGNINIT =
    routine LINIT : novalue =
        begin
%;

macro
    BGNAU =
    routine LAU : novalue =
        begin
%;

macro
    BGNAUTO =
    routine LAUTO : novalue =
        begin
%;

macro
    BGNCLN =
    routine LCLEAN : novalue =
        begin
%;

macro
    BGNDU =
    routine LDU : novalue =
        begin
%;

macro
    BGNRPT =
    routine LRPT : novalue =
        begin
%;
```

```
macro
    LASTAD =
    psect
        code = $XYZ$;

    global routine $END_LINK : novalue =
!
! THE FOLLOWING DATA STRUCTURE MUST BE IN
! A ROUTINE AND NOT AT 'MODULE' LEVEL.
! OTHERWISE THE PRESENT COMPILER COMPLAINS.
!
        begin
        forward
            T$FREE : vector [1]          psect ($XYZ$);
        global BL$LAS : vector [2]       psect ($XYZ$)
            initial (T$FREE, ((T$FREE - BL$LAS [2]) / 2));
        global bind
            L$LAST = BL$LAS [2];                    !THE ADDR OF THE WORD AFTER T$SIZE
%;
```

```
macro
    BGNSETUP (NUM) =
    global bind
        T$PTHV = NUM;

    compiletime
!
! P$PTR  = 0, THIS USED TO BE TRUE BUT CALCULATIONS FOR POINTERS WERE OFF
!
        P$PTR  = 4,
        PT$NUM = 0;
    %assign (PT$NUM, NUM)
    psect plit = $XYZ$;
%;

macro
    ENDSETUP =
    %if PT$NUM neq 0
        %then %errormacro ('ERROR IN PTABLE BLOCK')
    %fi
    global T$FREE : vector [1]          psect ($XYZ$) initial (0);
!
! THIS LOCATION MAY GET OVERWRITTEN WITH NO PROBLEM
!
    end;
%;

macro
    BGNPTAB =
    %assign (PT$NUM, PT$NUM + 1)
        PTABS (
%;

macro
    PTABS [] =
!
! %ASSIGN (P$PTR, P$PTR + (2 * (%LENGTH - 4))) THIS USED TO BE TRUE BUT CALCULATIONS FOR POINTERS WERE OFF
!
    %assign (P$PTR, P$PTR + (2 * (%length - 2)))
    %if PT$NUM eql 2
        %then
            bind $$LAS1 = uplit (0);
        %else
            bind %name ('$LAS', %number (PT$NUM)) = uplit (L$LAST + P$PTR);
    %fi
    bind
        %name ('$REM', %number (PT$NUM)) = plit (%remaining);
%;
```

```
macro
    ENDPTAB =
        )
    %assign (PT$NUM, PT$NUM - 2)
%;

macro
    T$$TAG =
    %name ('T$', %number ($TSTNUM), %number ($SUBNUM), %number ($SEGNUM))
%;

macro
    BGNTST =
    %if $BGNSUB neq 0
        %then
            %errormacro ('"BGNTST" IN SUB')
            %exitmacro
    %fi
    %if $BGNSEG neq 0
        %then
            %errormacro ('"BGNTST" IN SEG')
            %exitmacro
    %fi
    %if $BGNTST neq 0
        %then
            %errormacro ('"BGNTST" IN TST')
            %exitmacro
    %fi
    %assign ($PUSHLEV, 0)
    %assign ($POPLEV, 8)                        !DCR'D TO 0;<0=ERROR
    %assign ($SUBNUM, 0)
    %assign ($SEGNUM, 0)
    %assign ($BGNTST, 1)
    routine %name ('$T', %number ($TSTNUM)) ; novalue =
        begin
            label T$$TAG;
            T$$TAG:   begin
%;

macro
    BGNSEG =
    %assign ($PUSHLEV, $PUSHLEV + 1)
    %assign ($SEGNUM, $SEGNUM + 1)
    %assign ($BGNSEG, 1)
    %if $PUSHLEV gtr 8
        %then %errormacro ('TOO MANY "BGNSEG"S')
    %fi
    do  begin
        label  T$$TAG;
        T$$TAG: begin
                T$RAPRO (C$BSEG);
%;
```

```
macro
    BGNSUB =
    %if $BGNSEG neq 0
        %then
            %errormacro ( "BGNSUB" IN SEG' )
            %exitmacro
    %fi
    %if $BGNSUB neq 0
        %then
            %errormacro ('"BGNSUB" IN SUB' )
            %exitmacro
    %fi
    %assign ($BGNSUB, 1)
    %assign ($SUBNUM, $SUBNUM + 1)
    do  begin
        label  T$$TAG;
        T$$TAG: begin
                T$RAPRO (C$BSUB);
%;

macro
    BGNMSG (NAME) =
    forward routine %name ('M$', NAME) : novalue;
    global routine NAME : novalue =
        begin
            %name ('M$', NAME) ();
            L$MSG (C$MSG);
        end;
    routine %name ('M$', NAME) : novalue =
        begin
%;

linkage
    L$MSG = trap;

macro
    BGNSRV (ISR) =
    global routine ISR (OLDPSW, OLDPC) : L$ISR novalue =
        begin
%;

linkage
    L$ISR = interrupt;

macro
    ENDSRV =
        end;
%;
```

```
macro
    ENDSRV_MODPRI (PRIORITY) =
    %if PRIORITY gtr 7
        %then
            %errormacro ( 'PRIORITY MUST BE 0 TO 7' )
            %exitmacro
        %else
            OLDPSW <5,3> = PRIORITY;
    %fi
    ENDSRV
%;

macro
    ENDMSG =
        end;
%;

macro
    ENDINIT =
        end;
    global routine L$INIT : novalue =
        begin
            LINIT ();
            T$RAP (C$INIT);
        end;
%;

macro
    ENDAU =
        end;
    global routine L$AU : novalue =
        begin
            LAU ();
            T$RAP (C$AU);
        end;
%;

macro
    ENDAUTO =
        end;
    global routine L$AUTO : novalue =
        begin
            LAUTO ();
            T$RAP (C$AUTO);
        end;
%;

macro
    ENDCLN =
        end;
    global routine L$CLEAN : novalue =
        begin
            LCLEAN ();
            T$RAP (C$CLEAN);
        end;
%;
```

```
macro
    ENDDU =
        end;
    global routine L$DU : novalue =
        begin
            LDU ();
            T$RAP (C$DU);
        end;
%;

macro
    ENDRPT =
        end;
    global routine L$RPT : novalue =
        begin
            LRPT ();
            T$RAP (C$RPT);
        end;
%;

macro
    ENDTST =
        %if $BGNTST eql 0
            %then
                %errormacro ('MISSING "BGNTST"')
                %exitmacro
        %fi
        end
    end;
    global routine %name ('T', %number ($TSTNUM)) : novalue =
        do
            %name ('$T', %number ($TSTNUM)) ()
        while T$RAP (C$ETST);
        %assign ($TSTNUM, $TSTNUM + 1)
        %assign ($BGNTST, 0)
        %if (($PUSHLEV + $POPLEV) - 8) neq 0
            %then %errormacro ('TEST CONTAINS UNEQUAL "BGNSEG"S AND "ENDSEG"S')
        %fi
%;

macro
    ENDSUB =
        %if $BGNSUB eql 0
            %then
                %errormacro ('MISSING "BGNSUB"')
                %exitmacro
        %fi
        end
    end
    while  (T$RAPRO (C$ESUB));
    %assign ($BGNSUB, 0)
%;
```

```
macro
    ENDSEG =
        %assign (%POPLEV, %POPLEV - 1)
        %if %POPLEV lss 0
            %then %errormacro ('TOO MANY "ENDSEG"S')
        %fi
!
!THE TEST FOR PROPER NESTING IS IN MACRO "ENDTST"
!
        end
    end
    while  (T%RAPRO (C%ESEG));
    %assign (%BGNSEG, 0)
%;

linkage
    T%RAPRO = trap (register = 0);

macro
    SETVEC (VECADD, ROUT, PRIOR) =
    L%SETVEC (C%SVEC, PRIOR, ROUT, VECADD, 3)
%;

linkage
    L%SETVEC = trap (standard, standard, standard, standard);

macro
    SETPRI (PRIOR) =
    L%SETPRI (C%SPRI, PRIOR)
%;

linkage
    L%SETPRI = trap (register = 0);

macro
    CLRVEC (VECADD) =
    L%CLRVEC (C%CVEC, VECADD)
%;

linkage
    L%CLRVEC = trap (register = 0);

macro
    PRINTB [] =
    begin
        builtin sp;
        %name ('D%ERR', %length) (C%PNTB, REVERSEACTUALS (%remaining), %length, .sp);
    end
%;
```

```
macro
    PRINTX [] =
    begin
        builtin sp;
        %name ('D$ERR', %length) (C$PNTX, REVERSEACTUALS (%remaining), %length, .sp);
    end
%;

macro
    PRINTS [] =
    begin
        builtin sp;
        %name ('D$ERR', %length) (C$PNTS, REVERSEACTUALS (%remaining), %length, .sp);
    end
%;

macro
    PRINTF [] =
    begin
        builtin sp;
        %name ('D$ERR', %length) (C$PNTF, REVERSEACTUALS (%remaining), %length, .sp);
    end
%;

macro
!+
! MAKE A ARGUMENT LIST THAT IS THE REVERSE OF THE ONE PASSED IN.
!-
    REVERSEACTUALS (A) [] =
    %if %length eql 1
        %then A
        %else REVERSEACTUALS (%remaining), A
    %fi
%;

linkage
    D$ERR1 = trap (standard, standard, register = 0);

linkage
    D$ERR2 = trap (standard, standard, standard, register = 0);

linkage
    D$ERR3 = trap (standard, standard, standard, standard, register = 0);

linkage
    D$ERR4 = trap (standard, standard, standard, standard, standard, register = 0);

linkage
    D$ERR5 = trap (standard, standard, standard, standard, standard, standard, register = 0);

linkage
    D$ERR6 = trap (standard, standard, standard, standard, standard, standard, standard, register = 0);
```

```
linkage
    D$ERR7 = trap (standard, standard, standard, standard, standard, standard, standard, standard, register =
             0);

linkage
    D$ERR8 = trap (standard, standard, standard, standard, standard, standard, standard, standard, standard,
             register = 0);

linkage
    D$ERR9 = trap (standard, standard, standard, standard, standard, standard, standard, standard, standard,
             standard, register = 0);

macro
    GPHARD (UNIT, POINTER) =
    (POINTER = L$GPHARD (C$GPHRD, UNIT))
%;

linkage
    L$GPHARD = trap (register = 0);

macro
    CLOSE =
    T$RAP (C$CLOS)
%;

macro
    OPEN (FILENAME) =
    D$FILE (C$OPEN, FILENAME)
%;

macro
    GETWORD (DEST) =
    GET_DATA (C$GETW, DEST)
%;

macro
    GETBYTE (DEST) =
    GET_DATA (C$GETB, DEST)
%;

macro
    GET_DATA (code, DEST) =
    begin
        if D$GET (code,DEST)
            then (builtin r0; DEST = .r0; 1)    !CASE FOR CARRY SET(COMPLETE)
            else (0)                            !CARRY CLEAR = END OF FILE
    end
%;

linkage
    D$GET = trap (register = 0) : VALUECBIT clearstack;
```

```
macro
    GETPRI (PRIO) =
    PRIO = (D$FILE (C$GPRI))
%;

macro
    DORPT =
    L$DORPT (C$DRPT)
%;

linkage
    L$DORPT = trap;

macro
    DODU (UNIT) =
    L$DODU (C$DODU, UNIT)
%;

linkage
    L$DODU = trap (register = 0);

macro
    DOCLN =
    L$DOCLN (C$DCLN)
%;

linkage
    L$DOCLN = trap;

macro
    READEF (EFN) =
    L$READEF (C$REFG, EFN)
%;

linkage
    L$READEF = trap (register = 0) : VALUECBIT clearstack;

macro
    CLOCK (TYPE, POINTER) =
    %if %identical (TYPE, L)
        %then
            M$CLCK (POINTER, %o'114')
            %exitmacro
    %fi
    %if %identical (TYPE, P)
        %then
            M$CLCK (POINTER, %o'120')
            %exitmacro
        %else
            %errormacro ('TYPE MUST BE L OR P')
    %fi
%;
```

```
macro
    M$CLCK (POINTER, KIND) =
    begin
        if L$CLCK (C$CLCK, KIND)
            then (builtin r0; POINTER = .r0; 1) !1 IS THE CASE FOR CARRY SET.
                                                !OTHERWISE KNOWN AS "COMPLETE"
            else (0)                            !0 IS THE CASE FOR CARRY CLEAR.
                                                !OTHERWISE KNOWN AS "INCOMPLETE"
    end
%;

linkage
    L$CLCK = trap (register = 0) : VALUECBIT clearstack;

linkage
    D$FILE = trap (register = 0);

linkage
    T$RAP = trap;

macro
    GMANIL (MSGADR, DATADR, MASK, DFLT) =
    begin
        builtin DECX;
        %assign (T$CODE, G$PRML or G$RADL)
        M$DFLT (DFLT, T$TEMP);
        %assign (T$CODE, T$CODE or T$TEMP)
        DECX (C$GMAN, %o'404', DATADR, T$CODE, MSGADR, MASK);
    end
%;

macro
    GMANID (MSGADR, DATADR, RADIX, MASK, LOW, HIGH, DFLT) =
    begin
        builtin DECX;
        %assign (T$CODE, G$PRMD)
        M$RAD (RADIX, T$TEMP);
        %assign (T$CODE, T$CODE or T$TEMP)
        M$DFLT (DFLT, T$TEMP);
        %assign (T$CODE, T$CODE or T$TEMP)
        DECX (C$GMAN, %o'406', DATADR, T$CODE, MSGADR, MASK, LOW, HIGH);
    end
%;
```

```
macro
    GMANIA (MSGADR, DATADR, RADIX, LOW, HIGH, DFLT) =
    begin
        builtin DECX;
        %if %identical (A, RADIX)
            %then %errormacro ('INVALID RADIX')
        %fi
        %assign (T$CODE, G$PRMA)
        M$RAD (RADIX, T$TEMP);
        %assign (T$CODE, T$CODE or T$TEMP)
        M$DFLT (DFLT, T$TEMP);
        %assign (T$CODE, T$CODE or T$TEMP)
        DECX (C$GMAN, %o'405', DATADR, T$CODE, MSGADR, LOW, HIGH);
    end
%;

macro
    M$RAD (IN, OUT) =
    %if %identical (IN, B)
        %then
            %assign (OUT, G$RADB)
            %exitmacro
    %fi
    %if %identical (IN, O)
        %then
            %assign (OUT, G$RADO)
            %exitmacro
    %fi
    %if %identical (IN, D)
        %then
            %assign (OUT, G$RADD)
            %exitmacro
    %fi
    %if %identical (IN, L)
        %then
            %assign (OUT, G$RADL)
            %exitmacro
    %fi
    %if %identical (IN, A)
        %then
            %assign (OUT, G$RADA)
            %exitmacro
    %fi
    %errormacro ('ILLEGAL RADIX')
%;

macro
    M$DFLT (DFLT, OUT) =
    %if %identical (YES, DFLT)
        %then %assign (OUT, G$YES)
        %else
            %if %identical (NO, DFLT)
                %then %assign (OUT, G$NO)
                %else %errormacro ('INCORRECT DEFAULT')
            %fi
    %fi
%;
```

```
compiletime
    C$TSCT     = 0,
    O$BGNSFT   = 0,
    O$BGNRPT   = 0,
    O$GNSW     = 0,
    O$APTS     = 0,
    O$AU       = 0,
    O$DU       = 0,
    O$ERRTBL   = 0,
    O$SETUP    = 0,
    O$POINTER  = 0,
    C$ITCNT    = 0;

macro
    BGNPROT (A, B, C) =
    global L$PROT : vector [3]  psect ($code$)
          initial (A, B, C);
%;

macro
    ENDPROT =
%;

macro
    DISPATCH (DSNUM) =
    global D$PCNT : vector [1]          psect ($code$)
        initial (DSNUM);
    %assign (C$TSCT, 1)
    FORDEC (DSNUM)                                  !DECLARE ROUTINE NAMES 'FORWARD' OR 'EXTERNAL'
    %assign (C$TSCT, 1)
    global  L$DISPATCH : vector [DSNUM] psect ($code$)
        initial (
            LOADTS (DSNUM)
            );
%;

macro
    LOADTS (TSTNMS) [] =
    %assign (C$ITCNT, TSTNMS)
    %name ('T', %number (C$TSCT))
    %assign (C$ITCNT, C$ITCNT - 1)
    %assign (C$TSCT, C$TSCT + 1)
    %if %number (C$ITCNT) eql 0
        %then %exitmacro
        %else ,
            LOADTS (C$ITCNT)
    %fi
%;
```

```
macro
    FORDEC (TSTNMS) [] =
    %assign (C$ITCNT, TSTNMS)
    external routine %name ('T', %number (C$TSCT)) : novalue;
        %assign (C$ITCNT, C$ITCNT - 1)
        %assign (C$TSCT, C$TSCT + 1)
        %if %number (C$ITCNT) eql 0
            %then %exitmacro
            %else FORDEC (C$ITCNT)
        %fi
%;

macro
    POINTER (PNTR) [] =
    $POINT (PNTR);
    psect plit = $code$;
%;
```

```
macro
    $POINT (PNTR) [] .
    $if $identical (ALL, PNTR)
        $then
            $assign (O$BGNSFT, 1)
            $assign (O$BGNRPT, 1)
            $assign (O$GNSW, 1)
            $assign (O$APTS, 1)
            $assign (O$AU, 1)
            $assign (O$DU, 1)
            $assign (O$ERRTBL, 1)
            $assign (O$SETUP, 1)
            $assign (O$POINTER, 1)
            $exitmacro
    $fi
    $if $identical (NONE, PNTR)
        $then
            $assign (O$POINTER, 1)
            $exitmacro
    $fi
    $if $identical (SFT, PNTR)
        $then
            $assign (O$BGNSFT, 1)
            $assign (O$POINTER, 1)
    $fi
    $if $identical (RPT, PNTR)
        $then
            $assign (O$BGNRPT, 1)
            $assign (O$POINTER, 1)
    $fi
```

```
    %if %identical (SW, PNTR)
        %then
            %assign (O$GNSW, 1)
            %assign (O$POINTER, 1)
    %fi;
    %if %identical (APTSTAT, PNTR)
        %then
            %assign (O$APTS, 1)
            %assign (O$POINTER, 1)
    %fi;
    %if %identical (AU, PNTR)
        %then
            %assign (O$AU, 1)
            %assign (O$POINTER, 1)
    %fi;
    %if %identical (DU, PNTR)
        %then
            %assign (O$DU, 1)
            %assign (O$POINTER, 1)
    %fi;
    %if %identical (TBL, PNTR)
        %then
            %assign (O$ERRTBL, 1)
            %assign (O$POINTER, 1)
    %fi;
    %if %identical (SETUP, PNTR)
        %then
            %assign (O$SETUP, 1)
            %assign (O$POINTER, 1)
    %fi;
    $POINT (%remaining);
%;
```

```
macro
    BGNSW (NAM) =
    psect own = $code$;
    psect global = $code$;
    forward L$NDSW: vector [1];
    global
        L$SWLEN : vector [1]              psect ($code$)
            initial ((L$NDSW - L$SWLEN) / 2);
    %if %length eql 0
        %then %exitmacro
        %else global bind NAM = L$SW;
    %fi
%;

macro
    ENDSW =
    global L$NDSW : vector [1]            psect ($code$);
    psect global = $GLOB$;
    psect own = $own$;
%;

macro
    BGNHW (NAM) =
    psect global = $code$;
    psect own = $code$;
    forward L$NDHW : vector [1];
    global L$HWLEN : vector [1]           psect ($code$)
        initial ((L$NDHW - L$HWLEN) / 2);
    %if %length eql 0
        %then %exitmacro
        %else global bind NAM = L$HW;
    %fi
%;

macro
    ENDHW =
    global L$NDHW : vector [1]            psect ($code$);
    psect own = $own$;
    psect global = $GLOB$;
%;

macro
    ERRTBL =
    psect global = $code$;
    global ERRTYP : vector [1];
    global ERRNBR : vector [1];
    global ERRMSG : vector [1];
    global ERRBLK : vector [1];
    psect global = $GLOB$;
%;
```

```
macro
    HEADER (FILNAM, REVLEV, DEPO, TIME, TYPE, PRI) =
    %if O$POINTER eql 0
        %then
            %errormacro ( 'MISSING POINTER MACRO' )
            %exitmacro
    %fi;
    external                                        !TO RESOLVE REFERENCES IN OTHER SKELS
        L$SOFT, T$PTHV, L$RPT, L$INIT, L$CLEAN,
        L$LAST, L$HARD, L$DVTYP, L$DESC, L$DU, L$AU, L$AUTO;
    compiletime
        C$REVISION = 3,                             !
        C$EDIT = 3;                                 !
    forward
        ERRTYP : vector [1],
        L$SWLEN : vector [1],
        L$HWLEN : vector [1],
        L$PROT : vector [3],
        L$DISPATCH : vector [DS$NBR_OF_TESTS];
    global bind
        L$ERRTBL = ERRTYP,
        L$SW = L$SWLEN + 2,
        L$HW = L$HWLEN + 2;
    compiletime EMT_E$LOAD = %o'104035';
    %assign (C$COUNT, 8 - %charcount (FILNAM));
    global L$NAME : vector [4]              psect ($code$);
    %if %charcount (FILNAM) gtr 7
        %then
            %errormacro ('NAME TOO BIG')
            %exitmacro
    %fi
        initial (FILNAM, rep C$COUNT of byte (0));
    global L$REV : vector [1]               psect ($code$)
        initial (byte (REVLEV), byte (DEPO));
    global bind L$DEPO = L$REV + 1;
    global L$UNIT : vector [1]              psect ($code$)
        initial (%if O$SETUP eql 0
                    %then 0
                    %else T$PTHV
                %fi);
    global L$TIML : vector [1]              psect ($code$)
        initial (TIME);
    global L$HPCP : vector [1]              psect ($code$)
        initial (L$HARD);
    global L$SPCP : vector [1]              psect ($code$)
        initial (%if O$BGNSFT eql 0
                    %then 0
                    %else L$SOFT
                %fi);
    global L$HPTP : vector [1]              psect ($code$)
        initial (L$HW);
    global L$SPTP : vector [1]              psect ($code$)
        initial (%if O$GNSW eql 0
                    %then 0
                    %else L$SW
                %fi);
```

```
global L$LADP : vector [1]              psect ($code$)
    initial (L$LAST);
global L$STA : vector [1]               psect ($code$)
    initial (0);
global L$CO : vector [1]                psect ($code$)
    initial (0);
%if (1 - TYPE) neq 0
    %then %if TYPE neq 0
          %then
              %errormacro ('DIAG. TYPE MUST BE 0 OR 1')
              %exitmacro
          %fi
%fi
global L$DTYP : vector [1]              psect ($code$)
    initial (TYPE);
global L$APT : vector [1]               psect ($code$)
    initial (0);
global L$DTP : vector [1]               psect ($code$)
    initial (L$DISPATCH);
global L$PRIO : vector [1]              psect ($code$)
    initial (PRI);
global L$ENVI : vector [1]              psect ($code$)
    initial (0);
global L$EXP1 : vector [1]              psect ($code$)
    initial (0);
global L$MREV : vector [1]              psect ($code$)
    initial (byte (C$REVISION), byte (C$EDIT));
global L$EF : vector [2]                psect ($code$)
    initial (0, 0);
global L$SPC : vector [1]               psect ($code$)
    initial (0);
global L$DEVP : vector [1]              psect ($code$)
    initial (L$DVTYP);
global L$REPP : vector [1]              psect ($code$)
    initial (%if O$BGNRPT eql 0
                %then 0
                %else L$RPT
             %fi);
global L$EXP4 : vector [1]              psect ($code$)
    initial (0);
global L$EXP5 : vector [1]              psect ($code$)
    initial (0);
global L$AUT : vector [1]               psect ($code$)
    initial (%if O$AU eql 0
                %then 0
                %else L$AU
             %fi);
global L$DUT : vector [1]               psect ($code$)
    initial (%if O$DU eql 0
                %then 0
                %else L$DU
             %fi);
global L$LUN : vector [1]               psect ($code$)
    initial (0);
global L$DESP : vector [1]              psect ($code$)
    initial (L$DESC);
global L$LOAD : vector [1]              psect ($code$)
    initial (EMT_E$LOAD);
```

```
    global L$ETP : vector [1]           psect ($code$)
        initial (%if O$ERRTBL eql 0
                    %then 0
                    %else L$ERRTBL
                 %fi);
    global L$ICP : vector [1]           psect ($code$)
        initial (L$INIT);
    global L$CCP : vector [1]           psect ($code$)
        initial (L$CLEAN);
    global L$ACP : vector [1]           psect ($code$)
        initial (L$AUTO);
    global L$PRT : vector [1]           psect ($code$)
        initial (L$PROT);
    global L$TEST : vector [1]          psect ($code$)
        initial (0);
    global L$DLY : vector [1]           psect ($code$)
        initial (0);
    global L$HIME : vector [1]          psect ($code$)
        initial (0);
    global L$CPU : vector [1]           psect ($code$)
        initial (1);
%;

compiletime
    X$XFER_SKIP = %o'001004',                   !THIS IS A NO-OP, I.E., "XFER .PC + 2"
    T$HILIM     = 0,
    T$LOLIM     = 0,
    T$EXCP      = 0,
    C$ATLO      = 0,
    GP$_COUNT   = 0,
    C$ATHI      = 0;

macro
    DEVTYP (TYP) =
    global L$DVTYP : vector [(%charcount (TYP) / 2) + 1]
        psect ($code$)
        initial (TYP);
%;

macro
    DESCRIPT (DESC) =
    global L$DESC : vector [(%charcount (DESC) / 2) + 1]
        psect ($code$)
        initial (DESC);
%;

macro
    XFER (LAB) =
    M$XFER (LAB, X$ALWAYS);
%;
```

```
macro
    XFERF (LAB) =
    M$XFER (LAB, X$FALSE)
%;

macro
    XFERT (LAB) =
    M$XFER (LAB, X$TRUE)
%;

macro
    M$XFER (LAB, FLAVOR) =
    forward
        %name ('$L', LAB);
! GENERATE THE XFER :
    psect own = $code$;
    own %name ('$', LAB):
        initial (((%name ('$L', LAB) - %name ('$', LAB)) ^ X$OFFSET) +
                 G$XFER + FLAVOR);
    psect own = $own$;
%;

macro
    $L (LAB) =
! GEN A LABEL FOR OFFSET CALC AND FILL IT
! WITH A NO-OP
    psect own = $code$;
    own
        %name ('$L', LAB) : initial (X$XFER_SKIP);
%;

macro
    BGNHRD =
    forward L$NDHRD : vector [1];
    global L$HRDLN : vector [1]          psect ($code$)
        initial (((L$NDHRD - L$HRDLN) / 2) - 1);
    global bind L$HARD = L$HRDLN + 2;
%;

macro
    BGNSFT =
    forward L$NDSFT : vector [1];
    global L$SFTLN : vector [1]          psect ($code$)
        initial (((L$NDSFT - L$SFTLN) / 2) - 1);
    global bind L$SOFT = L$SFTLN + 2;
%;

macro
    ENDHRD =
    global L$NDHRD : vector [1]          psect ($code$);
%;
```

```
macro
    ENDSFT =
    global L$NDSFT : vector [1]          psect ($code$);
%;

macro
    GPRMA (MSGADR, DATADR, RADIX, LOW, HIGH, DFLT, COUNT) =
    %assign (GP$_COUNT, GP$_COUNT + 1)                    !GEN. UNIQUE NAMES
    %assign (T$EXCP, 0)
    %assign (T$CODE, 0)
    %assign (C$COUNT, 4)                                  !4 ARGS, COUNT IS 'OPTIONAL'
    NO_A_RADIX (RADIX);
    NO_ODD_ADR (DATADR);
    OFFSET_SIZE (DATADR);
    %assign  (T$CODE, G$PRMA + (DATADR ^ G$OFFSET))
    M$RAD (RADIX, T$TEMP);
    %assign (T$CODE, T$CODE or T$TEMP)
    M$DFLT (DFLT, T$TEMP);
    %assign (T$CODE, T$CODE or T$TEMP)
    M$EXCP (LOW,HIGH)                                     !LOAD LOW AND HIGH LIMITS.
    %if COUNT gtr 1
        %then
            %assign (T$CODE, T$CODE or G$CNTOP)
            %assign (C$COUNT, C$COUNT + 1)
    %fi
    global %name ('GP$', %number (GP$_COUNT)) : vector [C$COUNT] psect ($code$)
        initial (T$CODE, MSGADR, T$LOLIM, T$HILIM
            %if (C$ATLO or C$ATHI) neq 0
                %then , T$EXCP
            %fi
            %if COUNT gtr 1
                %then , (COUNT / 2));
                %else );
            %fi
    %assign (C$ATLO, 0)                                   !
    %assign (C$ATHI, 0)
%;
```

```
macro
    GPRMD (MSGADR, DATADR, RADIX, MASK, LOW, HIGH, DFLT, COUNT) =
    %assign (GP$_COUNT, GP$_COUNT + 1)          !GEN. UNIQUE NAMES
    %assign (T$EXCP, 0)
    %assign (T$CODE, 0)
    %assign (C$COUNT, 5)                        !5 ARGS.
    NO_A_RADIX (RADIX);
    NO_ODD_ADR (DATADR);
    OFFSET_SIZE (DATADR);
    %assign (T$CODE, G$PRMD + (DATADR * G$OFFSET))
    M$RAD (RADIX, T$TEMP);
    %assign (T$CODE, T$CODE or T$TEMP)
    M$DFLT (DFLT, T$TEMP);
    %assign (T$CODE, T$CODE or T$TEMP)
    M$EXCP (LOW,HIGH)                           !LOAD LOW AND HIGH LIMITS.
    %if COUNT gtr 1
        %then
            %assign (T$CODE, T$CODE or G$CNTOP)
            %assign (C$COUNT, C$COUNT + 1)
    %fi
    global %name ('GP$', %number (GP$_COUNT)) : vector [C$COUNT] psect ($code$)
        initial (T$CODE, MSGADR, MASK, T$LOLIM, T$HILIM
            %if (C$ATLO or C$ATHI) neq 0
                %then , T$EXCP
            %fi
            %if COUNT gtr 1
                %then , (COUNT / 2));
                %else );
            %fi
    %assign (C$ATLO, 0)                         !
    %assign (C$ATHI, 0)
%;

macro
    GPRML (MSGADR, DATADR, MASK, DFLT, COUNT) =
    %assign (GP$_COUNT, GP$_COUNT + 1)          !GEN. UNIQUE NAMES
    %assign (T$EXCP, 0)
    %assign (T$CODE, 0)
    %assign (C$COUNT, 3)                        !3 ARGS, COUNT IS 'OPTIONAL'
    NO_ODD_ADR (DATADR);
    OFFSET_SIZE (DATADR);
    %assign (T$CODE, G$PRML + (DATADR * G$OFFSET))
    M$RAD (L, T$TEMP);
    %assign (T$CODE, T$CODE or T$TEMP)
    M$DFLT (DFLT, T$TEMP);
    %assign (T$CODE, T$CODE or T$TEMP)
    %if COUNT gtr 1
        %then
            %assign (T$CODE, T$CODE or G$CNTOP)
            %assign (C$COUNT, C$COUNT + 1)
    %fi
    global %name ('GP$', %number (GP$_COUNT)) : vector [C$COUNT] psect ($code$)
        initial (T$CODE, MSGADR, MASK
            %if COUNT gtr 1
                %then , (COUNT / 2));
                %else );
            %fi
    %assign (C$ATLO, 0)                         !
    %assign (C$ATHI, 0)
%;
```

```
macro
    M$EXCP (LOW, HIGH) =
    %if (C$ATLO or C$ATHI) neq 0
        %then
            %assign (T$CODE, T$CODE or G$EXCP)
            %assign (C$COUNT, C$COUNT + 1)
    %fi
    %if C$ATLO neq 0
        %then                                           !LOW LIMIT WAS INDIRECT I.E. <@,N>
            %assign (T$EXCP, T$EXCP or G$LOLIM)
            %assign (T$LOLIM, (LOW / 2))
        %else
            %assign (T$LOLIM, LOW)
    %fi
    %if C$ATHI neq 0
        %then                                           !HI LIMIT WAS INDIRECT
            %assign (T$EXCP, T$EXCP or G$HILIM)
            %assign (T$HILIM, (HIGH / 2))
        %else
            %assign (T$HILIM, HIGH)
    %fi
%;

macro
    GP$ATLO (OFFSET) =
    %assign (C$ATLO, 1)
    OFFSET
%;

macro
    GP$ATHI (OFFSET) =
    %assign (C$ATHI, 1)
    OFFSET
%;

macro
    NO_A_RADIX (RADIX) =
    %if %identical (A, RADIX)
        %then %errormacro ('INVALID RADIX')
    %fi
%;

macro
    NO_ODD_ADR (OFFSET) =
    %assign (T$TEMP, OFFSET and 1)                      !YE OLD MASK
    %if T$TEMP neq 0
        %then %errormacro ('OFFSET IS ODD')
    %fi
%;
```

```
macro
    OFFSET_SIZE (OFFSET) =
    %if (G$OFSIZE - OFFSET) lss 0
        %then %errormacro ('OFFSET TOO BIG')
    %fi
%;

macro
    DISPLAY (ARG) =
    global GP$DISP : vector [2]          psect ($code$)
        initial (G$DISP, ARG);
%;
```