

RQDX1, RD51
RX50

RQDX1 FORMATTER
CZRQBAO

AH-T567A-MC
FICHE 1 OF 2

OCT 1983
COPYRIGHT © 1983
MADE IN USA



A large grid of approximately 15 columns and 15 rows of small, illegible text blocks, likely representing a data table or a series of microfilm frames. The text is too small to be read accurately.

RQDX1, RD51
RX50

RQDX1 FORMATTER
CZRQBA0

AH-T567A-MC
FICHE 2 OF 2

OCT 1983
COPYRIGHT © 1983
MADE IN USA



ZRQB1

RDRX DISK FORMATTER

B 1

28-Jun-1983 13:01:32
28-Jun-1983 12:58:15

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB1.B16;2 (1)

SEQ 1
Page 1

0001 MODULE ZRQB1 (%TITLE 'RDRX DISK FORMATTER'
0002 IDENT = 'REV A PATCH 00',
0003 ADDRESSING MODE (ABSOLUTÉ) ,
0004 ENVIRONMENT (NOEIS)
0005) =
0006

0007 BEGIN

0008 %(
C 0009

C 0010
C 0011 IDENTIFICATION
C 0012 -----
C 0013

C 0014 PRODUCT CODE: AC-T566A-MC
C 0015 PRODUCT NAME: CZRQBA0 RQDX1 FORMATTER
C 0016
C 0017 PRODUCT DATE: 11 JULY 1983
C 0018
C 0019 MAINTAINER: SMALL SYSTEMS DIAGNOSTIC ENGINEERING
C 0020
C 0021 AUTHOR: RUSSELL YOUNG
C 0022

C 0023
C 0024 THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
C 0025 NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
C 0026 EQUIPMENT CORPORATION. DIGITAL EQUIPMENT CORPORATION ASSUMES NO
C 0027 RESPONSIBILITY FOR ANY ERRORS THAT MAY APPEAR IN THIS DOCUMENT.
C 0028 NO RESPONSIBILITY IS ASSUMED FOR THE USE OR RELIABILITY OF
C 0029 SOFTWARE ON EQUIPMENT THAT IS NOT SUPPLIED BY DIGITAL OR ITS
C 0030 AFFILIATED COMPANIES.
C 0031

C 0032 COPYRIGHT (C) 1983 BY DIGITAL EQUIPMENT CORPORATION
C 0033 THE FOLLOWING ARE TRADEMARKS OF DIGITAL EQUIPMENT CORPORATION:
C 0034

C 0035 DIGITAL PDP UNIBUS MASSBUS
C 0036 DEC DECUS DECTAPE

ZRQB1
REV A PATCH 00

RDRX DISK FORMATTER

C 1

28-Jun-1983 13:01:32
28-Jun-1983 12:58:15

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB1.B16;2 (2)

SEQ 2
Page 2

..	C 0037	
..	C 0038	
..	C 0039	TABLE OF CONTENTS
..	C 0040	
..	C 0041	
..	C 0042	
..	C 0043	1.0 GENERAL INFORMATION
..	C 0044	
..	C 0045	1.1 PROGRAM ABSTRACT
..	C 0046	1.1.1 HOST RESIDENT PROGRAM
..	C 0047	1.1.2 CONTROLLER RESIDENT PROGRAM
..	C 0048	
..	C 0049	1.2 SYSTEM REQUIREMENTS
..	C 0050	1.2.1 HARDWARE REQUIREMENTS
..	C 0051	1.2.2 SOFTWARE REQUIREMENTS
..	C 0052	
..	C 0053	1.3 RELATED DOCUMENTS AND STANDARDS
..	C 0054	
..	C 0055	2.0 OPERATING INSTRUCTIONS
..	C 0056	2.1 HARDWARE QUESTIONS
..	C 0057	2.2 SOFTWARE QUESTIONS
..	C 0058	2.3 FORMATTER QUESTIONS
..	C 0059	
..	C 0060	3.0 RUNNING THE FORMATTER
..	C 0061	3.1 ERRORS
..	C 0062	3.2 SUCCESS

.....
C 0063
C 0064
C 0065
C 0066
C 0067
C 0068
C 0069
C 0070
C 0071
C 0072
C 0073
C 0074
C 0075
C 0076
C 0077
C 0078
C 0079
C 0080
C 0081
C 0082
C 0083
C 0084
C 0085
C 0086
C 0087
C 0088
C 0089
C 0090
C 0091
C 0092
C 0093
C 0094
C 0095
C 0096
C 0097
C 0098
C 0099
C 0100
C 0101
C 0102
C 0103
C 0104
C 0105
C 0106
C 0107
C 0108
C 0109
C 0110
C 0111
C 0112
C 0113
C 0114
C 0115
.....

1.0 GENERAL INFORMATION

1.1 PROGRAM ABSTRACT

1.1.1 HOST RESIDENT PROGRAM

This program is the front end which invokes the formatter for the RD51 disk connected to the RQDX1 controller. It interfaces with the actual formatter which is in the controller. This involves initialization of the port, invoking the actual formatter via the DUP protocol, getting needed data from the user and sending it to the controller, and finally informing the user of the final outcome.

1.1.2 CONTROLLER RESIDENT PROGRAM

When invoked by the host resident portion, this will prompt for any information it needs, and then begin running. A run consists of marking the disk as unformatted, formatting it, running three passes of a surface analysis, saving the FCT and RCT, and marking the disk as formatted.

1.2 SYSTEM REQUIREMENTS

1.2.1 HARDWARE REQUIREMENTS

LSI - 11/23 processor with 28K or more of memory, console device (EX. VT100) and RQDX1 CONTROLLER board and attached RD-51 WINCHESTER drive(s).

1.2.2 SOFTWARE REQUIREMENTS

THIS DIAGNOSTIC IS DESIGNED TO RUN WITH THE DIAGNOSTIC SUPERVISOR AS DESCRIBED IN PARAGRAPH 2.0.

1.3 RELATED DOCUMENTS AND STANDARDS

XXDP+ SUPERVISOR/USERS MANUAL CHQUS

ZRQB1
REV A PATCH 00

RDRX DISK FORMATTER

E 1

28-Jun-1983 13:01:32
28-Jun-1983 12:58:15

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB1.B16;2 (3)

SEQ 4

Page 4

; C 0116

DSDFV12 DEC DISK FORMATTING STANDARDS

.....
C 0117
C 0118
C 0119
C 0120
C 0121
C 0122
C 0123
C 0124
C 0125
C 0126
C 0127
C 0128
C 0129
C 0130
C 0131
C 0132
C 0133
C 0134
C 0135
C 0136
C 0137
C 0138
C 0139
C 0140
C 0141
C 0142
C 0143
C 0144
C 0145
C 0146
C 0147
C 0148
C 0149
C 0150
C 0151
C 0152
C 0153
C 0154
C 0155
C 0156
C 0157
C 0158
C 0159
C 0160
C 0161
C 0162
C 0163
C 0164
C 0165
C 0166
C 0167
C 0168
C 0169
.....

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 re-assembly 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.

ZRQB1
REV A PATCH 00

RDRX DISK FORMATTER

28-Jun-1983 13:01:32
28-Jun-1983 12:58:15

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB1.B16;2 (4)

: C 0170
: C 0171
: C 0172
: C 0173
: C 0174

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 0175
C 0176
C 0177
C 0178
C 0179
C 0180
C 0181
C 0182
C 0183
C 0184
C 0185
C 0186
C 0187
C 0188
C 0189
C 0190
C 0191
C 0192
C 0193
C 0194
C 0195
C 0196
C 0197
C 0198
C 0199
C 0200
C 0201
C 0202
C 0203
C 0204
C 0205
C 0206
C 0207
C 0208
C 0209
C 0210
C 0211
C 0212
C 0213
C 0214
C 0215
C 0216
C 0217
C 0218
C 0219
C 0220
C 0221
C 0222
C 0223
.....

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 in response a 0 or a 1, the only possible unit numbers for the RD51 drive on the RQDX1 controller. It will default to 0 if <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

After these DRS set up questions, the 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.

1. ENTER UNIT NUMBER TO FORMAT (0)

This will accept in response a 0 or a 1, the only possible unit numbers for the RD51 drive on the RQDX1 controller. It will default to 0 if <CR> is struck.

.....
C 0224
C 0225
C 0226
C 0227
C 0228
C 0229
C 0230
C 0231
C 0232
C 0233
C 0234
C 0235
C 0236
C 0237
C 0238
C 0239
C 0240
C 0241
C 0242
C 0243
C 0244
C 0245
C 0246
C 0247
C 0248
C 0249
C 0250
C 0251
C 0252
C 0253
C 0254
C 0255
C 0256
C 0257
C 0258
C 0259
C 0260
C 0261
C 0262
C 0263
C 0264
C 0265
C 0266
C 0267
C 0268
C 0269
C 0270
C 0271
C 0272
C 0273
C 0274
C 0275
.....

2. The next three questions select the type of format which will be done. The three modes are explained in Unibus Disk Adapter Functional Specication REV: 2.8. Since they are mutually exclusive, answering Y to any one of them will cause the formatter to skip the remaining ones and go on to the next question. Answering N to all three will cause it to default to REFORMAT mode, the same as answering Y to question 2a. In this case, the following message will be printed.

EXISTING BAD BLOCK INFORMATION USED

2a. USE EXISTING BAD BLOCK INFORMATION (N)

Answering Y to this will cause the formatter to try a REFORMAT mode format. This means it will try to read its own FCT to get its serial number, and try reading the manufacturer's bad spot record on the inner cylinder to initialize the RCT. If it fails in either attempt, it will give up and return an error.

2b. USE DOWN LINE LOAD (N)

This mode is known as RESTORE, it is not currently supported, but is included for possible future improvement. Answering Y to it will have the same result as answering Y to question 2c.

2c. CONTINUE IF BAD BLOCK INFORMATION IS INACCURATE (N)

Answering Y to this will cause a RECONSTRUCT mode format to be done. Nothing will be assumed about the disk, and the manufacturer's bad spot data, even if present, will be ignored.

3. ENTER 8 CHARACTER SERIAL NUMBER

If REFORMAT mode is selected, this question will not be asked. Otherwise, it needs 8 characters to be entered. The number is not important, and must only be unique on the controller. Thus, on a one RD51 system, any eight characters will suffice, and on a two RD51 system each one must be different - this is not difficult to achieve, since each character may be any printable ASCII symbol.

4. ENTER DATE IN MM-DD-YY FORMAT

Type in the date of formatting. It needs exactly eight characters, so January 1, 1984 must be entered as 01-01-84. This question will be asked in all modes of formatting.

C 0276
C 0277
C 0278
C 0279
C 0280
C 0281
C 0282
C 0283
C 0284
C 0285
C 0286
C 0287
C 0288
C 0289
C 0290
C 0291
C 0292
C 0293
C 0294
C 0295
C 0296
C 0297
C 0298
C 0299
C 0300
C 0301
C 0302
C 0303
C 0304
C 0305
C 0306
C 0307
C 0308
C 0309
C 0310
C 0311
C 0312
C 0313
C 0314
C 0315
C 0316
C 0317
C 0318
C 0319
C 0320
C 0321
C 0322
C 0323
C 0324
C 0325
C 0326
C 0327
C 0328

3.0 **RUNNING**

After asking the date, the actual formatting will begin. If all goes well, in just under 11 minutes it will return a successful completion message. Otherwise, it will print an error message, probably much sooner.

3.1 **Errors**

The following are the error messages generated by the formatter. If any other message appears it has been printed by DRS or XXDP+, so refer to the pertinent documentation for explanation. Errors 1, 2, and 3 will occur almost immediately, 4 can appear up to about a minute after starting, 5 from about 1 minute to 10 minutes, and 6 and 7 after 10 minutes.

1. **UNIT IS NOT WINCHESTER OR CAN NOT BE SELECTED**
The unit selected is either unavailable or is not an RD51. Check to assure it is not write protected.
2. **INITIAL FAILURE ACCESSING FCT**
The Format Control Table cannot be read. If you are trying REFORMAT mode, try RECONSTRUCT. If that fails also, the disk may be bad.
3. **FACTORY BAD BLOCK INFORMATION IS INACCESSABLE**
This will only occur if a REFORMAT is attempted and the factory bad spot data is not accessible. Run in RECONSTRUCT mode.
4. **SEEK FAILURE DURING ACTUAL FORMATTING**
There has been a hardware error during the actual formatting. If this error persists, check for hardware problems.
5. **REVECTOR LIMIT EXCEEDED**
The disk can only handle 144 bad blocks, and more than that have been found. If this persists the disk is bad.
6. **RCT WRITE FAILURE**
The formatting and surface analysis were completed successfully, but a write to the disk afterwards failed.
7. **FAILURE CLOSING FCTS**
Everything has been completed, but the disk is still marked as being unformatted.

3.2 **SUCCESS**

ZRQB1
REV A PATCH 00

RDRX DISK FORMATTER

K 1

28-Jun-1983 13:01:32
28-Jun-1983 12:58:15

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB1.B16;2 (7)

SEQ 10
Page 10

:
: C 0329
: C 0330
: C 0331
: C 0332
: C 0333
: C 0334
: C 0335
: C 0336
: 0337)%

If all goes well, in about 11 minutes the format will be complete. Successful completion is signaled by a message

FORMAT COMPLETED, xxx REVECTORED LBNS

where xxx is a decimal number. This should be a small number - performance will deteriorate if it is too large.

ZRQB1
REV A PATCH 00

RDRX DISK FORMATTER
PROGRAM HEADER

L 1

28-Jun-1983 13:01:32
28-Jun-1983 12:58:15

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB1.B16;2 (8)

SEQ 11
Page 11

```
0338 %sbttl 'PROGRAM HEADER'
0339
0340 library 'ZRQBA0.L16';           !Define RDRX Formatter Library
0341
0342 require 'BLSMAC.REQ';         !Define Bliss Macro Require file
1833
1834 !+
1835 | The psect named "code or $code$" is redefined here
1836 | to be called "aa$code". This is done to force the TKB
1837 | linker to place the programs header information starting
1838 | at absolute address 2000.
1839 | -
1840 | psect
1841 |     code = aa$code;
1842 |
1843 | literal
1844 |     DSS$NBR_OF_TESTS = 1;       !Indicates number of test in Diag
1845 |
1846 | +
1847 | The structure of a diagnostic program may contain any or all of the
1848 | ten optional sections. But five of the optional sections require a
1849 | pointer that is derived by and for the supervisor, and is located in
1850 | the header block. Therefore, in relation to the effective use of
1851 | these five pointers, the optional sections call must be coded to re-
1852 | flect usage (i.e., any,all,or none).
1853 |
1854 | The following coding possibilities exist:
1855 |
1856 |     POINTER (BGNRPT,BGNSW,BGNSFT,BGNAU,BGNDU,ERRTBL,BGNSETUP)
1857 |
1858 |     (or any subset of the args)
1859 |
1860 |     POINTER (ALL)                ; All provides pointers for all five
1861 |                                     ; sections
1862 |     POINTER (NONE)               ; None indicates to supervisor that no
1863 |                                     ; pointers are required.
1864 |                                     ; this is the default
1865 |
1866 |     No pointers are optional using bliss. Make sure the following
1867 |     sections of code are in place (in the correct skels),even if
1868 |     the sections are blank.
1869 |
1870 |     ARGUMENT      FUNCTION
1871 |     -----      -
1872 |     RPT           REPORT CODE
1873 |     SW            SOFTWARE TABLE
1874 |     SFT           SOFTWARE TABLE QUESTIONS
1875 |     AU            ADD CODE
1876 |     DU            DROP CODE
1877 |     TBL           ERROR TABLE
1878 |     SETUP         ASSEMBLED P-TABLES
1879 |
1880 |     POINTER (ALL);
```

ZRQB1
REV A PATCH 00

RDRX DISK FORMATTER
PROGRAM HEADER

28-Jun-1983 13:01:32
28-Jun-1983 12:58:15

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB1.B16;2 (9)

```
: 1881 !+
: 1882 | The program header section contains general information which des-
: 1883 | cribes the major characteristics of the diagnostic program. This in-
: 1884 | cludes, the program name, and revision and patch-order levels. The
: 1885 | header also provides space for an event flag register, and for the
: 1886 | storage of pointers, through which the supervisor may find access to
: 1887 | other key sections of the program(e.g., dispatch table, initialize and
: 1888 | clean-up code, etc.). An argument on the header gives the device type
: 1889 | if it is an XXDP+ bootable device. This enables the supervisor to pro-
: 1890 | vide load medium protection when necessary.
: 1891 | -
: 1892 HEADER (%ascii'ZRQB ', %ascii'A', %ascii'0', 1200, 0, PRI00);
```

ZRQB1
REV A PATCH 00

RDRX DISK FORMATTER
DISPATCH TABLE

28-Jun-1983 13:01:32
28-Jun-1983 12:58:15

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB1.B16;2 (10)

```

:      1893 %sbttl 'DISPATCH TABLE'
:      1894 | +
:      1895 | | The dispatch table section contains address pointers to the various
:      1896 | | tests contained within the diagnostic program. This section requires
:      1897 | | the coding of only the dispatch macro.
:      1898 | | -
:      1899 DISPATCH (DSS$NBR_OF_TESTS);
:      1900 ERR_TBL;                                !Define Supervisor Error table storage

```

ZRQB1
REV A PATCH 00

RDRX DISK FORMATTER
DEFAULT HARDWARE P-TABLE

28-Jun-1983 13:01:32
28-Jun-1983 12:58:15

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB1.B16;2 (11)

```

:
: 1901 %sbttl 'DEFAULT HARDWARE P-TABLE'
: 1902 :+
: 1903 | The default hardware P-Table contains default values of
: 1904 | the test-device parameters. The structure of this table
: 1905 | is identical to the structure of the hardware P-Tables,
: 1906 | and is used as a "template" for building the P-Tables.
: 1907 | -
: 1908 BGNHW (DFPTBL);
: 1909
: 1910 global
: 1911     HW_IP_ADRS : word initial (%'172150'),    !Define RDRX Controler IP reg
: 1912     HW_VECTOR : word initial (%'154'),      !Define RDRX interrupt vector addrs
: 1913     HW_BR_LEVEL : word initial (4),        !Define RDRX bus request level
: 1914     HW_UNIT_NO : word initial (0);        !Define RDRX unit no. to format
: 1915
: 1916 ENDDHW;

```


ZRQB1
REV A PATCH 00

RDRX DISK FORMATTER
SOFTWARE P-TABLE

28-Jun-1983 13:01:32
28-Jun-1983 12:58:15

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB1.B16;2 (12)

```

:      1917 %sbttl 'SOFTWARE P-TABLE'
:      1918 |
:      1919 | +
:      1920 | The software table contains various data used by the
:      1921 | program as operational parameters. These parameters are
:      1922 | set up at assembly time and may be varied by the operator
:      1923 | at run time.
:      1924 | -
:      1924 | BGNSW (SFPTBL);
:      1925 | +
:      1926 | All software parameter coding is done within the RDRX
:      1927 | DM code. This is per DUP functional spec compliance.
:      1928 | -
:      1929 | global
:      1930 |     SW_UNIT_NO :          word initial (0),          !default unit to 0
:      1931 |     SW_MODE   :          word initial (3);          !default mode to reconstruct
:      1932 | ENDSW;

```

ZRQB1
REV A PATCH 00

RDRX DISK FORMATTER
PROTECTION TABLE

28-Jun-1983 13:01:32
28-Jun-1983 12:58:15

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB1.B16;2 (13)

```

:      1933 %sbttl 'PROTECTION TABLE'
:      1934 |*
:      1935 | This table is used by the runtime
:      1936 | services to protect the load media.
:      1937 |
:      1938 | 1st arg =      Offset into P_Table for csr address
:      1939 | 2nd arg =      Offset into P_Table for massbus address
:      1940 | 3rd arg =      Offset into P_Table for drive number
:      1941 | -
:      1942 BGNPROT (-1, -1, -1);
:      1943 ENDPROT;

```

ZRQB1
REV A PATCH 00

RDRX DISK FORMATTER
MODULE DECLARATIONS

E 2
28-Jun-1983 13:01:32
28-Jun-1983 12:58:15

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB1.B16;2 (14)

SEQ 17
Page 17

```
1944 %sbttl 'MODULE DECLARATIONS'
1945 !+
1946 | Within BLSMAC.REQ the psect names, plit global and own,
1947 | are redefined to be aa$code. This is done to force the
1948 | tkb linker to link the header information starting at
1949 | at absolute address 2000. Redefine these psect names
1950 | back to their original names for house keeping purposes.
1951 |
1952 | Also change the attributes for the psect "global" so that
1953 | global data will not be linked starting at absolute address
1954 | 2000.
1955 | -
1956 psect
1957 |   plit = $plit$( global),
1958 |   global = $glob$(nowrite, noexecute, global, concatenate),
1959 |   own = $own$;
1960 |
1961 !+
1962 | Structure declarations used within this
1963 | module.
1964 | -
1965 |
1966 structure
1967 |
1968 | +
1969 | RDRX register accessing structure. This
1970 | structure allows RDRX register accessing
1971 | to be transportable between the PDP-11 and
1972 | VAX Diagnostic Supervisors.
1973 |
1974 | This also defines an access algorithm for
1975 | VAX to allow field reference to MBA address
1976 | space without generating machine checks.
1977 | -
1978 |
1979 RDRX [O, P, S, E] =
1980 |   begin
1981 |     local
1982 |       RCSS_REG;
1983 |
1984 |     RCSS_REG = .(RDRX + %upval*0)<0, %bpval, 0>;
1985 |     RCSS_REG
1986 |   end
1987 |   <P, S, E>;
1988 |
1989 |
```

```
1990 %sbttl 'GLOBAL DATA SECTION'
1991 |
1992 | The global data section contains data that are used
1993 | in more than one test or module.
1994 |
1995 |
1996 global
1997 |
1998 | Communication area Declarations
1999 |
2000 COM AREA : blockvector [REC_ALLOCATE + SND_ALLOCATE + HDR_SIZ, 2, word],
2001 HEAD AREA : ref block [4, word] field (HDR_FIELD),
2002 RECEIVE RING : ref blockvector [REC_ALLOCATE, 2, word] field (DSC_FIELD),
2003 SEND RING : ref blockvector [SND_ALLOCATE, 2, word] field (DSC_FIELD),
2004 REC_ENVELOPE : blockvector [REC_ALLOCATE, RB_SIZE + 2, word] field (ENV_FIELD),
2005 SND_ENVELOPE : blockvector [SND_ALLOCATE, SB_SIZE + 2, word] field (ENV_FIELD),
2006 REC_BUF : block [RECB_SIZE, word] field (RECB_FIELD),
2007 SND_BUF : vector [SNDB_SIZE, word],
2008 OUT$STD_BUF : BLOCKVECTOR [REC_ALLOCATE, 2, WORD] FIELD (OUT$FIELD),
2009 RET_EN$AD : ref block [RB_SIZE + 2, word] field (ENV_FIELD);
2010
2011 global bind
2012 |
2013 | Diagnostic supervisor printing ascii format strings.
2014 |
2015 | FMT1 = uplit (%asciz'%T'), !Print one ascii string pointer
2016 | FMT2 = uplit (%asciz'%N%A%FORMATTING LOGICAL UNIT %D2%$A%PHYSICAL UNIT %D3'),
2017 | FMT3 = uplit (%asciz'%N%A%LOGICAL UNIT %D2%$A%PHYSICAL UNIT %D2%$A%FORMAT ABORTED'),
2018 | FMT4 = uplit (%asciz'%N%A%FORMAT COMPLETED, %D2%$A REVECTORED LBNS'),
2019 | ! FMT5 = uplit (%asciz'%N%A%FORMAT ABORTED, ERROR NUMBER %D2'),
2020 |
2021 | Formatter messages (so formatter can conserve space).
2022 |
2023 | UNITS$ MSG = uplit (%asciz'ENTER UNIT TO BE FORMATTED'),
2024 | EXISTS$ MSG = uplit (%asciz'USE EXISTING BAD BLOCK INFORMATION'),
2025 | DOWNS$ MSG = uplit (%asciz'USE DOWN LINE LOAD'),
2026 | INACC$ MSG = uplit (%asciz'CONTINUE IF BAD BLOCK INFORMATION IS INACCURATE'),
2027 | DFLT$ MSG = uplit (%asciz'%N%A%EXISTING BAD BLOCK INFORMATION USED'),
2028 | SERIAL$ MSG = uplit (%asciz'ENTER 8 CHARACTER SERIAL NUMBER'),
2029 | DATES$ MSG = uplit (%asciz'ENTER DATE IN MM-DD-YY FORMAT'),
2030 |
2031 | default strings
2032 | DEF_DATE = uplit (%asciz'01-29-58'),
2033 | DEF_SERIAL = uplit (%asciz'40502179'),
2034 | Ring base address declaration
2035 |
2036 | RINGBASE = COM_AREA [REC_BASE],
2037 |
2038 |
2039 |
2040 | MSGADR = REC_BUF [MSG_TXT];
2041 |
2042 global
```

```

2043      |
2044      | : Miscellaneous data declarations
2045      |
2046      | : OVSA : WORD,                                !Overlay section starting adrs
2047      | : NXT_CRN : BYTE,                          !Stores next cmd ref number
2048      | : RET_STATUS : word initial (%o'000000'), !Saves various return status codes
2049      | : LUN : word,                              !Stores logical unit number being formatted
2050      | : PID_SAVE : word,                         !Saves proces indicator word
2051      | : UC_VER : byte,                          !Stores ucode version number
2052      | : NSD_SLOT : word,                        !Next send Descriptor slot
2053      | : NRD_SLOT : word,                        !Next receive Descriptor slot
2054      |
2055      | : Hardware P_Table storage declarations
2056      |
2057      | : RDRX_ADDR : ref RDRX field (ISD_FIELD), !Controller register access structure
2058      | : VEC_ADDR : word,                       !Interrupt vector address storage
2059      | : BR_LEVEL : word,                       !Bus request level storage
2060      | : UNIT_NO : word,                        !Unit number to format storage
2061      | : PTBL_PTR : ref vector [4, word],       !Stores P_Table base address
2062      |
2063      | : Dup Protocol data structures
2064      |
2065      | :+
2066      | : Reserved field mask structure declaration
2067      | :-
2068      | : RSVD_STRUCT : vector [4, word] preset ( !Reserved SA reg fields definitions
2069      | :   [0] = %x'00FF',                       !Step one rsvd field
2070      | :   [1] = %x'0000',                       !Step two rsvd field
2071      | :   [2] = %x'0700',                       !Step three rsvd field
2072      | :   [3] = %x'07FF'),                     !Step four rsvd & ucode field
2073      | :+
2074      | : Init Sequence Data_Structure declaration
2075      | :-
2076      |
2077      | : ISD_STRUCT : blockvector [4, 2, word] field (ISD_FIELD) preset (
2078      | :
2079      | :   Step one read SA register field declaration
2080      | :
2081      | :   [BLKO, WRDO, ERR_BIT] = 0,           !Error bit
2082      | :   [BLKO, WRDO, STP_FIELD] = %b'0001', !All step bit fields
2083      | :   [BLKO, WRDO, S1R_NV] = 0,           !No host inter vec settable adrs
2084      | :   [BLKO, WRDO, S1R_QB] = 1,          !22-bit addressing support
2085      | :   [BLKO, WRDO, S1R_DI] = 1,          !Enhanced diag implementation
2086      | :   [BLKO, WRDO, S1R_RSVD]= %o'377',   !Reserved field
2087      | :
2088      | :   Step one write SA register field declaration
2089      | :
2090      | :   [BLKO, WRD1, ERR_BIT] = 1,         !Error bit
2091      | :   [BLKO, WRD1, S1W_WR]= 0,           !Diag wrap around
2092      | :   [BLKO, WRD1, S1W_CRING]= SND_SIZ,  !Number of Send-ring slots 'pwr of 2'
2093      | :   [BLKO, WRD1, S1W_RRING]= REC_SIZ, !Number of Receive-ring slots 'pwr of 2'
2094      | :   [BLKO, WRD1, S1W_IE]= 0,          !Init Sequence interrupt request
2095      | :   [BLKO, WRD1, S1W_VADR]= %o'33',    !Interrupt vector address

```

```
2096  
2097  
2098  
2099  
2100  
2101  
2102  
2103  
2104  
2105  
2106  
2107  
2108  
2109  
2110  
2111  
2112  
2113  
2114  
2115  
2116  
2117  
2118  
2119  
2120  
2121  
2122  
2123  
2124  
2125  
2126  
2127  
2128  
2129  
2130  
2131  
2132  
2133  
2134  
2135  
2136  
2137  
2138  
2139  
2140  
2141  
2142  
2143
```

Step two read SA register field declaration

```
[BLK1, WRD0, ERR_BIT] = 0,           !Error bit  
[BLK1, WRD0, STP_FIELD] = %b'0010', !All step bit fields  
[BLK1, WRD0, S2R_PTY] = 0,           !Port type number  
[BLK1, WRD0, S2R_BIT7] = 1,          !Echoed IE bit from step one write  
[BLK1, WRD0, S2R_WR] = 0,            !Echoed bit 14 from step one write  
[BLK1, WRD0, S2R_CRING] = SND_SIZ,   !Echoed bits 3-5 from step one write  
[BLK1, WRD0, S2R_RRING] = REC_SIZ,   !Echoed bits 0-2 from step one write
```

Step two write SA register field declaration

```
[BLK1, WRD1, S2W_LRBASE] = RINGBASE, !Ring base lower address
```

NOTE:
The adapter purge interrupt is loaded within
the bgninit code due to the inability to field
select bits <1, 15, 0> from the ringbase adrs.

```
[BLK1, WRD1, S2W_PI] = 0,           !Adapter purge interrupt request
```

Step three read SA register field declaration

```
[BLK2, WRD0, ERR_BIT] = 0,           !Error bit  
[BLK2, WRD0, STP_FIELD] = %b'0100', !All step bit fields  
[BLK2, WRD0, S3R_RSVD] = %o'7',      !Reserved  
[BLK2, WRD0, S3R_IE] = 0,            !Echoed IE bit from step one write  
[BLK2, WRD0, S3R_VADR] = %o'33',     !Echoed VADR from step one write
```

Step three write SA register field declaration

```
[BLK2, WRD1, S3W_PP] = 0,            !Purge & Poll test request  
[BLK2, WRD1, S3W_HRBASE] = 0,        !Ring base high address
```

Step four read SA register field declaration

```
[BLK3, WRD0, ERR_BIT] = 0,           !Error bit  
[BLK3, WRD0, STP_FIELD] = %b'1000', !All step bit fields  
[BLK3, WRD0, S4R_RSVD] = %o'7',      !Reserved  
[BLK3, WRD0, S4R_UVER] = %o'377',    !Controller u-CODE version
```

Step four write SA register field declaration

```
[BLK3, WRD1, S4W_RSVD] = %o'377',    !Reserved  
[BLK3, WRD1, S4W_BURST] = 0,         !Max number longwords per NPR xfer  
[BLK3, WRD1, S4W_LF] = 0,           !Last fail request  
[BLK3, WRD1, S4W_GO] = 0);          !Go bit
```

```
2144 %sbttl 'GLOBAL TEXT SECTION'
2145 !+
2146 ! The global text section contains format statements,
2147 ! messages, and ASCII information that are used in
2148 ! all modules.
2149 !-
2150
2151 global bind
2152 !
2153 ! Self-detected fatal port/controller errors
2154 !
2155 PFE_STRUCT = uplit (
2156     uplit (%asciz'XNZASFTLERR- UNRECOGNIZABLE ERROR CODE'),
2157     uplit (%asciz'XNZASFTLERR- ENVELOPE/PACKET READ (PARITY OR TIMEOUT)'),
2158     uplit (%asciz'XNZASFTLERR- ENVELOPE/PACKET WRITE (PARITY OR TIMEOUT)'),
2159     uplit (%asciz'XNZASFTLERR- CONTROLLER ROM AND RAM PARITY'),
2160     uplit (%asciz'XNZASFTLERR- CONTROLLER RAM PARITY'),
2161     uplit (%asciz'XNZASFTLERR- CONTROLLER ROM PARITY'),
2162     uplit (%asciz'XNZASFTLERR- RING READ (PARITY OR TIMEOUT)'),
2163     uplit (%asciz'XNZASFTLERR- RING WRITE (PARITY OR TIMEOUT)'),
2164     uplit (%asciz'XNZASFTLERR- INTERRUPT MASTER'),
2165     uplit (%asciz'XNZASFTLERR- HOST ACCESS TIMEOUT'),
2166     uplit (%asciz'XNZASFTLERR- CREDIT LIMIT EXCEEDED'),
2167     uplit (%asciz'XNZASFTLERR- UNIBUS MASTER ERROR'),
2168     uplit (%asciz'XNZASFTLERR- DIAGNOSTIC CONTROLLER FATAL ERROR'),
2169     uplit (%asciz'XNZASFTLERR- INSTRUCTION LOOP TIMEOUT'),
2170     uplit (%asciz'XNZASFTLERR- INVALID CONNECTION IDENTIFIER'),
2171     uplit (%asciz'XNZASFTLERR- INTERRUPT WRITE'),
2172     uplit (%asciz'XNZASFTLERR- MAINTENANCE READ/WRITE INVALID REGION IDENTIFIER'),
2173     uplit (%asciz'XNZASFTLERR- MAINTENANCE WRITE LOAD TO NON-LOADABLE CONTROLLER'),
2174     uplit (%asciz'XNZASFTLERR- CONTROLLER RAM ERROR (NON-PARITY)'),
2175     uplit (%asciz'XNZASFTLERR- INIT SEQUENCE ERROR'),
2176     uplit (%asciz'XNZASFTLERR- HIGH-LEVEL PROTOCOL INCOMPATIBILITY ERROR'),
2177     uplit (%asciz'XNZASFTLERR- PURGE/POLL HARDWARE FAILURE ') : vector [22],
2178 !
2179 ! Init code error and informational
2180 ! messages
2181 !
2182 PWR_MSG = uplit (%asciz'XNZASFTLERR- INIT CODE RE-ENTERED DUE TO PWR FAIL'),
2183 ABO_MSG = uplit (%asciz'XNZASFTLERR- ABORTING HOST AND REMOTE PROGRAMS'),
2184 TO_MANY_UNITS = uplit (%asciz'XNZASFTLERR- ILLEGAL NUMBER OF UNITS SELECTED'),
2185 GOOD_NUM_UNITS = uplit (%asciz'XNZASFTLERR- LIMIT OF SIXTEEN UNITS PER FORMATING SESSION'),
2186 BOOT_FAILURE = uplit (%asciz'XNZASFTLERR- RDRX CONTROLLER INITIALIZATION ERROR'),
2187 PROTO_VIOLATION = uplit (%asciz'XNZASFTLERR- PROTOCOL VIOLATION ERROR'),
2188 PORT_INIT_ERR = uplit (%asciz'XNZASFTLERR- COMMUNICATION AREA INIT ERROR'),
2189 !
2190 ! Local load media DM module file name.ext
2191 !
2192 ! DM_FN$EXT = UPLIT (%ASCIZ'AZFMTR.SAV'),
2193 !
2194 ! Hardware parameter coding questions
2195 !
2196 HW_Q1_IP = uplit (%asciz'IP REGISTER ADDRESS'),
```

```

2197 HW_Q2_VECTOR = uplit (%asciz'INTERRUPT VECTOR ADDRESS'),
2198 HW_Q3_BR = uplit (%asciz'BUS REQUEST LEVEL '),
2199
2200 SW_Q1_UNIT = uplit (%asciz'SOFTWARE QUESTIONS ONLY APPLY UNDER APT. ENTER UNIT NUMBER '),
2201 SW_Q2_MODE = uplit (%asciz'ENTER MODE [1 = REFORMAT, 2 = RESTORE, 3 = RECONSTRUCT]'),
2202
2203 Program flow ascii string messages
2204
2205 RP_MSG = uplit (%asciz'%N%AREPORT CODE SECTION'),
2206 IN_MSG = uplit (%asciz'%N%AINIT CODE SECTION'),
2207 AUTO_MSG = uplit (%asciz'%N%AAUTODROP SECTION'),
2208 CLN_MSG = uplit (%asciz'%N%ACLEAN-UP CODE SECTION'),
2209 DU_MSG = uplit (%asciz'%N%ADROP-UNIT CODE SECTION'),
2210 AU_MSG = uplit (%asciz'%N%AADD UNIT CODE SECTION'),
2211 T1_MSG = uplit (%asciz'%N%AATEST CODE SECTION 1'),
2212 T2_MSG = uplit (%asciz'%N%AATEST CODE SECTION 2'),
2213 T3_MSG = uplit (%asciz'%N%AATEST CODE SECTION 3'),
2214 DINT_MSG = uplit (%asciz'%N%ADUP$I_SERVICE ROUTINE'),
2215 IINT_MSG = uplit (%asciz'%N%AINT$I_SERVICE ROUTINE'),
2216
2217
2218 ! Formatter error returns
2219 FMT_ERR = uplit (
2220 uplit (%asciz'%N%AFailure CLOSING FCTS'),
2221 uplit (%asciz'%N%ARCT WRITE FAILURE'),
2222 uplit (%asciz'%N%AREVECTOR LIMIT EXCEEDED'),
2223 uplit (%asciz'%N%ASEEK FAILURE DURING ACTUAL FORMATTING'),
2224 uplit (%asciz'%N%AFACTORY BAD BLOCK INFORMATION IS INACCESSABLE'),
2225 uplit (%asciz'%N%AINITIAL FAILURE ACCESSING FCT'),
2226 uplit (%asciz'%N%AUNIT IS NOT WINCHESTER OR CAN NOT BE SELECTED')) : vector [7],
2227
2228
2229 Error message structure
2230
2231 EMSG_STRUCT = uplit (
2232 uplit (%asciz'%N%$FTLERR- RESPONCE STATUS ERROR'),
2233 uplit (%asciz'%N%$FTLERR- HOST/CONTROLLER OUT OF SEQ'),
2234 uplit (%asciz'%N%$FTLERR- REMOTE PROG NOT RUNNING'),
2235 uplit (%asciz'%N%$FTLERR- UNKNOWN RETURN STATUS CODE'),
2236 uplit (%asciz'%N%$FTLERR- COM AREA INIT ERROR'),
2237 uplit (%asciz'%N%$FTLERR- PORT/HOST SYNC ERROR'),
2238 uplit (%asciz'%N%$FTLERR- MESSAGE LENGTH ERROR'),
2239 uplit (%asciz'%N%$FTLERR- UNKNOWN ENDCODE RECEIVED'),
2240 uplit (%asciz'%N%$FTLERR- ADAPTOR PURGE ERROR'),
2241 uplit (%asciz'%N%$FTLERR- UNKNOWN INTERRUPT'),
2242 uplit (%asciz'%N%$FTLERR- INIT SEQ STEP TIMED OUT'),
2243 uplit (%asciz'%N%$FTLERR- INIT SEQ COMPARE ERROR'),
2244 uplit (%asciz'%N%$FTLERR- UNEXPECTED ATTENTION END MESSAGE RECEIVED'),
2245 uplit (%asciz'%N%$FTLERR- UNEXPECTED COMMAND OPCODE IN END MESSAGE RECEIVED'),
2246 uplit (%asciz'%N%$FTLERR- UNEXPECTED SERIOUS EXCEPTION END MESSAGE RECEIVED'),
2247 uplit (%asciz'%N%$FTLERR- INVALID COMMAND END MESSAGE RECEIVED'),
2248 uplit (%asciz'%N%$FTLERR- UNKNOWN MESSAGE TYPE RECEIVED'),
2249 uplit (%asciz'%N%$FTLERR- OUTSTANDING COMMAND BUFFER FULL'),

```


ZRQB1
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL TEXT SECTION

28-Jun-1983 13:01:32
28-Jun-1983 12:58:15

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB1.B16;2 (16)

```

:      2250  uplit (%asciz'XNZASFTLERR- OUT STANDING COMMAND BUFFER OUT OF SYNC ERROR'),
:      2251  uplit (%asciz'XNZASFTLERR- UNKNOWN MESSAGE NUMBER RECEIVED'),
:      2252  uplit (%asciz'XNZASFTLERR- FILE READ ERROR'),
:      2253  uplit (%asciz'XNZASFTLERR- PORT/CONTROLLER TIMEOUT ERROR')) : vector [22];
:      2254
:      2255  end
:      2256
:      2257  eludom

```

```

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

```

000000					.PSECT	SCODES,	RO
000000	132	122	121	LSNAME::	.ASCII	/ZRQ/	
000003	102	040	040		.ASCII	/B /	
000006	000				.BYTE	0	
000007	000				.BYTE	0	
000010				LSREV::			
000010	101				.ASCII	/A/	
000011	060				.ASCII	/O/	
000012	000000G			LSUNIT::	.WORD	TSPTHV	
000014	002260			LSTIML::	.WORD	2260	
000016	000000G			LSHPCP::	.WORD	LSHARD	
000020	000000G			LSSPCP::	.WORD	LSSOFT	
000022	000140'			LSHPTP::	.WORD	LSHW	
000024	000154'			LSSPTP::	.WORD	LSSW	
000026	000000G			LSLADP::	.WORD	LSLAST	
000030	000000			LSSTA::	.WORD	0	
000032	000000			LSCO::	.WORD	0	
000034	000000			LSDTYP::	.WORD	0	
000036	000000			LSAPT::	.WORD	0	
000040	000124'			LSDTP::	.WORD	LSDISPATCH	
000042	000000			LSPRIO::	.WORD	0	
000044	000000			LSENV1::	.WORD	0	
000046	000000			LSEXP1::	.WORD	0	
000050				LSMREV::			
000050	003				.BYTE	3	
000051	003				.BYTE	3	
000052	000000			LSEF::	.WORD	0	
000054	000000				.WORD	0	
000056	000000			LSSPC::	.WORD	0	
000060	000000G			LSDEVP::	.WORD	LSDVTYP	
000062	000000G			LSREPP::	.WORD	LSRPT	
000064	000000			LSEXP4::	.WORD	0	
000066	000000			LSEXP5::	.WORD	0	
000070	000000G			LSAUT::	.WORD	LSAU	
000072	000000G			LSDUT::	.WORD	LSDU	
000074	000000			LSLUN::	.WORD	0	
000076	000000G			LSDESP::	.WORD	LSDESC	
000100	104035			LSLOAD::	.WORD	-73743	

ZRQB1
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL TEXT SECTION

M 2

28-Jun-1983 13:01:32
28-Jun-1983 12:58:15

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB1.B16;2 (16)

SEQ 25
Page 25

000045	045	101	120
000050	110	131	123
000053	111	103	101
000056	114	040	125
000061	116	111	124
000064	040	045	104
000067	063	000	000
000072	045	116	045
000075	101	114	117
000100	107	111	103
000103	101	114	040
000106	125	116	111
000111	124	040	045
000114	104	062	045
000117	123	045	101
000122	120	110	131
000125	123	111	103
000130	101	114	040
000133	125	116	111
000136	124	040	045
000141	104	062	045
000144	123	045	101
000147	106	117	122
000152	115	101	124
000155	040	101	102
000160	117	122	124
000163	105	104	000
000166	045	116	045
000171	101	106	117
000174	122	115	101
000177	124	040	103
000202	117	115	120
000205	114	105	124
000210	105	104	054
000213	040	045	104
000216	062	045	123
000221	045	101	040
000224	122	105	126
000227	105	103	124
000232	117	122	105
000235	104	040	114
000240	102	116	123
000243	000		
000244	105	116	124
000247	105	122	040
000252	125	116	111
000255	124	040	124
000260	117	040	102
000263	105	040	106
000266	117	122	115
000271	101	124	124
000274	105	104	000
000277	000		

.ASCII	/ %AP/
.ASCII	/HYS/
.ASCII	/ICA/
.ASCII	/L U/
.ASCII	/NIT/
.ASCII	/ %D/
.ASCII	/3/<00><00>
P.AAC:	.ASCII /%NZ/
	.ASCII /ALO/
	.ASCII /GIC/
	.ASCII /AL /
	.ASCII /UNI/
	.ASCII /T %/
	.ASCII /D2%/
	.ASCII /SZA/
	.ASCII /PHY/
	.ASCII /SIC/
	.ASCII /AL /
	.ASCII /UNI/
	.ASCII /T %/
	.ASCII /D2%/
	.ASCII /SZA/
	.ASCII /FOR/
	.ASCII /MAT/
	.ASCII / AB/
	.ASCII /ORT/
P.AAD:	.ASCII /ED/<00>
	.ASCII /%NZ/
	.ASCII /AFO/
	.ASCII /RMA/
	.ASCII /T C/
	.ASCII /OMP/
	.ASCII /LET/
	.ASCII /ED,/
	.ASCII / %D/
	.ASCII /2%S/
	.ASCII /XA /
	.ASCII /REV/
	.ASCII /ECT/
	.ASCII /ORE/
	.ASCII /D L/
	.ASCII /BNS/
	.ASCII <00>
P.AAE:	.ASCII /ENT/
	.ASCII /ER /
	.ASCII /UNI/
	.ASCII /T T/
	.ASCII /O B/
	.ASCII /E F/
	.ASCII /ORM/
	.ASCII /ATT/
	.ASCII /ED/<00>
	.ASCII <00>

ZRQB1
REV A PATCH 00 RDRX DISK FORMATTER
GLOBAL TEXT SECTION

000534	101	103	124	.ASCII	/ACT/	
000537	105	122	040	.ASCII	/ER /	
000542	123	105	122	.ASCII	/SER/	
000545	111	101	114	.ASCII	/IAL/	
000550	040	116	125	.ASCII	/ NU/	
000553	115	102	105	.ASCII	/MBE/	
000556	122	000		.ASCII	/R/<00>	
000560	105	116	124	P.AAK:	.ASCII	/ENT/
000563	105	122	040	.ASCII	/ER /	
000566	104	101	124	.ASCII	/DAT/	
000571	105	040	111	.ASCII	/E I/	
000574	116	040	115	.ASCII	/N M/	
000577	115	055	104	.ASCII	/M-D/	
000602	104	055	131	.ASCII	/D-Y/	
000605	131	040	106	.ASCII	/Y F/	
000610	117	122	115	.ASCII	/ORM/	
000613	101	124	000	.ASCII	/AT/<00>	
000616	060	061	055	P.AAL:	.ASCII	/01-/
000621	062	071	055	.ASCII	/29-/	
000624	065	070	000	.ASCII	/58/<00>	
000627	000			.ASCII	<00>	
000630	064	060	065	P.AAM:	.ASCII	/405/
000633	060	062	061	.ASCII	/021/	
000636	067	071	000	.ASCII	/79/<00>	
000641	000			.ASCII	<00>	
000642	045	116	045	P.AAO:	.ASCII	/XNZ/
000645	101	044	106	.ASCII	/ASF/	
000650	115	124	105	.ASCII	/MTE/	
000653	122	122	055	.ASCII	/RR-/	
000656	040	125	116	.ASCII	/ UN/	
000661	122	105	103	.ASCII	/REC/	
000664	117	107	116	.ASCII	/OGN/	
000667	111	132	101	.ASCII	/IZA/	
000672	102	114	105	.ASCII	/BLE/	
000675	040	105	122	.ASCII	/ ER/	
000700	122	117	122	.ASCII	/ROR/	
000703	040	103	117	.ASCII	/ CO/	
000706	104	105	000	.ASCII	/DE/<00>	
000711	000			.ASCII	<00>	
000712	045	116	045	P.AAP:	.ASCII	/XNZ/
000715	101	044	106	.ASCII	/ASF/	
000720	124	114	105	.ASCII	/TLE/	
000723	122	122	055	.ASCII	/RR-/	
000726	040	105	116	.ASCII	/ EN/	
000731	126	105	114	.ASCII	/VEL/	
000734	117	120	105	.ASCII	/OPE/	
000737	057	120	101	.ASCII	<57>/PA/	
000742	103	113	105	.ASCII	/CKE/	
000745	124	040	122	.ASCII	/T R/	
000750	105	101	104	.ASCII	/EAD/	
000753	040	050	120	.ASCII	/ (P/	
000756	101	122	111	.ASCII	/ARI/	
000761	124	131	040	.ASCII	/TY /	

ZRQB1 RDRX DISK FORMATTER
REV A PATCH 00 GLOBAL TEXT SECTION

000764	117	122	040
000767	124	111	115
000772	105	117	125
000775	124	051	000
001000	045	116	045
001003	101	044	106
001006	124	114	105
001011	122	122	055
001014	040	105	116
001017	126	105	114
001022	117	120	105
001025	057	120	101
001030	103	113	105
001033	124	040	127
001036	122	111	124
001041	105	040	050
001044	120	101	122
001047	111	124	131
001052	040	117	122
001055	040	124	111
001060	115	105	117
001063	125	124	051
001066	000	000	
001070	045	116	045
001073	101	044	106
001076	124	114	105
001101	122	122	055
001104	040	103	117
001107	116	124	122
001112	117	114	114
001115	105	122	040
001120	122	117	115
001123	040	101	116
001126	104	040	122
001131	101	115	040
001134	120	101	122
001137	111	124	131
001142	000	000	
001144	045	116	045
001147	101	044	106
001152	124	114	105
001155	122	122	055
001160	040	103	117
001163	116	124	122
001166	117	114	114
001171	105	122	040
001174	122	101	115
001177	040	120	101
001202	122	111	124
001205	131	000	000
001210	045	116	045
001213	101	044	106
001216	124	114	105

	.ASCII	/OR /
	.ASCII	/TIM/
	.ASCII	/EOU/
P.AAQ:	.ASCII	/T)/<00>
	.ASCII	/XNZ/
	.ASCII	/ASF/
	.ASCII	/TLE/
	.ASCII	/RR-/
	.ASCII	/ EN/
	.ASCII	/VEL/
	.ASCII	/OPE/
	.ASCII	<57>/PA/
	.ASCII	/CKE/
	.ASCII	/T W/
	.ASCII	/RIT/
	.ASCII	/E (/
	.ASCII	/PAR/
	.ASCII	/ITY/
	.ASCII	/ OR/
	.ASCII	/ TI/
	.ASCII	/MEO/
	.ASCII	/UT)/
	.ASCII	<00><00>
P.AAR:	.ASCII	/XNZ/
	.ASCII	/ASF/
	.ASCII	/TLE/
	.ASCII	/RR-/
	.ASCII	/ CO/
	.ASCII	/NTR/
	.ASCII	/OLL/
	.ASCII	/ER /
	.ASCII	/ROM/
	.ASCII	/ AN/
	.ASCII	/D R/
	.ASCII	/AM /
	.ASCII	/PAR/
	.ASCII	/ITY/
	.ASCII	<00><00>
P.AAS:	.ASCII	/XNZ/
	.ASCII	/ASF/
	.ASCII	/TLE/
	.ASCII	/RR-/
	.ASCII	/ CO/
	.ASCII	/NTR/
	.ASCII	/OLL/
	.ASCII	/ER /
	.ASCII	/RAM/
	.ASCII	/ PA/
	.ASCII	/RIT/
	.ASCII	/Y/<00><00>
P.AAT:	.ASCII	/XNZ/
	.ASCII	/ASF/
	.ASCII	/TLE/

ZRQB1 RDRX DISK FORMATTER
REV A PATCH 00 GLOBAL TEXT SECTION

001221	122	122	055	.ASCII	/RR-/
001224	040	103	117	.ASCII	/ CO/
001227	116	124	122	.ASCII	/NTR/
001232	117	114	114	.ASCII	/OLL/
001235	105	122	040	.ASCII	/ER /
001240	122	117	115	.ASCII	/ROM/
001243	040	120	101	.ASCII	/ PA/
001246	122	111	124	.ASCII	/RIT/
001251	131	000	000	.ASCII	/Y/<00><00>
001254	045	116	045	P.AAU:	.ASCII /XN%/
001257	101	044	106	.ASCII	/ASF/
001262	124	114	105	.ASCII	/TLE/
001265	122	122	055	.ASCII	/RR-/
001270	040	122	111	.ASCII	/ RI/
001273	116	107	040	.ASCII	/NG /
001276	122	105	101	.ASCII	/REA/
001301	104	040	050	.ASCII	/D (/
001304	120	101	122	.ASCII	/PAR/
001307	111	124	131	.ASCII	/ITY/
001312	040	117	122	.ASCII	/ OR/
001315	040	124	111	.ASCII	/ TI/
001320	115	105	117	.ASCII	/MEO/
001323	125	124	051	.ASCII	/UT)/
001326	000	000		.ASCII	<00><00>
001330	045	116	045	P.AAV:	.ASCII /XN%/
001333	101	044	106	.ASCII	/ASF/
001336	124	114	105	.ASCII	/TLE/
001341	122	122	055	.ASCII	/RR-/
001344	040	122	111	.ASCII	/ RI/
001347	116	107	040	.ASCII	/NG /
001352	127	122	111	.ASCII	/WRI/
001355	124	105	040	.ASCII	/TE /
001360	050	120	101	.ASCII	/(PA/
001363	122	111	124	.ASCII	/RIT/
001366	131	040	117	.ASCII	/Y O/
001371	122	040	124	.ASCII	/R T/
001374	111	115	105	.ASCII	/IME/
001377	117	125	124	.ASCII	/OUT/
001402	051	000		.ASCII	/)/<00>
001404	045	116	045	P.AAW:	.ASCII /XN%/
001407	101	044	106	.ASCII	/ASF/
001412	124	114	105	.ASCII	/TLE/
001415	122	122	055	.ASCII	/RR-/
001420	040	111	116	.ASCII	/ IN/
001423	124	105	122	.ASCII	/TER/
001426	122	125	120	.ASCII	/RUP/
001431	124	040	115	.ASCII	/T M/
001434	101	123	124	.ASCII	/AST/
001437	105	122	000	.ASCII	/ER/<00>
001442	045	116	045	P.AAX:	.ASCII /XN%/
001445	101	044	106	.ASCII	/ASF/
001450	124	114	105	.ASCII	/TLE/
001453	122	122	055	.ASCII	/RR-/

28-Jun-1983 13:01:32
28-Jun-1983 12:58:15

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB1.B16;2 (16)

ZRQB1
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL TEXT SECTION

001711	123	124	122	.ASCII	/STR/
001714	125	103	124	.ASCII	/UCT/
001717	111	117	116	.ASCII	/ION/
001722	040	114	117	.ASCII	/LO/
001725	117	120	040	.ASCII	/OP /
001730	124	111	115	.ASCII	/TIM/
001733	105	117	125	.ASCII	/EOU/
001736	124	000		.ASCII	/T/<00>
001740	045	116	045	P.ABC: .ASCII	/XN%/
001743	101	044	106	.ASCII	/ASF/
001746	124	114	105	.ASCII	/TLE/
001751	122	122	055	.ASCII	/RR-/
001754	040	111	116	.ASCII	/ IN/
001757	126	101	114	.ASCII	/VAL/
001762	111	104	040	.ASCII	/ID /
001765	103	117	116	.ASCII	/CON/
001770	116	105	103	.ASCII	/NEC/
001773	124	111	117	.ASCII	/TIO/
001776	116	040	111	.ASCII	/N I/
002001	104	105	116	.ASCII	/DEN/
002004	124	111	106	.ASCII	/TIF/
002007	111	105	122	.ASCII	/IER/
002012	000	000		.ASCII	<00><00>
002014	045	116	045	P.ABD: .ASCII	/XN%/
002017	101	044	106	.ASCII	/ASF/
002022	124	114	105	.ASCII	/TLE/
002025	122	122	055	.ASCII	/RR-/
002030	040	111	116	.ASCII	/ IN/
002033	124	105	122	.ASCII	/TER/
002036	122	125	120	.ASCII	/RUP/
002041	124	040	127	.ASCII	/T W/
002044	122	111	124	.ASCII	/RIT/
002047	105	000	000	.ASCII	/E/<00><00>
002052	045	116	045	P.ABE: .ASCII	/XN%/
002055	101	044	106	.ASCII	/ASF/
002060	124	114	105	.ASCII	/TLE/
002063	122	122	055	.ASCII	/RR-/
002066	040	115	101	.ASCII	/ MA/
002071	111	116	124	.ASCII	/INT/
002074	105	116	101	.ASCII	/ENA/
002077	116	103	105	.ASCII	/NCE/
002102	040	122	105	.ASCII	/ RE/
002105	101	104	057	.ASCII	/AD/<57>
002110	127	122	111	.ASCII	/WRI/
002113	124	105	040	.ASCII	/TE /
002116	111	116	126	.ASCII	/INV/
002121	101	114	111	.ASCII	/ALI/
002124	104	040	122	.ASCII	/D R/
002127	105	107	111	.ASCII	/EGI/
002132	117	116	040	.ASCII	/ON /
002135	111	104	105	.ASCII	/IDE/
002140	116	124	111	.ASCII	/NTI/
002143	106	111	105	.ASCII	/FIE/

ZROB1
REV A PATCH 00 RDRX DISK FORMATTER
GLOBAL TEXT SECTION

002610	045	116	045
002613	101	044	106
002616	124	114	105
002621	122	122	055
002624	040	111	116
002627	111	124	040
002632	103	117	104
002635	105	040	122
002640	105	055	105
002643	116	124	105
002646	122	105	104
002651	040	104	125
002654	105	040	124
002657	117	040	120
002662	127	122	040
002665	106	101	111
002670	114	000	
002672	045	116	045
002675	101	044	106
002700	124	114	105
002703	122	122	055
002706	040	101	102
002711	117	122	124
002714	111	116	107
002717	040	110	117
002722	123	124	040
002725	101	116	104
002730	040	122	105
002733	115	117	124
002736	105	040	120
002741	122	117	107
002744	122	101	115
002747	123	000	000
002752	045	116	045
002755	101	044	106
002760	124	114	105
002763	122	122	055
002766	040	111	114
002771	114	105	107
002774	101	114	040
002777	116	125	115
003002	102	105	122
003005	040	117	106
003010	040	125	116
003013	111	124	123
003016	040	123	105
003021	114	105	103
003024	124	105	104
003027	000		
003030	045	116	045
003033	101	044	106
003036	124	114	105
003041	122	122	055

P.ABK:	.ASCII	/XNZ/
	.ASCII	/ASF/
	.ASCII	/TLE/
	.ASCII	/RR-/
	.ASCII	/ IN/
	.ASCII	/IT /
	.ASCII	/COD/
	.ASCII	/E R/
	.ASCII	/E-E/
	.ASCII	/NTE/
	.ASCII	/RED/
	.ASCII	/ DU/
	.ASCII	/E T/
	.ASCII	/O P/
	.ASCII	/WR /
	.ASCII	/FAI/
	.ASCII	/L/<00>
P.ABL:	.ASCII	/XNZ/
	.ASCII	/ASF/
	.ASCII	/TLE/
	.ASCII	/RR-/
	.ASCII	/ AB/
	.ASCII	/ORT/
	.ASCII	/ING/
	.ASCII	/ HO/
	.ASCII	/ST /
	.ASCII	/AND/
	.ASCII	/ RE/
	.ASCII	/MOT/
	.ASCII	/E P/
	.ASCII	/ROG/
	.ASCII	/RAM/
	.ASCII	/S/<00><00>
P.ABM:	.ASCII	/XNZ/
	.ASCII	/ASF/
	.ASCII	/TLE/
	.ASCII	/RR-/
	.ASCII	/ IL/
	.ASCII	/LEG/
	.ASCII	/AL /
	.ASCII	/NUM/
	.ASCII	/BER/
	.ASCII	/ OF/
	.ASCII	/ UN/
	.ASCII	/ITS/
	.ASCII	/ SE/
	.ASCII	/LEC/
	.ASCII	/TED/
	.ASCII	<00>
P.ABN:	.ASCII	/XNZ/
	.ASCII	/ASF/
	.ASCII	/TLE/
	.ASCII	/RR-/

ZRQB1 RDRX DISK FORMATTER
REV A PATCH 00 GLOBAL TEXT SECTION

003044	040	114	111	.ASCII	/ LI/
003047	115	111	124	.ASCII	/MIT/
003052	040	117	106	.ASCII	/ OF/
003055	040	123	111	.ASCII	/ SI/
003060	130	124	105	.ASCII	/XTE/
003063	105	116	040	.ASCII	/EN /
003066	125	116	111	.ASCII	/UNI/
003071	124	123	040	.ASCII	/TS /
003074	120	105	122	.ASCII	/PER/
003077	040	106	117	.ASCII	/ FO/
003102	122	115	101	.ASCII	/RMA/
003105	124	111	116	.ASCII	/TIN/
003110	107	040	123	.ASCII	/G S/
003113	105	123	123	.ASCII	/ESS/
003116	111	117	116	.ASCII	/ION/
003121	000			.ASCII	<00>
003122	045	116	045	P.ABO: .ASCII	/XNZ/
003125	101	044	106	.ASCII	/ASF/
003130	124	114	105	.ASCII	/TLE/
003133	122	122	055	.ASCII	/RR-/
003136	040	122	104	.ASCII	/ RD/
003141	122	130	040	.ASCII	/RX /
003144	103	117	116	.ASCII	/CON/
003147	124	122	117	.ASCII	/TRO/
003152	114	114	105	.ASCII	/LLE/
003155	122	040	111	.ASCII	/R I/
003160	116	111	124	.ASCII	/NIT/
003163	111	101	114	.ASCII	/IAL/
003166	111	132	101	.ASCII	/IZA/
003171	124	111	117	.ASCII	/TIO/
003174	116	040	105	.ASCII	/N E/
003177	122	122	117	.ASCII	/RRO/
003202	122	000		.ASCII	/R/<00>
003204	045	116	045	P.ABP: .ASCII	/XNZ/
003207	101	044	106	.ASCII	/ASF/
003212	124	114	105	.ASCII	/TLE/
003215	122	122	055	.ASCII	/RR-/
003220	040	120	122	.ASCII	/ PR/
003223	117	124	117	.ASCII	/OTO/
003226	103	117	114	.ASCII	/COL/
003231	040	126	111	.ASCII	/ VI/
003234	117	114	101	.ASCII	/OLA/
003237	124	111	117	.ASCII	/TIO/
003242	116	040	105	.ASCII	/N E/
003245	122	122	117	.ASCII	/RRO/
003250	122	000		.ASCII	/R/<00>
003252	045	116	045	P.ABQ: .ASCII	/XNZ/
003255	101	044	106	.ASCII	/ASF/
003260	124	114	105	.ASCII	/TLE/
003263	122	122	055	.ASCII	/RR-/
003266	040	103	117	.ASCII	/ CO/
003271	115	115	125	.ASCII	/MMU/
003274	116	111	103	.ASCII	/NIC/

ZRQB1
REV A PATCH 00 RDRX DISK FORMATTER
GLOBAL TEXT SECTION

003277	101	124	111	.ASCII	/ATI/
003302	117	116	040	.ASCII	/ON /
003305	101	122	105	.ASCII	/ARE/
003310	101	040	111	.ASCII	/A I/
003313	116	111	124	.ASCII	/NIT/
003316	040	105	122	.ASCII	/ ER/
003321	122	117	122	.ASCII	/ROR/
003324	000	000		.ASCII	<00><00>
003326	111	120	040	P.ABR:	.ASCII /IP /
003331	040	040	122	.ASCII	/ R/
003334	105	107	111	.ASCII	/EGI/
003337	123	124	105	.ASCII	/STE/
003342	122	040	040	.ASCII	/R /
003345	040	101	104	.ASCII	/ AD/
003350	104	122	105	.ASCII	/DRE/
003353	123	123	000	P.ABS:	.ASCII /SS/<00>
003356	111	116	124	.ASCII	/INT/
003361	105	122	122	.ASCII	/ERR/
003364	125	120	124	.ASCII	/UPT/
003367	040	126	105	.ASCII	/ VE/
003372	103	124	117	.ASCII	/CTO/
003375	122	040	101	.ASCII	/R A/
003400	104	104	122	.ASCII	/DDR/
003403	105	123	123	.ASCII	/ESS/
003406	000	000		.ASCII	<00><00>
003410	102	125	123	P.ABT:	.ASCII /BUS/
003413	040	040	040	.ASCII	/ /
003416	122	105	121	.ASCII	/REQ/
003421	125	105	123	.ASCII	/UES/
003424	124	040	040	.ASCII	/T /
003427	040	114	105	.ASCII	/ LE/
003432	126	105	114	.ASCII	/VEL/
003435	040	040	000	.ASCII	/ /<00>
003440	123	117	106	P.ABU:	.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	040	000	.ASCII	/R /<00>

ZRQB1
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL TEXT SECTION

28-Jun-1983 13:01:32
28-Jun-1983 12:58:15

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB1.B16;2 (16)

003534	105	116	124
003537	105	122	040
003542	115	117	104
003545	105	040	133
003550	061	040	075
003553	040	122	105
003556	106	117	122
003561	115	101	124
003564	054	040	062
003567	040	075	040
003572	122	105	123
003575	124	117	122
003600	105	054	040
003603	063	040	075
003606	040	122	105
003611	103	117	116
003614	123	124	122
003617	125	103	124
003622	135	000	
003624	045	116	045
003627	101	106	101
003632	111	114	125
003635	122	105	040
003640	103	114	117
003643	123	111	116
003646	107	040	106
003651	103	124	123
003654	000	000	
003656	045	116	045
003661	101	122	103
003664	124	040	127
003667	122	111	124
003672	105	040	106
003675	101	111	114
003700	125	122	105
003703	000		
003704	045	116	045
003707	101	122	105
003712	126	105	103
003715	124	117	122
003720	040	114	111
003723	115	111	124
003726	040	105	130
003731	103	105	105
003734	104	105	104
003737	000		
003740	045	116	045
003743	101	123	105
003746	105	113	040
003751	106	101	111
003754	114	125	122
003757	105	040	104
003762	125	122	111

P.ABV: .ASCII /ENT/
 .ASCII /ER /
 .ASCII /MOD/
 .ASCII /E [/
 .ASCII /1 =/
 .ASCII / RE/
 .ASCII /FOR/
 .ASCII /MAT/
 .ASCII /, 2/
 .ASCII / = /
 .ASCII /RES/
 .ASCII /TOR/
 .ASCII /E. /
 .ASCII /3 =/
 .ASCII / RE/
 .ASCII /CON/
 .ASCII /STR/
 .ASCII /UCT/
 .ASCII /J/<00>
 P.ABX: .ASCII /%N%/
 .ASCII /AFA/
 .ASCII /ILU/
 .ASCII /RE /
 .ASCII /CLO/
 .ASCII /SIN/
 .ASCII /G F/
 .ASCII /CTS/
 .ASCII <00><00>
 P.ABY: .ASCII /%N%/
 .ASCII /ARC/
 .ASCII /T W/
 .ASCII /RIT/
 .ASCII /E F/
 .ASCII /AIL/
 .ASCII /URE/
 .ASCII <00>
 P.ABZ: .ASCII /%N%/
 .ASCII /ARE/
 .ASCII /VEC/
 .ASCII /TOR/
 .ASCII / LI/
 .ASCII /MIT/
 .ASCII / EX/
 .ASCII /CEE/
 .ASCII /DED/
 .ASCII <00>
 P.ACA: .ASCII /%N%/
 .ASCII /ASE/
 .ASCII /EK /
 .ASCII /FAI/
 .ASCII /LUR/
 .ASCII /E D/
 .ASCII /URI/

ZRQB1 RDRX DISK FORMATTER
REV A PATCH 00 GLOBAL TEXT SECTION

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.ACB: .ASCII	/XN%/
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.ACC: .ASCII	/XN%/
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.ACD: .ASCII	/XN%/
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>

ZRQB1
REV

.....

004220	003624*		
004222	003656*		
004224	003704*		
004226	003740*		
004230	004012*		
004232	004074*		
004234	004136*		
004236	045	116	045
004241	101	044	106
004244	124	114	105
004247	122	122	055
004252	040	122	105
004255	123	120	117
004260	116	103	105
004263	040	123	124
004266	101	124	125
004271	123	040	105
004274	122	122	117
004277	122	000	000
004302	045	116	045
004305	101	044	106
004310	124	114	105
004313	122	122	055
004316	040	110	117
004321	123	124	057
004324	103	117	116
004327	124	122	117
004332	114	114	105
004335	122	040	117
004340	125	124	040
004343	117	106	040
004346	123	105	121
004351	000		
004352	045	116	045
004355	101	044	106
004360	124	114	105
004363	122	122	055
004366	040	122	105
004371	115	117	124
004374	105	040	120
004377	122	117	107
004402	040	116	117
004405	124	040	122
004410	125	116	116
004413	111	116	107
004416	000	000	
004420	045	116	045
004423	101	044	106
004426	124	114	105
004431	122	122	055
004434	040	125	116
004437	113	116	117
004442	127	116	040

P.ABW:	.WORD	P.ABX
	.WORD	P.ABY
	.WORD	P.ABZ
	.WORD	P.ACA
	.WORD	P.ACB
	.WORD	P.ACC
	.WORD	P.ACD
P.ACF:	.ASCII	/XN%/
	.ASCII	/ASF/
	.ASCII	/TLE/
	.ASCII	/RR-/
	.ASCII	/ RE/
	.ASCII	/SPO/
	.ASCII	/NCE/
	.ASCII	/ ST/
	.ASCII	/ATU/
	.ASCII	/S E/
	.ASCII	/RRO/
P.ACG:	.ASCII	/R/<00><00>
	.ASCII	/XN%/
	.ASCII	/ASF/
	.ASCII	/TLE/
	.ASCII	/RR-/
	.ASCII	/ HO/
	.ASCII	/ST/<57>
	.ASCII	/CON/
	.ASCII	/TRO/
	.ASCII	/LLE/
	.ASCII	/R O/
	.ASCII	/UT /
	.ASCII	/OF /
	.ASCII	/SEQ/
	.ASCII	<00>
P.ACH:	.ASCII	/XN%/
	.ASCII	/ASF/
	.ASCII	/TLE/
	.ASCII	/RR-/
	.ASCII	/ RE/
	.ASCII	/MOT/
	.ASCII	/E P/
	.ASCII	/ROG/
	.ASCII	/ NO/
	.ASCII	/T R/
	.ASCII	/UNN/
	.ASCII	/ING/
	.ASCII	<00><00>
P.ACI:	.ASCII	/XN%/
	.ASCII	/ASF/
	.ASCII	/TLE/
	.ASCII	/RR-/
	.ASCII	/ UN/
	.ASCII	/KNO/
	.ASCII	/WN /

.....

ZRQB1
REV A PATCH 00 RDRX DISK FORMATTER
GLOBAL TEXT SECTION

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.ACJ:	.ASCII /XN%/
004473	101	044	106		.ASCII /ASF/
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.ACK:	.ASCII /XN%/
004535	101	044	106		.ASCII /ASF/
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.ACL:	.ASCII /XN%/
004577	101	044	106		.ASCII /ASF/
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.ACM:	.ASCII /XN%/
004641	101	044	106		.ASCII /ASF/
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/

ZRQB1 RDRX DISK FORMATTER
REV A PATCH 00 GLOBAL TEXT SECTION

004674	105	103	105		.ASCII	/ECE/
004677	111	126	105		.ASCII	/IVE/
004702	104	000			.ASCII	/D/<00>
004704	045	116	045	P.ACN:	.ASCII	/XN%/
004707	101	044	106		.ASCII	/ASF/
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.ACO:	.ASCII	/XN%/
004751	101	044	106		.ASCII	/ASF/
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.ACP:	.ASCII	/XN%/
005011	101	044	106		.ASCII	/ASF/
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.ACQ:	.ASCII	/XN%/
005057	101	044	106		.ASCII	/ASF/
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.ACR:	.ASCII	/XN%/
005123	101	044	106		.ASCII	/ASF/

.....

ZRQB1
REV A PATCH 00 RDRX DISK FORMATTER
GLOBAL TEXT SECTION

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.ACS: .ASCII	/XN%/
005213	101	044	106	.ASCII	/ASF/
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.ACT: .ASCII	/XN%/
005313	101	044	106	.ASCII	/ASF/
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 /

28-Jun-1983 13:01:32
28-Jun-1983 12:58:15

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB1.B16;2 (16)

ZRQB1
REV A PATCH 00 RDRX DISK FORMATTER
GLOBAL TEXT SECTION

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.ACU: .ASCII	/XNZ/
005413	101	044	106	.ASCII	/ASF/
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.ACV: .ASCII	/XNZ/
005475	101	044	106	.ASCII	/ASF/
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.ACW: .ASCII	/XNZ/
005551	101	044	106	.ASCII	/ASF/
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/

28-Jun-1983 13:01:32
28-Jun-1983 12:58:15

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB1.B16;2 (16)

ZRQB1 RDRX DISK FORMATTER
REV A PATCH 00 GLOBAL TEXT SECTION

005615	040	106	125
005620	114	114	000
005623	000		
005624	045	116	045
005627	101	044	106
005632	124	114	105
005635	122	122	055
005640	040	117	125
005643	124	040	123
005646	124	101	116
005651	104	111	116
005654	107	040	103
005657	117	115	115
005662	101	116	104
005665	040	102	125
005670	106	106	105
005673	122	040	117
005676	125	124	040
005701	117	106	040
005704	123	131	116
005707	103	040	105
005712	122	122	117
005715	122	000	000
005720	045	116	045
005723	101	044	106
005726	124	114	105
005731	122	122	055
005734	040	125	116
005737	113	116	117
005742	127	116	040
005745	115	105	123
005750	123	101	107
005753	105	040	116
005756	125	115	102
005761	105	122	040
005764	122	105	103
005767	105	111	126
005772	105	104	000
005775	000		
005776	045	116	045
006001	101	044	106
006004	124	114	105
006007	122	122	055
006012	040	106	111
006015	114	105	040
006020	122	105	101
006023	104	040	105
006026	122	122	117
006031	122	000	000
006034	045	116	045
006037	101	044	106
006042	124	114	105
006045	122	122	055

	.ASCII	/ FU/	
	.ASCII	/LL/<00>	
	.ASCII	<00>	
P.ACX:	.ASCII	/XNZ/	
	.ASCII	/ASF/	
	.ASCII	/TLE/	
	.ASCII	/RR-/	
	.ASCII	/ OU/	
	.ASCII	/T S/	
	.ASCII	/TAN/	
	.ASCII	/DIN/	
	.ASCII	/G C/	
	.ASCII	/OMM/	
	.ASCII	/AND/	
	.ASCII	/ BU/	
	.ASCII	/FFE/	
	.ASCII	/R O/	
	.ASCII	/UT /	
	.ASCII	/OF /	
	.ASCII	/SYN/	
	.ASCII	/C E/	
	.ASCII	/RRO/	
	.ASCII	/R/<00><00>	
P.ACY:	.ASCII	/XNZ/	
	.ASCII	/ASF/	
	.ASCII	/TLE/	
	.ASCII	/RR-/	
	.ASCII	/ UN/	
	.ASCII	/KNO/	
	.ASCII	/WN /	
	.ASCII	/MES/	
	.ASCII	/SAG/	
	.ASCII	/E N/	
	.ASCII	/UMB/	
	.ASCII	/ER /	
	.ASCII	/REC/	
	.ASCII	/EIV/	
	.ASCII	/ED/<00>	
	.ASCII	<00>	
P.ACZ:	.ASCII	/XNZ/	
	.ASCII	/ASF/	
	.ASCII	/TLE/	
	.ASCII	/RR-/	
	.ASCII	/ FI/	
	.ASCII	/LE /	
	.ASCII	/REA/	
	.ASCII	/D E/	
	.ASCII	/RRO/	
	.ASCII	/R/<00><00>	
P.ADA:	.ASCII	/XNZ/	
	.ASCII	/ASF/	
	.ASCII	/TLE/	
	.ASCII	/RR-/	

ZRQB1 RDRX DISK FORMATTER
REV A PATCH 00 GLOBAL TEXT SECTION

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.ACE: .WORD	P.ACF
006112	004302'			.WORD	P.ACG
006114	004352'			.WORD	P.ACH
006116	004420'			.WORD	P.ACI
006120	004470'			.WORD	P.ACJ
006122	004532'			.WORD	P.ACK
006124	004574'			.WORD	P.ACL
006126	004636'			.WORD	P.ACM
006130	004704'			.WORD	P.ACN
006132	004746'			.WORD	P.ACO
006134	005006'			.WORD	P.ACP
006136	005054'			.WORD	P.ACQ
006140	005120'			.WORD	P.ACR
006142	005210'			.WORD	P.ACS
006144	005310'			.WORD	P.ACT
006146	005410'			.WORD	P.ACU
006150	005472'			.WORD	P.ACV
006152	005546'			.WORD	P.ACW
006154	005624'			.WORD	P.ACX
006156	005720'			.WORD	P.ACY
006160	005776'			.WORD	P.ACZ
006162	006034'			.WORD	P.ADA

000000		.PSECT	\$GLOBS, 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

ZRQB1 RDRX DISK FORMATTER
REV A PATCH 00 GLOBAL TEXT SECTION

H 4
28-Jun-1983 13:01:32
28-Jun-1983 12:58:15

SEQ 46
Page 46
VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB1.B16;2 (16)

001430		OUTSSTD.BUF::		
		.BLKW	10	
001450		RET.ENSAD::		
		.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	005777	ISD.STRUCT::		
		.WORD	5777	
001514	111033	.WORD	-66745	
001516	010222	.WORD	10222	
001520	000010	.WORD	RINGBASE	
001522	023433	.WORD	23433	
001524	000000	.WORD	0	
001526	043777	.WORD	43777	
001530	177400	.WORD	-400	
		.GLOBL	LSSOFT, TSPTHV, LSRPT, LSINIT	
		.GLOBL	LSCLEAN, LSLAST, LSHARD, LSDVTYP	
		.GLOBL	LSDESC, LSDU, LSAU, LSAUTO, T1	
000126		LSERRTBL==	ERRTYP	
000154		LSSW==	LSSWLEN+2	
000140		LSHW==	LSHWLEN+2	
000011		LSDEPO==	LSREV+1	

000140'	DFPTBL==	LSHWLEN+2
000154'	SFPTBL==	LSSWLEN+2
000000'	FMT1==	P.AAA
000004'	FMT2==	P.AAB
000072'	FMT3==	P.AAC
000166'	FMT4==	P.AAD
000244'	UNITS.MSG==	P.AAE
000300'	EXISTS.MSG==	P.AAF
000344'	DNWNS.MSG==	P.AAG
000370'	INACC.MSG==	P.AAH
000450'	DFLTS.MSG==	P.AAI
000520'	SERIALS.MSG==	P.AAJ
000560'	DATES.MSG==	P.AAK
000616'	DEF.DATE==	P.AAL
000630'	DEF.SERIAL==	P.AAM
000010'	RINGBASE==	COM.AREA+10
000740'	MSGADR==	REC.BUF+2
002534'	PFE.STRUCT==	P.AAN
002610'	PWR.MSG==	P.ABK
002672'	ABO.MSG==	P.ABL
002752'	TO.MANY.UNITS==	P.ABM
003030'	GOOD.NUM.UNITS==	P.ABN
003122'	BOOT.FAILURE==	P.ABO
003204'	PROTO.VIOLATION==	P.ABP
003252'	PORT.INIT.ERR==	P.ABQ
003326'	HW.Q1.IP==	P.ABR
003356'	HW.Q2.VECTOR==	P.ABS
003410'	HW.Q3.BR==	P.ABT
003440'	SW.Q1.UNIT==	P.ABU
003534'	SW.Q2.MODE==	P.ABV
004220'	FMT.ERR==	P.ABW
006110'	EMSG.STRUCT==	P.ACE

PSECT SUMMARY

Psect Name	Words	Attributes			
\$CODE\$	60	RO , I ,	LCL,	REL,	CON
\$GLOB\$	429	RO , D ,	GBL,	REL,	CON
\$PLIT\$	1594	RO , D ,	GBL,	REL,	CON

LIBRARY STATISTICS

File	----- Total	Symbols Loaded	----- Percent	Blocks Read
DISK\$USER:[PRUCHA.RELEASE]ZRQBA0.L16;2	356	222	62	40

ZRQB1 RDRX DISK FORMATTER
REV A PATCH 00 GLOBAL TEXT SECTION

J 4
28-Jun-1983 13:01:32
28-Jun-1983 12:58:15

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB1.B16;2 (16)

SEQ 48
Page 48

COMMAND QUALIFIERS

:
: Size: 0 code + 2083 data words
: Run Time: 00:21.2
: Elapsed Time: 01:05.6
: Memory Used: 192 pages
: Compilation Complete

ZRQB2

RDRX DISK FORMATTER

28-Jun-1983 13:02:43
27-Jun-1983 18:05:59

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB2.B16;1 (1)

.....

```

0001 MODULE ZRQB2 (%TITLE 'RDRX DISK FORMATTER'
0002             IDENT = 'REV A PATCH 00',
0003             ADDRESSING MODE (ABSOLUTE) ,
0004             ENVIRONMENT (NOEIS)
0005             ) =
0006 BEGIN
0007
0008 !           CONTROLLER FUNCTIONS
0009
0010 library 'ZRQBA0.L16';
0011
0012 require 'BLSMAC.REQ';
1503
1504 %sbttl 'MODULE DECLARATIONS'

```

```

!Define RDRX Library module
!Define Bliss Macro Library

```

ZRQB2
REV

.....

0001
0002
0003
0004
0005
0006
0007
0008
0009
0010
0011
0012
0013
0014
0015
0016
0017
0018
0019
0020
0021
0022
0023
0024
0025
0026
0027
0028
0029
0030
0031
0032
0033
0034
0035
0036
0037
0038
0039
0040
0041
0042
0043
0044
0045
0046
0047
0048
0049
0050
0051
0052
0053
0054
0055
0056
0057
0058
0059
0060
0061
0062
0063
0064
0065
0066
0067
0068
0069
0070
0071
0072
0073
0074
0075
0076
0077
0078
0079
0080
0081
0082
0083
0084
0085
0086
0087
0088
0089
0090
0091
0092
0093
0094
0095
0096
0097
0098
0099
0100

ZRQB2
REV A PATCH 00

RDRX DISK FORMATTER
MODULE DECLARATIONS

L 4

28-Jun-1983 13:02:43
27-Jun-1983 18:05:59

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB2.B16;1 (2)

SEQ 50
Page 2

```
1505 !+
1506 | Structure declarations used within this module.
1507 | -
1508
1509 structure
1510
1511 !+
1512 | RDRX register accessing structure. This
1513 | structure allows RDRX register accessing
1514 | to be transportable between the PDP-11 and
1515 | VAX Diagnostic Supervisors.
1516
1517 | This also defines an access algorithm for
1518 | VAX to allow field reference to MBA address
1519 | space without generating machine checks.
1520 | -
1521
1522 RDRX [O, P, S, E] =
1523     begin
1524
1525         local
1526             RCSS_REG;
1527
1528         RCSS_REG = .(RDRX + %upval*0)<0, %bpval, 0>;
1529         RCSS_REG
1530     end
1531     <P, S, E>;
1532
```

ZRQB2
REV A PATCH 00RDRX DISK FORMATTER
MODULE DECLARATIONS28-Jun-1983 13:02:43
27-Jun-1983 18:05:59VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB2.B16;1 (3)SEQ 51
Page 3

```
:
: 1533 | +
: 1534 | The psect named "code or $code$" is redefined here
: 1535 | to be called "aa$code". This is done to force the tkb
: 1536 | linker to place the programs header information starting
: 1537 | at absolute address 2000. Then for consistency "a$code"
: 1538 | is used inplace of "code or $code" across all modules.
: 1539 | -
: 1540 | psect
: 1541 |     code = aa$code;
: 1542 |
: 1543 | +
: 1544 | External Routine declared outside this module.
: 1545 | -
: 1546 | external routine
: 1547 |     DECODE : NOVALUE,
: 1548 |     LOAD_FILE;
```

```
1549 !+
1550 ! External Declaration of datums
1551 ! declared outside of this module.
1552 !-
1553
1554 external
1555
1556     Hardware question ascii string messages
1557
1558     HW_Q1_IP,                !H/W question 1 for IP reg address
1559     HW_Q2_VECTOR,          !H/W question 2 for interrupt vector address
1560     HW_Q3_BR,              !H/W question 3 for bus req level
1561
1562     SW_Q1_UNIT,            !software question 1 for unit
1563     SW_Q2_MODE,            !software question 2 for unit
1564
1565     Formatting print string
1566
1567     FMT2,                  !Notifies unit being formatted
1568     FMT3,                  !Notifies format abort
1569
1570     Init code error and informational messages
1571
1572     PWR_MSG,
1573     ABO_MSG,
1574     TO_MANY_UNITS,
1575     GOOD_NUM_UNITS,
1576
1577     Miscellaneous external data declarations
1578
1579     LUN : byte,
1580     PTBL_PTR : ref vector [4, word],
1581
1582     Supervisor defined data declarations
1583
1584     LSUNIT,
1585
1586     Hardware P_Table storage declarations
1587
1588     RDRX_ADDR : ref RDRX field (ISD_FIELD),
1589     VEC_ADDR : word,
1590     BR_LEVEL : word,
1591     UNIT_NO : word,
1592
1593     Formatter data structures
1594
1595     ISD_STRUCT : blockvector [4, 2, word] field (ISD_FIELD);
1596
```

ZRQB2
REV A PATCH 00

RDRX DISK FORMATTER
TYPE AND DESCRIPTION

28-Jun-1983 13:02:43
27-Jun-1983 18:05:59

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB2.B16;1 (5)

```

: 1597 %sbttl 'TYPE AND DESCRIPTION'
: 1598 |*
: 1599 | Two lines of text will be printed to the operator (in addition to the
: 1600 | program name). The first will come from the 'DESCRIPT' macro at start
: 1601 | up time and will identify the diagnostics. The second will come from
: 1602 | the 'DEVTYPE' macro at hardware dialogue time and will identify the
: 1603 | device under test. The arguments of both macros are 72 character
: 1604 | ascii strings enclosed in parentheses:
: 1605 | -
: 1606 | DESCRIPT (%asciz'RD51 DISK FORMATTER');
: 1607 | DEVTYP (%asciz'RQDX1 DISK DRIVE SUBSYSTEM');
:

```

ZRQB2
REV A PATCH 00

RDRX DISK FORMATTER
HARDWARE PARAMETER CODING

28-Jun-1983 13:02:43
27-Jun-1983 18:05:59

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB2.B16;1 (6)

```

:      1608 %sbttl 'HARDWARE PARAMETER CODING'
:      1609 |
:      1610 | The hardware parameter coding section contains macros
:      1611 | that are used by the supervisor to build P-Tables. The
:      1612 | macros are not executed as machine instructions but are
:      1613 | interpreted by the supervisor as data structures. The
:      1614 | macros allow the supervisor to establish communications
:      1615 | with the operator.
:      1616 | -
:      1617 BGNHRD:
:      1618 GPRMA (HW_Q1_IP, %o'0', 0, %o'16000', %o'177777', YES, 1);
:      1619 GPRMA (HW_Q2_VECTOR, %o'2', 0, 4, %o'774', YES, 1);
:      1620 GPRMD (HW_Q3_BR, %o'4', 0, %o'177777', 0, 7, YES, 1);
:      1621 ENDHRD:

```

```

!Get RDRX Controller IP register
!Get RDRX Interrupt Vector address
!Get RDRX Bus Request Priority

```


ZRQB2
REV A PATCH 00

RDRX DISK FORMATTER
SOFTWARE PARAMETER CODING SECTION

28-Jun-1983 13:02:43
27-Jun-1983 18:05:59

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB2.B16;1 (7)

```

:      1622  %sbttl 'SOFTWARE PARAMETER CODING SECTION'
:      1623  |*
:      1624  | The software parameter coding section contains macros
:      1625  | that are used by the supervisor to build P-Tables. The
:      1626  | macros are not executed as machine instructions but are
:      1627  | interpreted by the supervisor as data structures. The
:      1628  | macros allow the supervisor to establish communications
:      1629  | with the operator.
:      1630  | -
:      1631  | BGNSFT;
:      1632  |
:      1633  | Software parameter coding is accomplished via DM
:      1634  | micro code. This is to conform to DJP protocol
:      1635  | standards.
:      1636  |
:      1637  | GPRMD (SW_Q1_UNIT, %o'0', 0, %o'177777', 0, 1, YES, 0);      !get unit number
:      1638  | GPRMD (SW_Q2_MODE, %o'2', 0, %o'177777', 1, 3, YES, 0);      !Get mode
:      1639  | ENDSFT;

```

ZRQB2
REV A PATCH 00

RDRX DISK FORMATTER
REPORT CODING SECTION

28-Jun-1983 13:02:43
27-Jun-1983 18:05:59

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB2.B16;1 (8)

```

: 1640 %sbttl 'REPORT CODING SECTION'
: 1641 |
: 1642 | The statistical report coding section contains the PRINTS macros that
: 1643 | will be used to generate statistical reports. The 'BGNRPT' and 'ENDRPT'
: 1644 | macros are used as begining and ending directives for the coding con-
: 1645 | tained in the section. However, an externally located DORPT call, or
: 1646 | a print command from the operator, may be used to request the execu-
: 1647 | tion of the report coding.
: 1648 | -
: 1649 BGNRPT;
: 1650 return;
: 1651 |:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
: 1652 |: THIS SECTION CONTAINS THE CODE FOR PRINTING
: 1653 |: STATISTICAL INFORMATION GATHERED BY THE DIAGNOSTIC. IT IS
: 1654 |: EXECUTED BY THE OPERATOR COMMAND 'PRINT' OR BY THE MACRO CALL
: 1655 |: 'DORPT'. USE THE PRINTS MACRO TO PRINT THE INFORMATION.
: 1656 |: USE FORMAT STATEMENTS AS IN THE PRINTB/PRINTX MACROS. IT IS
: 1657 |: THE PROGRAMMER'S RESPONSIBILTY TO DEVISE AND IMPLEMENT THE
: 1658 |: FORM AND CONTENT OF THE STATISTICS.
: 1659 |:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
: 1660 ENDRPT;

```

```

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

```

Address	Offset	Length	Value	Label	Details
000000					
000000	122	104	065	L\$DESC::	.ASCII /RD5/
000003	061	040	104		.ASCII /1 D/
000006	111	123	113		.ASCII /ISK/
000011	040	106	117		.ASCII /FO/
000014	122	115	101		.ASCII /RMA/
000017	124	124	105		.ASCII /TTE/
000022	122	000			.ASCII /R/<00>
000024					.BLKB 2
000026	122	121	104	L\$DVTYP::	
					.ASCII /RQD/
000031	130	061	040		.ASCII /X1 /
000034	104	111	123		.ASCII /DIS/
000037	113	040	104		.ASCII /K D/
000042	122	111	126		.ASCII /RIV/
000045	105	040	123		.ASCII /E S/
000050	125	102	123		.ASCII /UBS/
000053	131	123	124		.ASCII /YST/
000056	105	115	000		.ASCII /EM/<00>
000061	000				.ASCII <00>
000062	000000C			L\$HRDLN::	
					.WORD <<<L\$NDHRD-L\$HRDLN>/2>-1>
000064	000031			GPS1::	.WORD 31
000066	000000G				.WORD HW.Q1.IP
000070	016000				.WORD 16000

000072 177777
000074 001031
000076 000000G
000100 000004
000102 000774
000104 002032
000106 000000G
000110 177777
000112 000000
000114 000007
000116

GPS2:: .WORD -1
.WORD 1031
.WORD HW.Q2.VECTOR
.WORD 4
GPS3:: .WORD 774
.WORD 2032
.WORD HW.Q3.BR
.WORD -1
.WORD 0
.WORD 7

LSNDHRD:: .BLKW 1

LSSFTLN:: .WORD <<<LSNDSFT-LSSFTLN>/2>-1>
GPS4:: .WORD 32
.WORD SW.Q1.UNIT
.WORD -1
.WORD 0

GPS5:: .WORD 1
.WORD 1032
.WORD SW.Q2.MODE
.WORD -1
.WORD 1
.WORD 3

LSNDSFT:: .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 LSUNIT, RDRX.ADDR, VEC.ADDR, BR.LEVEL
.GLOBL UNIT.NO, ISD.STRUCT

000064'
000122'

LSHARD== LSHRDLN+2
LSSOFT== LSSFTLN+2

000000 000207

LRPT: .SBTTL LRPT REPORT CODING SECTION ; 1639
RTS PC

; Routine Size: 1 word, Routine Base: \$CODE\$ + 0150
; Maximum stack depth per invocation: 0 words

000000 004737 000150'
000004 104425
000006 000207

LSRPT:: .SBTTL LSRPT REPORT CODING SECTION ; 1650
JSR PC,LRPT
TRAP 25
RTS PC

ZRQB2 RDRX DISK FORMATTER
REV A PATCH 00 REPORT CODING SECTION

G 5

28-Jun-1983 13:02:43
27-Jun-1983 18:05:59

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB2.B16;1 (8)

SEQ 58
Page 10

: Routine Size: 4 words, Routine Base: \$CODES + 0152
: Maximum stack depth per invocation: 2 words

1661 %sbttl 'INITIALIZE SECTION'
1662 BGNINIT;

1663
1664 !++
1665 | The initialization code is executed at the beginning of every
1666 | sub-pass and is primarily used for requesting P Tables. Any
1667 | other set-up type functions may also be performed in the init
1668 | code.

1669
1670 | The initialize code is executed under five conditions. There
1671 | are supervisor event flags that are used to let the diagnostic
1672 | know under which condition the execution is taking place. The
1673 | event flags are read using the 'READEF' macro.

1674
1675 | The conditions under which the init code is executed and the
1676 | corresponding event flags are:

1677		
1678	START COMMAND	EF.START
1679	RESTART COMMAND	EF.RESTART
1680	CONTINUE COMMAND	EF.CONTINUE
1681	POWERDOWN/POWERUP	EF.PWR
1682	NEW PASS	EF.NEW

1683
1684 | Example of event flag use:

1685
1686 | if READEF(EF.START) then
1687 | START_FLAG = 1;

1688 |--
1689
1690 |+
1691 | First read the event flag EF_PWR to see
1692 | if this init code is being performed due
1693 | to a system power fail. If it is then
1694 | report the incident to the operator and
1695 | abort the DM machine and further execution
1696 | of this program.

1697 |--
1698
1699 | if READEF (EF_PWR)
1700 | then

1701 | begin
1702 | PRINTF (PWR_MSG);
1703 | PRINTF (ABO_MSG);
1704 | WRT_RDRX (RCIP, RC_ALL, ZEROS);
1705 | DOCLN;
1706 | end;

!Is the PWR event flag set

!Report the incident and abort
!Power fail print message
!Aborting program message
!Abort DM code execution
!Abort further program execution

1707
1708 |
1709 | See if the DRS commands START or RESTART were used to start
1710 | this formatting session. If either of them were used then
1711 | start the formatting session at logical unit zero.
1712 |
1713

ZRQB2
REV A PATCH 00

RDRX DISK FORMATTER
INITIALIZE SECTION

I 5

28-Jun-1983 13:02:43
27-Jun-1983 18:05:59

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB2.B16;1 (9)

SEQ 60
Page 12

```
1714 if (READEF (EF_START)) or (READEF (EF_RESTART)) !Did start of restart get us here
1715 then
1716     begin
1717     !
1718     ! Check the operator for trying to format
1719     ! more than the defined limit. If they
1720     ! are then report the error and die.
1721     !
1722     !
1723     if .LSUNIT gtru LSLIMIT !Is the formatting limit exceeded
1724     then
1725     begin
1726     PRINTF (TO_MANY_UNITS); !Report the error
1727     PRINTF (GOOD_NUM_UNITS); !Tell him/her the limit
1728     DOCLN; !Go to cleanup and die
1729     end;
1730     !
1731     !+
1732     ! Everything looks good so far so lets
1733     ! move on and get the hardware question
1734     ! responses and save them.
1735     !-
1736     !
1737     LUN = -1; !Start formatting at logical unit 0
1738     !
1739     do
1740     begin
1741     LUN = .LUN + 1;
1742     !
1743     if .LUN gequ .LSUNIT then DOCLN;
1744     !
1745     end
1746     until (GPHARD (.LUN, PTBL_PTR)) nequ ZERO;
1747     !
1748     RDRX_ADDR = .PTBL_PTR [wrđ0]; !Load up the controllers base address
1749     VEC_ADDR = .PTBL_PTR [wrđ1]; !Load up the controllers vector address
1750     BR_LEVEL = .PTBL_PTR [wrđ2]; !Load up the controllers bus request
1751     UNIT_NO = .PTBL_PTR [wrđ3]; !Load up the unit number to format
1752     !+
1753     ! Before leaving the init code section we must
1754     ! first do some house keeping left behind from
1755     ! the ISD_STRUCT preset declaration. The adapter
1756     ! purge interrupt bit must be defined for the
1757     ! type machine this formatter is running under.
1758     !-
1759     !
1760     if %bliss (bliss16) !Define compiler
1761     then
1762     ISD_STRUCT [BLK1, WRD1, S2W_PI] = ZERO !No purging for PDP-11
1763     else
1764     ISD_STRUCT [BLK1, WRD1, S2W_PI] = ONE; !Purging for VAX-11
1765     !
1766     !+
```

```

1767 : Now load in the RDRX formatter DM code from
1768 : the local boot device into the
1769 : blank buffer allocated in azkel6. Report
1770 : load error if an error code is returned.
1771
1772 : The DM code will only be read in during start
1773 : or restarts of the host code. Block zero of the
1774 : the DM .sav file will thrown out.
1775 :
1776 :
1777 : if LOAD_FILE (AZFMTR, DM_FNSEXT, DM_SIZE) then DECODE ();
1778
1779 :
1780 : Now calculate the overlay sections starting address
1781 : and store the result in global location 'ovsa' for
1782 : future ref.
1783
1784 : This takes the DM buffer starting adrs and adds to it
1785 : the number of bytes in the (initial load + remote prog
1786 : header size) resulting in the first adrs of the overlay
1787 : section.
1788 :
1789 :
1790 : OVSA = AZFMTR + (.AZFMTR [WRD0]);           !Calculate overlay start adrs
1791
1792 end
1793 else
1794 begin
1795 if READEF (EF_CONTINUE)
1796 then
1797 begin
1798 : End this Host code if this .lun is the
1799 : the last logical unit to format.
1800 :
1801 :
1802 :
1803 if (.LUN + 1) gequ .LSUNIT then DOCLN;
1804 PRINTF (FMT3, .LUN, .UNIT_NO);           !Report this luns format was aborted
1805 end;
1806
1807 if READEF (EF_NEW) then GPHARD (0, 0);     !New pass means all units formatted
1808
1809 do
1810 begin
1811 LUN = .LUN + 1;
1812 if .LUN gequ .LSUNIT then DOCLN;
1813 end
1814 until (GPHARD (.LUN, PTBL_PTR)) nequ ZERO;
1815
1816
1817
1818
1819

```

ZRQB2
REV A PATCH 00

RDRX DISK FORMATTER
INITIALIZE SECTION

K 5
28-Jun-1983 13:02:43
27-Jun-1983 18:05:59

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB2.B16;1 (9)
SEQ 62
Page 14

1820
1821 RDRX_ADDR = .PTBL_PTR [wrd0];
1822 VEC_ADDR = .PTBL_PTR [wrd1];
1823 BR_LEVEL = .PTBL_PTR [wrd2];
1824 UNIT_NO = .PTBL_PTR [wrd3];
1825 end;
1826 !
1827 ENDINIT;

!Load up the controllers base address
!Load up the controllers vector address
!Load up the controllers bus request
!Load up the unit number to format

Address	Offset	Label	Instruction	Comment	Address
000000	005746		LINIT: TST	-(SP)	1660
000002	012700	000034	MOV	#34,R0	1699
000006	104447		TRAP	47	
000010	103025		BHIS	1\$	
000012	012746	000000G	MOV	#PWR.MSG,-(SP)	1702
000016	012746	000001	MOV	#1,-(SP)	
000022	010600		MOV	SP,R0	: SP,*
000024	104417		TRAP	17	
000026	012716	000000G	MOV	#ABO.MSG,(SP)	1703
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)	: *,RCSS.REG : RC\$M.REG 1704
000050	005000		CLR	R0	
000052	005077	000000G	CLR	@RDRX.ADDR	
000056	104444		TRAP	44	
000060	062706	000006	ADD	#6,SP	1701
000064	012700	000040	MOV	#40,R0	1714
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	CMP	LSUNIT,#20	: 1723
000112	101417		BLOS	3\$	
000114	012746	000000G	MOV	#TO.MANY.UNITS,-(SP)	: 1726
000120	012746	000001	MOV	#1,-(SP)	
000124	010600		MOV	SP,R0	: SP,*
000126	104417		TRAP	17	
000130	012716	000000G	MOV	#GOOD.NUM.UNITS,(SP)	: 1727
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	: 1725
000152	112737	000377 000000G	MOVB	#377,LUN	: 1737
000160	105237	000000G	INCB	LUN	: 1741
000164	005000		CLR	R0	: 1743
000166	153700	000000G	BISB	LUN,R0	
000172	020037	000000G	CMP	R0,LSUNIT	
000176	103401		BLO	5\$	
000200	104444		TRAP	44	

ZRQB2
REV A PATCH 00

RDRX DISK FORMATTER
INITIALIZE SECTION

28-Jun-1983 13:02:43
27-Jun-1983 18:05:59

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB2.B16;1 (9)

000202	005000		5\$:	CLR	R0	:	1746
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	:	1748
000224	016037	000002 000000G		MOV	2(R0),VEC.ADDR	:	1749
000232	016037	000004 000000G		MOV	4(R0),BR.LEVEL	:	1750
000240	016037	000006 000000G		MOV	6(R0),UNIT.NO	:	1751
000246	142737	000001 000006G		BICB	#1,ISD.STRUCT+6	:	1760
000254	000474			BR	11\$:	1714
000256	012700	000036	6\$:	MOV	#36,R0	:	1796
000262	104447			TRAP	47	:	
000264	103025			BHIS	8\$:	
000266	005000			CLR	R0	:	1804
000270	153700	000000G		BISB	LUN,R0	:	
000274	005200			INC	R0	:	
000276	020037	000000G		CMP	R0,LSUNIT	:	
000302	103401			BLO	7\$:	
000304	104444			TRAP	44	:	
000306	013746	000000G	7\$:	MOV	UNIT.NO,-(SP)	:	1806
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	:	1798
000340	012700	000035	8\$:	MOV	#35,R0	:	1810
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	:	1814
000364	005000			CLR	R0	:	1816
000366	153700	000000G		BISB	LUN,R0	:	
000372	020037	000000G		CMP	R0,LSUNIT	:	
000376	103401			BLO	10\$:	
000400	104444			TRAP	44	:	
000402	005000		10\$:	CLR	R0	:	1819
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	:	1821
000424	016037	000002 000000G		MOV	2(R0),VEC.ADDR	:	1822
000432	016037	000004 000000G		MOV	4(R0),BR.LEVEL	:	1823
000440	016037	000006 000000G		MOV	6(R0),UNIT.NO	:	1824
000446	005726		11\$:	TST	(SP)+	:	1660
000450	000207			RTS	PC	:	

; Routine Size: 149 words, Routine Base: \$CODE\$ + 0162

ZRQB2 RDRX DISK FORMATTER
REV A PATCH 00 INITIALIZE SECTION

M 5
28-Jun-1983 13:02:43
27-Jun-1983 18:05:59

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB2.B16;1 (9)
SEQ 64
Page 16

: Maximum stack depth per invocation: 7 words

000000	004737	000162'							
000004	104411		LSINIT::JSR	.SBTTL	LSINIT INITIALIZE SECTION				
000006	000207		TRAP		PC,LINIT	:			1825
			RTS		PC				

: Routine Size: 4 words, Routine Base: \$CODES + 0634
: Maximum stack depth per invocation: 2 words

28-Jun-1983 13:02:43
27-Jun-1983 18:05:59

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB2.B16;1 (10)

ZRQB2 RDRX DISK FORMATTER
REV A PATCH 00 AUTODROP SECTION

```

:      1828 %sbttl 'AUTODROP SECTION'
:      1829 |
:      1830 | This code is executed immediately after the initialize code if
:      1831 | the 'ADR' flag was set. The unit(s) under test are checked to
:      1832 | see if they will respond. Those that don't are immediately
:      1833 | dropped from testing.
:      1834 | -
:      1835 | BGNAUTO;
:      1836 |
:      1837 | Testing devices for existence
:      1838 | is done via test 1 of this program.
:      1839 |
:      1840 | return;
:      1841 | ENDAUTO;

```

```

000000 000207          LAUTO: .SBTTL LAUTO AUTODROP SECTION          ;          1827
                                RTS    PC

```

```

; Routine Size: 1 word,          Routine Base: $CODE$ + 0644
; Maximum stack depth per invocation: 0 words

```

```

000000 004737 000644'    L$AUTO::JSR L$AUTO AUTODROP SECTION          ;          1840
000004 104461            TRAP    PC,LAUTO
000006 000207            RTS    61
                                PC

```

```

; Routine Size: 4 words,          Routine Base: $CODE$ + 0646
; Maximum stack depth per invocation: 2 words

```

ZRQB2
REV A PATCH 00

RDRX DISK FORMATTER
CLEANUP CODING SECTION

28-Jun-1983 13:02:43
27-Jun-1983 18:05:59

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB2.B16;1 (11)

```

:      1842 %sbttl 'CLEANUP CODING SECTION'
:      1843 |
:      1844 | Cleanup coding is assembled with the diagnostic program, utilizing in-
:      1845 | itiating (BGNCLN) and ending (ENDCLN) directives. The coding can be
:      1846 | used by either the diagnostic program or the supervisor to affect the
:      1847 | return of a test device to a static state.
:      1848 |
:      1849 | The clean-up code is invoked in three different ways:
:      1850 |
:      1851 |     A. At end of every sub-pass
:      1852 |     B. Issuance of DOCLN macro
:      1853 |     C. Operator ^C
:      1854 | -
:      1855 BGNCLN;
:      1856 BRESET;
:      1857 return;
:      1858 ENDCLN;

```

```

000000 104433          .SBTTL  LCLEAN CLEANUP CODING SECTION          1855
000002 000207          LCLEAN: TRAP 33                               1841
                                RTS  PC                               :

```

```

; Routine Size: 2 words,      Routine Base: $CODE$ + 0656
; Maximum stack depth per invocation: 2 words

```

```

000000 004737 000656'          .SBTTL  L$CLEAN CLEANUP CODING SECTION          1857
                                L$CLEAN::
                                JSR  PC,L$CLEAN                       :
000004 104412          TRAP 12
000006 000207          RTS  PC

```

```

; Routine Size: 4 words,      Routine Base: $CODE$ + 0662
; Maximum stack depth per invocation: 2 words

```

ZRQB2
REV A PATCH 00

RDRX DISK FORMATTER
DROP UNIT SECTION

28-Jun-1983 13:02:43
27-Jun-1983 18:05:59

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB2.B16;1 (12)

```

:      1859 %sbttl 'DROP UNIT SECTION'
:      1860 !+
:      1861 | The drop code is invoked by a DODU macro or a drop command, and con-
:      1862 | tains any code that needs to be executed in conjunction with the drop-
:      1863 | ping of a unit from the test cycle. No coding is required in this
:      1864 | section.
:      1865 |
:      1866 | The effect of a DODU is the same whether executed in the init code or
:      1867 | in a hardware test. It invokes the drop unit coding and causes subse-
:      1868 | quent GPHARD'S for that logical unit to be returned 'NOT COMPLETE'.
:      1869 | This effect lasts only for the duration of the current command.
:      1870 | -
:      1871 BGNDU;
:      1872 return;
:      1873 ENDDU;

```

```

000000 000207          LDU:  .SBTTL LDU DROP UNIT SECTION          1858
                        RTS    PC
; Routine Size: 1 word,      Routine Base: $CODE$ + 0672
; Maximum stack depth per invocation: 0 words

```

```

000000 004737 000672' LSDU:: .SBTTL LSDU DROP UNIT SECTION          1872
000004 104453          JSR    PC,LDU
000006 000207          TRAP  53
                        RTS    PC
; Routine Size: 4 words,      Routine Base: $CODE$ + 0674
; Maximum stack depth per invocation: 2 words

```

ZRQB2
REV A PATCH 00

RDRX DISK FORMATTER
ADD UNIT SECTION

28-Jun-1983 13:02:43
27-Jun-1983 18:05:59

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB2.B16;1 (13)

```

:      1874  %sbttl 'ADD UNIT SECTION'
:      1875  +
:      1876  | The add code is invoked by the ADD command, and contains any code that
:      1877  | needs to be executed in conjunction with adding a unit back to the
:      1878  | test cycle. No coding is required in this section.
:      1879  |
:      1880  | Units may be added to the test sequence only through the use of opera-
:      1881  | tor ADD command. Each unit must have a P-TABLE in memory due to an
:      1882  | earlier hardware dialogue (i.e. the unit was previously dropped).
:      1883  | The ADD code must be delimited by BGNAU, ENDAU. There is no particu-
:      1884  | lar coding required in the add code to cause the add to be effective:
:      1885  |
:      1886  | The section is just for programmer housekeeping.
:      1887  |
:      1888  | BGNAU;
:      1889  | return;
:      1890  | ENDAU;

```

```

000000 000207          LAU:  .SBTTL  LAU ADD UNIT SECTION          1873
                        RTS    PC
; Routine Size: 1 word,      Routine Base: $CODE$ + 0704
; Maximum stack depth per invocation: 0 words

```

```

000000 004737 000704'  LSAU:: .SBTTL  LSAU ADD UNIT SECTION          1889
000004 104452          JSR    PC,LAU
000006 000207          TRAP  52
                        RTS    PC
; Routine Size: 4 words,      Routine Base: $CODE$ + 0706
; Maximum stack depth per invocation: 2 words

```

```

:      1891  end
:      1892
:      1893  eludom

```

PSECT SUMMARY

```

:      Psect Name          Words  Attributes
:      $CODE$             231     RO , I , LCL, REL, CON

```

LIBRARY STATISTICS

ZRQB2 RDRX DISK FORMATTER
REV A PATCH 00 ADD UNIT SECTION

E 6

28-Jun-1983 13:02:43
27-Jun-1983 18:05:59

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB2.B16;1 (13)

SEQ 69
Page 21

File	----- Total	Symbols Loaded	----- Percent	Blocks Read
DISK\$USER:[PRUCHA.RELEASE]ZRQBA0.L16;2	356	103	28	20

COMMAND QUALIFIERS

: Size: 179 code + 52 data words
: Run Time: 00:15.5
: Elapsed Time: 00:59.3
: Memory Used: 190 pages
: Compilation Complete

ZRQB3

RDRX DISK FORMATTER

F 6

28-Jun-1983 13:03:45
27-Jun-1983 18:06:20

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB3.B16;1 (1)

SEQ 70
Page 1

```
.....  
: 0001 MODULE ZRQB3 (XTITLE 'RDRX DISK FORMATTER'  
0002 IDENT = 'REV A PATCH 00'  
0003 ADDRESSING MODE (ABSOLUTE) ,  
0004 ENVIRONMENT (NOEIS)  
0005 ) =  
0006 BEGIN  
0007  
0008 ! MAIN CODE FOR FORMATTER  
0009  
0010 library 'ZRQBA0.L16';  
0011 require 'BLSMAC.REQ';  
1503  
1504 %sbttl 'MODULE DECLARATIONS'
```

```
!Define RDRX formatter library  
!Define Bliss Macro require file
```


ZRQB3
REV A PATCH 00

RDRX DISK FORMATTER
MODULE DECLARATIONS

6 6

28-Jun-1983 13:03:45
27-Jun-1983 18:06:20

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB3.B16;1 (2)

SEQ 71
Page 2

```
1505 !+
1506 ! The psect named "code or $code$" is redefined here
1507 ! to be called "ab$code". This is done to organize the
1508 ! formatters test sections into a seperate psect.
1509 !-
1510 ! psect
1511 !   code = ab$code;
1512 !
1513 !+
1514 ! Structure declarations used within this module.
1515 !-
1516 !
1517 ! structure
1518 !
1519 !   !+
1520 !   ! RDRX register accessing structure. This
1521 !   ! structure allows RDRX register accessing
1522 !   ! to be transportable between the PDP-11 and
1523 !   ! VAX Diagnostic Supervisors.
1524 !   !
1525 !   ! This also defines an access algorithm for
1526 !   ! VAX to allow field reference to MBA address
1527 !   ! space without generating machine checks.
1528 !   !-
1529 !
1530 ! RDRX [O, P, S, E] =
1531 !   begin
1532 !
1533 !     local
1534 !       RCSS_REG;
1535 !
1536 !       RCSS_REG = .(RDRX + %upval*0)<0, %bpval, 0>;
1537 !       RCSS_REG
1538 !     end
1539 !     <P, S, E>;
1540 !
```

ZRQB3
REV A PATCH 00

RDRX DISK FORMATTER
MODULE DECLARATIONS

H 6

28-Jun-1983 13:03:45
27-Jun-1983 18:06:20

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB3.B16;1 (3)

SEQ 72
Page 3

```
1541 !+
1542 ! External Routines declared outside this module.
1543 !-
1544
1545 external routine
1546     copy,
1547     DATE_CHECK,
1548     ABORT,
1549     GET_DUST_STATUS,
1550     EX_LOC_PROG,
1551     REC_DATA,
1552     SEND_DATA,
1553     SET_CNTRL_CHAR,
1554     DUP$SERVICE : INT_LNKSTYP novalue,
1555     INT$SERVICE : INT_LNKSTYP novalue,
1556     INIT_COM_AREA,
1557     BOOT_RDRX,
1558     DECODE : novalue;
1559
```

```
1560 !+
1561 ! External Declaration of datums
1562 ! declared outside of this module.
1563 !-
1564
1565 external
1566
1567 ! Micellanious external data declarations
1568
1569 NXT_CRN : byte, !Next seq command ref number
1570 RET_EN$AD : ref block [RB_SIZE + 2, word] field (ENV_FIELD),
1571 SND_BUF : vector [SNDB_SIZE, word], !DUP send cmd text buffer
1572 REC_BUF : block [RECB_SIZE, word] field (RECB_FIELD), !DUP receive cmd text buffer
1573 MSGADR, !Pointer to DM sent ascii text
1574 NSD_SLOT : word, !Stores next send ring slot to load cmd into
1575 NRD_SLOT : word, !Stores next receive slot to expect response in
1576 VEC_ADDR : word, !Stores controllers vector address
1577 RET_STATUS : word, !Stores return status of called routines
1578 PID_SAVE : word, !Saves process indicator word
1579 RDRX_ADDR : ref RDRX field (ISD_FIELD), !RDRX reference structure
1580 PTBL_PTR : ref vector [4, word], !table pointer for fetching unit number
1581 SW_UNIT_NO : word, !software table unit number
1582 SW_MODE : word, !software table mode
1583
1584 ! Init sequence code error and informational messages
1585
1586 BOOT_FAILURE,
1587 PROT0_VIOLATION,
1588 PORT_INIT_ERR,
1589
1590 ! Default values
1591
1592 DEF_SERIAL,
1593 DEF_DATE,
1594
1595 ! Printing format strings
1596
1597 FMT1,
1598 FMT4,
1599 FMT_ERR : vector [7],
1600
1601
1602 !Special questions so the formatter can save code.
1603
1604 UNITS_MSG,
1605 EXISTS_MSG,
1606 DOWNS_MSG,
1607 INACCS_MSG,
1608 DFLTS_MSG,
1609 SERIALS_MSG,
1610 DATES_MSG;
1611
```

```
1612 Xsbttl 'FORMATTER SECTION 1'
1613 BGNTST;
1614
1615 !++
1616 | Functional Description :
1617 | Implicit Inputs :
1618 | Implicit Outputs :
1619 | Completion Codes :
1620 | Side Effects :
1621 |--
1622 LABEL TLOOP;
1623 local
1624 RETRIES;                                !Store number of retries to perform
1625
1626 !+
1627 | The next thing to be done is to boot the
1628 | RDRX controller. We will allow a few
1629 | retries if not successful after the first
1630 | boot before we considered the Controller
1631 | dead.
1632
1633 | But before we do the boot sequence the processors
1634 | priority and interrupt vector address must first
1635 | be loaded. During the init sequence the Init
1636 | sequence interrupt service routine will just flag
1637 | any interrupts and ignore them since interrupts are
1638 | disabled during init sequence. Later the DUP interrupt
1639 | service routine will be load and do the DUP communications
1640 | protocol.
1641
1642 | The following priorities will be assigned:
1643 | 1. Processor will run at priority zero.
1644 | 2. The RDRX runs at priority 5 by default.
1645 | 3. The DUP interrupt routine will run at priority 7.
1646 |--
1647 SETPRI (PRI00);                          !Set the processors priority
1648 CLRVEC (.VEC_ADDR);                       !Clear out the vector before setting
1649 SETVEC (.VEC_ADDR, INT$I_SERVICE, PRI07); !Set the interrupt service priority
1650
1651 !+
1652 | Retry the RDRX booting until the
1653 | return is true or the retry limit
1654 | is reached.
1655 |--
1656 RETRIES = -1;                              !Reset the retry counter
1657
1658 do
1659     begin
1660         RET_STATUS = BOOT_RDRX ();          !Boot the RDRX controller
1661         RETRIES = .RETRIES + 1;           !Up the retry count
1662     end
1663 until (.RET_STATUS) or (.RETRIES eql ONE); !Repeat the Boot until done
1664
```

ZRQB3
REV A PATCH 00

RDRX DISK FORMATTER
FORMATTER SECTION 1

K 6

28-Jun-1983 13:03:45
27-Jun-1983 18:06:20

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB3.B16;1 (5)

SEQ 75
Page 6

```
1665 |
1666 | Report booting error if the return
1667 | status never came back as true.
1668 |
1669 | if not .RET_STATUS          !Did the Controller boot
1670 | then
1671 |     begin                  !Report a boot error
1672 |     PRINTF (BOOT_FAILURE);
1673 |     DORPT;
1674 |     DOCLN;
1675 |     end;                  !Report some statistical data
1676 |
1677 |
1678 | !+ Now that the RDRX controller is
1679 | !+ booted, check to make sure that the
1680 | !+ controller has done its part of the
1681 | !+ DUP protocol by clearing out the
1682 | !+ port communications area.
1683 |
1684 | !+ While we're there set up the communication
1685 | !+ area for the up and coming communications
1686 | !+ between the remote and host program.
1687 | !+
1688 |
1689 | if INIT_COM_AREA ()      !Was the com area cleared out as expected
1690 | then
1691 |     begin
1692 |     PRINTF (PROTO_VIOLATION);
1693 |     PRINTF (PORT_INIT_ERR);
1694 |     DOCLN;
1695 |     end;
1696 |
1697 |
1698 | !+ Before writing the go to the IP register
1699 | !+ load the DUP interrupt service adrs into
1700 | !+ the RDRX vector address for the up and
1701 | !+ coming communications between the host
1702 | !+ and controller.
1703 |
1704 | CLRVEC (.VEC_ADDR);      !Clear out the vector before starting
1705 | SETVEC (.VEC_ADDR, DUP$I_SERVICE, PRI07); !Set the interrupt service priority
1706 | !+
1707 | !+ All systems are go. Start the controllers
1708 | !+ functional micro code off and running. Write
1709 | !+ to the SA Register with the go bit set and start
1710 | !+ things rolling.
1711 | !+
1712 | WRT_RDRX (RCSA, RC_ALL, ONE); !Let the controller go
1713 |
```

ZRQB3
REV A PATCH 00

RDRX DISK FORMATTER
FORMATTER SECTION 1

```

: 1714 !%sbttl 'FORMATTER SECTION 2'
: 1715 !
: 1716 !
: 1717 !++
: 1718 ! Functional Description :
: 1719 ! Implicit Inputs :
: 1720 ! Implicit Outputs :
: 1721 ! Completion Codes :
: 1722 ! Side Effects :
: 1723 !--
: 1724 !
: 1725 !+
: 1726 ! Set the controller characteristics. Default
: 1727 ! values will be taken in all cases except for
: 1728 ! the host time out value which will be changed
: 1729 ! to 'wait for ever'.
: 1730 !-
: 1731 !
: 1732 ! if SET_CNTLCHAR ( ) then DECODE ( );
: 1733 !
: 1734 !+
: 1735 ! Do a unit on line cmd which will spin up the
: 1736 ! device and load the heads. The unit to be
: 1737 ! placed on line will be the last unit number
: 1738 ! received from the 'gphard' macro in the init
: 1739 ! code.
: 1740 !-
: 1741 !

```

!Call decode if not successful

```
1742 !%sbttl 'FORMATTER SECTION 3'
1743
1744 ! See if the Dup server in the RDRX Controller is in an
1745 ! idle state. To do this first get the dust status and
1746 ! then look at the flag field bit 3 for a :
1747 !     0 = idle
1748 !     1 = active
1749 !
1750
1751 if GET_DUST_STATUS () then DECODE ();           !Call Decode if connection error
1752
1753 !
1754 ! Look in the flag field bit 3 to see if the server is active.
1755 !
1756
1757 if .RET_EN$AD [FLG_B3]                          !If the server is active exit and reboot
1758 then
1759     begin
1760     !*****
1761     ! Do some stat recording
1762     !*****
1763     DOCLN;                                       !Exit this passes execution
1764     end;
1765
1766 !
1767 ! The server is not active so down line load the formatter
1768 ! and start its execution by issuing a "Execute local program".
1769 ! Call the decode routine if a connection error is detected.
1770 !
1771
1772 if EX_LOC_PROG () then DECODE ();               !Call decode if connection error
1773
1774 !
1775 ! Get the dust status to see if the server is in an active
1776 ! state. An active state is what we want so error if the
1777 ! server is in an idle state.
1778 !
1779 ! If in the active state then save the progress indicator
1780 ! in 'Pid_save' for future reference.
1781 !
1782
1783 if GET_DUST_STATUS () then DECODE ();           !Call decode if connection error
1784
1785 !
1786 ! Look at the flag field bit 3 to see if the server is active.
1787 !
1788
1789 if not (.RET_EN$AD [FLG_B3])                    !Reboot if server is idle
1790 then
1791     begin
1792     !*****
1793     ! Do some stat recording
1794     !*****
```

```
1795 DOCLN; !Exit this passes execution
1796 end
1797 else
1798 PID_SAVE = .RET_EN$AD [PLO_IND]; !Save the progress indicator
1799
1800 !+
1801 | The Dup server is in the active state running the formatter
1802 | program. This DO LOOP will loop on the DUP sub-protocol
1803 | doing the "send and receive" data commands. These commands
1804 | establish the communications between this host program and
1805 | the remote formatter program running in the RDRX controller.
1806 | -
1807 while TRUE do
1808 begin
1809
1810 !+
1811 | Do a 'Receive_data' command which poll's the remote program
1812 | for a message. The returned message can either be a:
1813 |
1814 | 1. Question
1815 | Where the ascii text is a prompt for information.
1816 |
1817 | 2. Default question
1818 | Where the default question message is identical
1819 | to the question message except that a null (zero
1820 | length) send data is taken to be a default answer
1821 | to the question.
1822 |
1823 | 3. Information
1824 | Where the ascii text is an informative message.
1825 |
1826 | 4. Termination
1827 | Where the ascii text is an normal termination message.
1828 |
1829 | 5. Fatal Error
1830 | Where the ascii text is a fatal error message.
1831 |
1832 | 6. Special
1833 | This type is used when only a host program could
1834 | respond.
1835 | -
1836
1837 if REC_DATA () then DECODE (); !Call decode if connection error
1838
1839 |
1840 | From the first word in the send/receive data buffer, look
1841 | to see what type message the remote program has sent to
1842 | the Host program. Use this message type number to index
1843 | into the select expression to perform the requested action
1844 | by the remote program.
1845
1846 CLR_SBUF; !Clear out the send buffer area
1847
```



```

1848 selectoneu .REC_BUF [MSG_TYP] of          !Select the appropriate action
1849 set
1850
1851 [1] :                                       !Question message type
1852 begin
1853
1854     Look into the send/receive data buffer at the message
1855     number field and see what question the remote program
1856     is asking. Use the fields value to index into the
1857     select expression to perform the appropriate action.
1858
1859
1860 selectoneu .REC_BUF [MSG_NUM] of          !Select the requested question
1861 set
1862
1863 [0] :                                       !Enter unit number to format
1864 begin
1865     if (MANUAL) then GMANID (UNITS_MSG, SND_BUF, D, %o'3', 0, 3, NO)
1866     else SND_BUF = .SW_UNIT_NO;
1867     if SEND_DATA () then DECODE ();
1868     end;
1869
1870 [1] :                                       !Choose one of the responses
1871 begin
1872     if (MANUAL) then
1873     begin
1874         SND_BUF = 0;
1875         GMANIL (EXISTS_MSG, SND_BUF, %o'1', YES, 1);
1876         if ( .SND_BUF<0, 1, 0> NEQU 1)
1877         then begin
1878             GMANIL (DOWNS_MSG, SND_BUF, %o'2', YES, 1);
1879             if ( .SND_BUF<1, 1, 0> NEQU 1)
1880             then begin
1881                 GMANIL (INACCS_MSG, SND_BUF, %o'4', YES, 1);
1882                 if ( .SND_BUF<2, 1, 0> NEQU 1)
1883                 then begin
1884                     SND_BUF<0, 1, 0> = 1;
1885                     PRINTF(DFLT$MSG);
1886                 end;
1887             end;
1888         end;
1889     end
1890     else SND_BUF = .SW_MODE;
1891
1892     if SEND_DATA () then DECODE ();
1893     end;
1894
1895 [2] :                                       !Enter an eight character non-zero ascii serial number
1896 begin                                       !(Any eight characters will do)
1897     if (MANUAL) then GMANID (SERIAL$MSG, SND_BUF, A, %o'177777', 8, 8, NO)
1898     else copy (DEF_SERIAL, SND_BUF, 8);
1899     if SEND_DATA () then DECODE ();
1900     end;

```

```
1901  
1902 [3] : !Enter current date MM-DD-YY <- exact format required  
1903 begin !(Actually, any six characters will do)  
1904 ! while DATE CHECK (SND_BUF)  
1905 if (MANUAL) then GMANID (DATE$ MSG, SND_BUF, A, %o'177777', 8, 8, NO)  
1906 else copy (DEF_DATE, SND_BUF, 8);  
1907 if SEND_DATA () then DECODE ();  
1908 end;  
1909  
1910 [otherwise] : !This message number is unknown  
1911 begin  
1912 RET_STATUS = UMN_CODE; !Unknown message number error code  
1913 DECODE (); !Report error and die  
1914 end;  
1915 tes;  
1916  
1917 end;  
1918  
1919 [2] : !Default Question  
1920 begin  
1921  
1922 selectoneu .REC_BUF [MSG_NUM] of  
1923 set  
1924 [otherwise] : !This message number is unknown  
1925 begin  
1926 RET_STATUS = UMN_CODE; !Unknown message number error code  
1927 DECODE (); !Report error and die  
1928 end;  
1929 tes;  
1930  
1931 end;  
1932  
1933 [3] : !Informational message  
1934 begin  
1935  
1936 selectoneu .REC_BUF [MSG_NUM] of  
1937 set  
1938  
1939 [0, 9] : !Format begun plus my debug msg  
1940 begin  
1941 PRINTF (FMT1, MSGADR);  
1942 end;  
1943  
1944 [otherwise] : !This message number is unknown  
1945 begin  
1946 RET_STATUS = UMN_CODE; !Unknown message number error code  
1947 DECODE (); !Report error and die  
1948 end;  
1949 tes;  
1950  
1951 end;  
1952  
1953 [4] : !Termination message
```

```
1954 begin  
1955 PRINTF (FMT4, .REC_BUF[MSG_NUM]);  
1956 WRT_RDRX (RCIP, RC_ALL, ONES); !Stop the remote program  
1957 EXIT_TST;  
1958 end;  
1959  
1960 [5] : !Fatal error message  
1961 begin  
1962  
1963 PRINTF (.FMT_ERR [.REC_BUF[MSG_NUM] - 1]);  
1964 WRT_RDRX (RCIP, RC_ALL, ONES); !Stop the remote program  
1965 DOCN; !Kill this formatter and return to init code  
1966  
1967 end;  
1968  
1969 [6] : !Special message type  
1970 begin  
1971  
1972 selectu .REC_BUF [MSG_NUM] of  
1973 set  
1974 [always] :  
1975 begin  
1976 RET_STATUS = UMN_CODE;  
1977 DECODE ();  
1978 end;  
1979 tes;  
1980  
1981 end;  
1982  
1983 [otherwise] : !This message number is unknown  
1984 begin  
1985 RET_STATUS = UMT_CODE; !Unknown message number error code  
1986 DECODE (); !Report error and die  
1987 end;  
1988 tes;  
1989  
1990 end;  
1991  
1992 ENDTST;
```

```
.TITLE ZRQB3 RDRX DISK FORMATTER  
.IDENT /REV A /  
.ENABL AMA  
  
.GLOBL COPY, DATE.CHECK, ABORT, GET.DUST.STATUS  
.GLOBL EX.LOC.PROG, REC.DATA, SEND.DATA  
.GLOBL SET.CNTRL.CHAR, DUP$I.SERVICE  
.GLOBL INT$I.SERVICE, INIT.COM.AREA, BOOT.RDRX  
.GLOBL DECODE, NXT.CRN, RET.EN$AD, SND.BUF  
.GLOBL REC.BUF, MSGADR, NSD.SLOT, NRD.SLOT  
.GLOBL VEC.ADDR, RET.STATUS, PID.SAVE  
.GLOBL RDRX.ADDR, PTBL.PTR, SW.UNIT.NO
```

.GLOBL SW.MODE, BOOT.FAILURE, PROTO.VIOLATION
.GLOBL PORT.INIT.ERR, DEF.SERIAL, DEF.DATE
.GLOBL FMT1, FMT4, FMT.ERR, UNITS.MSG
.GLOBL EXISTS.MSG, DOWNS.MSG, INACCS.MSG
.GLOBL DFLT\$.MSG, SERIAL\$.MSG, DATES.MSG

.SBTTL ST1 FORMATTER SECTION 1
.PSECT \$CODE\$, RO

Address	Offset	Label	Instruction	Comment	Address
000000					
000000	004137	000000G	ST1: JSR R1,\$SAVE2		1610
000004	162706	000006	SUB #6,SP		
000010	005000		CLR R0		1647
000012	104441		TRAP 41		
000014	013700	000000G	MOV VEC.ADDR,R0		1648
000020	104436		TRAP 36		
000022	012746	000340	MOV #340,-(SP)		1649
000026	012746	000000G	MOV #INT\$I.SERVICE,-(SP)		
000032	013746	000000G	MOV VEC.ADDR,-(SP)		
000036	012746	000003	MOV #3,-(SP)		
000042	104437		TRAP 37		
000044	012701	177777	MOV #-1,R1	: *,RETRIES	1656
000050	004737	000000G	1\$: JSR PC,BOOT.RDRX		1660
000054	010037	000000G	MOV R0,RET.STATUS		
000060	005201		INC R1	: RETRIES	1661
000062	032700	000001	BIT #1,R0	: *,RET.STATUS	1663
000066	001003		BNE 2\$		
000070	020127	000001	CMP R1,#1	: RETRIES,*	
000074	001365		BNE 1\$		
000076	032737	000001 000000G	2\$: BIT #1,RET.STATUS		1669
000104	001011		BNE 3\$		
000106	012716	000000G	MOV #BOOT.FAILURE,(SP)		1672
000112	012746	000001	MOV #1,-(SP)		
000116	010600		MOV SP,R0	: SP,*	
000120	104417		TRAP 17		
000122	104424		TRAP 24		
000124	104444		TRAP 44		1673
000126	005726		TST (SP)+		1671
000130	004737	000000G	3\$: JSR PC,INIT.COM.AREA		1689
000134	006000		ROR R0		
000136	103016		BCC 4\$		
000140	012716	000000G	MOV #PROTO.VIOLATION,(SP)		1692
000144	012746	000001	MOV #1,-(SP)		
000150	010600		MOV SP,R0	: SP,*	
000152	104417		TRAP 17		
000154	012716	000000G	MOV #PORT.INIT.ERR,(SP)		1693
000160	012746	000001	MOV #1,-(SP)		
000164	010600		MOV SP,R0	: SP,*	
000166	104417		TRAP 17		
000170	104444		TRAP 44		
000172	022626		CMP (SP)+,(SP)+		1691
000174	013700	000000G	4\$: MOV VEC.ADDR,R0		1704
000200	104436		TRAP 36		

000202	012716	000340		MOV	#340,(SP)	:	1705
000206	012746	000000G		MOV	#DUP\$I.SERVICE,-(SP)	:	
000212	013746	000000G		MOV	VEC.ADDR,-(SP)	:	
000216	012746	000003		MOV	#3,-(SP)	:	
000222	104437			TRAP	37	:	
000224	013700	000000G		MOV	RDRX.ADDR,R0	:	1712
000230	016066	000002	000022	MOV	2(R0),22(SP)	:	
000236	012701	000001		MOV	#1,R1	:	
000242	010160	000002		MOV	R1,2(R0)	:	
000246	004737	000000G		JSR	PC,SET.CNTRL.CHAR	:	1732
000252	006000			ROR	R0	:	
000254	103002			BCC	5\$:	
000256	004737	000000G		JSR	PC,DECODE	:	
000262	004737	000000G	5\$:	JSR	PC,GET.DUST.STATUS	:	1751
000266	006000			ROR	R0	:	
000270	103002			BCC	6\$:	
000272	004737	000000G		JSR	PC,DECODE	:	
000276	013700	000000G	6\$:	MOV	RET.ENSAD,R0	:	1757
000302	032760	004000	000022	BIT	#4000,22(R0)	:	
000310	001401			BEQ	7\$:	
000312	104444			TRAP	44	:	1759
000314	004737	000000G	7\$:	JSR	PC,EX.LOC.PROG	:	1772
000320	006000			ROR	R0	:	
000322	103002			BCC	8\$:	
000324	004737	000000G		JSR	PC,DECODE	:	
000330	004737	000000G	8\$:	JSR	PC,GET.DUST.STATUS	:	1783
000334	006000			ROR	R0	:	
000336	103002			BCC	9\$:	
000340	004737	000000G		JSR	PC,DECODE	:	
000344	013700	000000G	9\$:	MOV	RET.ENSAD,R0	:	1789
000350	032760	004000	000022	BIT	#4000,22(R0)	:	
000356	001002			BNE	10\$:	
000360	104444			TRAP	44	:	1791
000362	000405			BR	11\$:	1789
000364	013700	000000G	10\$:	MOV	RET.ENSAD,R0	:	1798
000370	016037	000024	000000G	MOV	24(R0),PID.SAVE	:	
000376	004737	000000G	11\$:	JSR	PC,REC.DATA	:	1837
000402	006000			ROR	R0	:	
000404	103002			BCC	12\$:	
000406	004737	000000G		JSR	PC,DECODE	:	
000412	005000		12\$:	CLR	R0	:	
000414	005060	000000G	13\$:	CLR	SND.BUF(R0)	:	
000420	062700	000002		ADD	#2,R0	:	
000424	020027	000110		CMP	R0,#110	:	
000430	101771			BLOS	13\$:	
000432	013701	000000G		MOV	REC.BUF,R1	:	1848
000436	006201			ASR	R1	:	
000440	006201			ASR	R1	:	
000442	006201			ASR	R1	:	
000444	006201			ASR	R1	:	
000446	000301			SWAB	R1	:	
000450	042701	177760		BIC	#177760,R1	:	
000454	020127	000001		CMP	R1,#1	:	

ZRQB3 RDRX DISK FORMATTER
REV A PATCH 00 FORMATTER-SECTION 1

28-Jun-1983 13:03:45
27-Jun-1983 18:06:20

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB3.B16;1 (7)

000460	001165				BNE	23\$				
000462	013702	000000G			MOV	REC.BUF,R2	:			1860
000466	042702	170000			BIC	#170000,R2	:			
000472	001023				BNE	16\$				
000474	104450				TRAP	50	:			1865
000476	103011				BHIS	14\$				
000500	104443				TRAP	43				
000502	000406				.WORD	406				
000504	000000G				.WORD	SND.BUF				
000506	000042				.WORD	42				
000510	000000G				.WORD	UNITS.MSG				
000512	000003				.WORD	3				
000514	000000				.WORD	0				
000516	000003				.WORD	3				
000520	000403				BR	15\$				
000522	013737	000000G	000000G	14\$:	MOV	SW.UNIT.NO,SND.BUF	:			1866
000530	004737	000000G		15\$:	JSR	PC,SEND.DATA	:			1867
000534	006000				ROR	R0				
000536	103317				BCC	11\$				
000540	000543				BR	25\$				
000542	020227	000001		16\$:	CMP	R2,#1	:			1860
000546	001061				BNE	18\$				
000550	104450				TRAP	50	:			1872
000552	103053				BHIS	17\$				
000554	005037	000000G			CLR	SND.BUF	:			1874
000560	104443				TRAP	43	:			1875
000562	000404				.WORD	404				
000564	000000G				.WORD	SND.BUF				
000566	000130				.WORD	130				
000570	000000G				.WORD	EXISTS.MSG				
000572	000001				.WORD	1				
000574	032737	000001	000000G		BIT	#1,SND.BUF	:			1876
000602	001352				BNE	15\$				
000604	104443				TRAP	43	:			1878
000606	000404				.WORD	404				
000610	000000G				.WORD	SND.BUF				
000612	000130				.WORD	130				
000614	000000G				.WORD	DOWN\$.MSG				
000616	000002				.WORD	2				
000620	032737	000002	000000G		BIT	#2,SND.BUF	:			1879
000626	001340				BNE	15\$				
000630	104443				TRAP	43	:			1881
000632	000404				.WORD	404				
000634	000000G				.WORD	SND.BUF				
000636	000130				.WORD	130				
000640	000000G				.WORD	INACCS.MSG				
000642	000004				.WORD	4				
000644	032737	000004	000000G		BIT	#4,SND.BUF	:			1882
000652	001326				BNE	15\$				
000654	152737	000001	000000G		BISB	#1,SND.BUF	:			1884
000662	012716	000000G			MOV	#DFLT\$.MSG,(SP)	:			1885
000666	012746	000001			MOV	#1,-(SP)	:			
000672	010600				MOV	SP,R0	:	SP,*		

ZRQB3
REV A PATCH 00

RDRX DISK FORMATTER
FORMATTER SECTION 1

H 7

28-Jun-1983 13:03:45
27-Jun-1983 18:06:20

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB3.S16;1 (7)

SEQ 85
Page 16

000674	104417			TRAP	17			
000676	005726			TST	(SP)+	:		1883
000700	000713			BR	15\$:		1872
000702	013737	000000G	000000G	17\$: MOV	SW.MODE,SND.BUF	:		1890
000710	000707			BR	15\$:		1892
000712	020227	000002		18\$: CMP	R2,#2	:		1860
000716	001025			BNE	21\$:		
000720	104450			TRAP	50	:		1897
000722	103011			BHIS	19\$			
000724	104443			TRAP	43			
000726	000406			.WORD	406			
000730	000000G			.WORD	SND.BUF			
000732	000142			.WORD	142			
000734	000000G			.WORD	SERIALS.MSG			
000736	177777			.WORD	-1			
000740	000010			.WORD	10			
000742	000010			.WORD	10			
000744	000671			BR	15\$			
000746	012716	000000G		19\$: MOV	#DEF.SERIAL,(SP)	:		1898
000752	012746	000000G		20\$: MOV	#SND.BUF,-(SP)			
000756	012746	000010		MOV	#10,-(SP)			
000762	004737	000000G		JSR	PC,COPY			
000766	022626			CMP	(SP)+,(SP)+			
000770	000657			BR	15\$:		1899
000772	020227	000003		21\$: CMP	R2,#3	:		1860
000776	001021			BNE	24\$:		
001000	104450			TRAP	50	:		1905
001002	103011			BHIS	22\$			
001004	104443			TRAP	43			
001006	000406			.WORD	406			
001010	000000G			.WORD	SND.BUF			
001012	000142			.WORD	142			
001014	000000G			.WORD	DATES.MSG			
001016	177777			.WORD	-1			
001020	000010			.WORD	10			
001022	000010			.WORD	10			
001024	000641			BR	15\$			
001026	012716	000000G		22\$: MOV	#DEF.DATE,(SP)	:		1906
001032	000747			BR	20\$:		
001034	020127	000002		23\$: CMP	R1,#2	:		1848
001040	001004			BNE	26\$:		
001042	012737	004001	000000G	24\$: MOV	#4001,RET.STATUS	:		1926
001050	000513			25\$: BR	31\$:		1927
001052	020127	000003		26\$: CMP	R1,#3	:		1848
001056	001022			BNE	28\$:		
001060	013700	000000G		MOV	REC.BUF,RO	:		1936
001064	042700	170000		BIC	#170000,RO			
001070	001403			BEQ	27\$			
001072	020027	000011		CMP	RO,#11			
001076	001361			BNE	24\$			
001100	012716	000000G		27\$: MOV	#MSGADR,(SP)	:		1941
001104	012746	000000G		MOV	#FMT1,-(SP)			
001110	012746	000002		MOV	#2,-(SP)			

ZRQB3
REV A PATCH 00

RDRX DISK FORMATTER
FORMATTER SECTION 1

28-Jun-1983 13:03:45
27-Jun-1983 18:06:20

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB3.B16;1 (7)

001114	010600			MOV	SP,R0	:	SP,*	
001116	104417			TRAP	17	:		
001120	022626			CMP	(SP)+,(SP)+	:		1940
001122	000470			BR	32\$:		1936
001124	020127	000004	28\$:	CMP	R1,#4	:		1848
001130	001025			BNE	29\$:		
001132	013716	000000G		MOV	REC.BUF,(SP)	:		1955
001136	042716	170000		BIC	#170000,(SP)	:		
001142	012746	000000G		MOV	#FMT4,-(SP)	:		
001146	012746	000002		MOV	#2,-(SP)	:		
001152	010600			MOV	SP,R0	:	SP,*	
001154	104417			TRAP	17	:		
001156	017766	000000G	000024	MOV	@RDRX.ADDR,24(SP)	:	*,RCSS.REG	1956
001164	012700	177777		MOV	#-1,R0	:	*,RCSM.REG	
001170	010077	000000G		MOV	R0,@RDRX.ADDR	:	RCSM.REG,*	
001174	104463			TRAP	63	:		
001176	062706	000022		ADD	#22,SP	:		1848
001202	000442			BR	33\$:		1954
001204	020127	000005	29\$:	CMP	R1,#5	:		1848
001210	001025			BNE	30\$:		
001212	013700	000000G		MOV	REC.BUF,R0	:		1963
001216	042700	170000		BIC	#170000,R0	:		
001222	006300			ASL	R0	:		
001224	016016	177776G		MOV	FMT.ERR-2(R0),(SP)	:		
001230	012746	000001		MOV	#1,-(SP)	:		
001234	010600			MOV	SP,R0	:	SP,*	
001236	104417			TRAP	17	:		
001240	017766	000000G	000020	MOV	@RDRX.ADDR,20(SP)	:	*,RCSS.REG	1964
001246	012700	177777		MOV	#-1,R0	:	*,RCSM.REG	
001252	010077	000000G		MOV	R0,@RDRX.ADDR	:	RCSM.REG,*	
001256	104444			TRAP	44	:		
001260	005726			TST	(SP)+	:		1961
001262	000410			BR	32\$:		1848
001264	020127	000006	30\$:	CMP	R1,#6	:		
001270	001664			BEQ	24\$:		1976
001272	012737	001001	000000G	MOV	#1001,RET.STATUS	:		1985
001300	004737	000000G	31\$:	JSR	PC,DECODE	:		1986
001304	000137	000376'	32\$:	JMP	11\$:		1807
001310	062706	000006	33\$:	ADD	#6,SP	:		1610
001314	000207			RTS	PC	:		

; Routine Size: 359 words, Routine Base: \$CODE\$ + 0000
; Maximum stack depth per invocation: 17 words

000000	004737	000000'	T1::	.SBTTL	T1 FORMATTER SECTION 1	:		
000000			1\$:	JSR	PC,\$T1	:		1990
000004	104466			TRAP	66	:		
000006	006000			ROR	R0	:		
000010	103773			BLO	1\$:		

ZRQB3 RDRX DISK FORMATTER
REV A PATCH 00 FORMATTER SECTION 1

J 7
28-Jun-1983 13:03:45
27-Jun-1983 18:06:20

SEQ 87
Page 18
VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB3.B16;1 (7)

000012 000207

RTS PC

: Routine Size: 6 words, Routine Base: \$CODE\$ + 1316
: Maximum stack depth per invocation: 2 words

: 1993 end
: 1994
: 1995 eludom

: OTS external references
: .GLOBL \$SAVE2

: PSECT SUMMARY

: Psect Name Words Attributes
: \$CODE\$ 365 RO , I , LCL, REL, CON

: LIBRARY STATISTICS

File	----- Total	Symbols Loaded	----- Percent	Blocks Read
DISK\$USER:[PRUCHA.RELEASE]ZRQBA0.L16;2	356	204	57	41

: COMMAND QUALIFIERS

: Size: 365 code + 0 data words
: Run Time: 00:17.2
: Elapsed Time: 00:47.7
: Memory Used: 296 pages
: Compilation Complete

ZRQB4

RDRX DISK FORMATTER

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (1)

.....

```

0001 MODULE ZRQB4 (%TITLE 'RDRX DISK FORMATTER'
0002                IDENT = 'REV A PATCH 00',
0003                ADDRESSING MODE (ABSOLUTE) ,
0004                ENVIRONMENT (NOEIS)
0005                ) =
0006 BEGIN
0007
0008 !                SUBROUTINES
0009
0010 library 'ZRQBA0.L16';
0011
0012 require 'BLSMAC.REQ';
1503
1504 %sbttl 'MODULE DECLARATIONS'

```

```

!Define RDRX Formatter library
!Define Bliss Macro require file

```

```

: 1505
: 1506 forward routine
: 1507 GET_NSD, !Get next send descriptor slot index
: 1508 GET_NRD, !Get next receive descriptor slot index
: 1509 LOAD_OUT$STD_BUF, !Load out standing command buffer
: 1510 GET_CMD$REF, !Get unique command reference number
: 1511 DECODE : novalue, !Decode return status error code
: 1512 DUP$I_SERVICE : INT_LNK$TYP novalue, !Dup/UQ port interrupt service routine
: 1513 CTO_WAIT, !Command time out wait
: 1514 ABORT, !Abort Dup command
: 1515 GET_DUST_STATUS, !Get Dust Status command
: 1516 EX_LOC_PROG, !Execute Local Program command
: 1517 SEND_DATA, !Send Data command
: 1518 REC_DATA, !Receive Data command
: 1519 SET_CNTL$R_CHAR, !Set Controller Characteristics command
: 1520 INT$I_SERVICE : INT_LNK$TYP novalue, !Initialization sequence interrupt service
: 1521 IS_TIMER, !Initialization sequence time-out wait
: 1522 BOOT_RDRX, !Initialize sequence for RDRX controller
: 1523 INIT_COM_AREA; !Initialize UQ Port communication area
: 1524

```

ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
MODULE DECLARATIONS

M 7

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (3)

SEQ 90
Page 3

```
1525 | +
1526 | | The psect named "code or $code$" is redefined here
1527 | | to be called "ac$code". This is done to organize
1528 | | formatter routine code into a seperate psect.
1529 | | -
1530 | | psect
1531 | | code = ac$code;
1532 | |
1533 | | +
1534 | | Structure declarations used within this module.
1535 | | -
1536 | |
1537 | | structure
1538 | |
1539 | | | +
1540 | | | RDRX register accessing structure. This
1541 | | | structure allows RDRX register accessing
1542 | | | to be transportable between the PDP-11 and
1543 | | | VAX Diagnostic Supervisors.
1544 | | |
1545 | | | This also defines an access algorithm for
1546 | | | VAX to allow field reference to MBA address
1547 | | | space without generating machine checks.
1548 | | | -
1549 | | |
1550 | | RDRX [O, P, S, E] =
1551 | | begin
1552 | | local
1553 | | RCSS_REG;
1554 | |
1555 | | RCSS_REG = .(RDRX + %upval*0)<0, %bpval, 0>;
1556 | | RCSS_REG
1557 | | end
1558 | | <P, S, E>;
1559 | |
1560 | |
```

```
1561 !+
1562 ! External Declaration of datums declared outside of this module.
1563 !-
1564
1565 external
1566
1567 ! Communications area declarations
1568
1569 COM_AREA : blockvector [REC_ALLOCATE + SND_ALLOCATE + HDR_SIZ, 2, word],
1570 HEAD_AREA : ref block [4, word] field (HDR_FIELD),
1571 RECEIVE_RING : ref blockvector [REC_ALLOCATE, 2, word] field (DSC_FIELD),
1572 SEND_RING : ref blockvector [SND_ALLOCATE, 2, word] field (DSC_FIELD),
1573 REC_ENVELOPE : blockvector [REC_ALLOCATE, RB_SIZE + 2, word] field (ENV_FIELD),
1574 SND_ENVELOPE : blockvector [SND_ALLOCATE, SB_SIZE + 2, word] field (ENV_FIELD),
1575 RET_EN$AD : ref block [RB_SIZE + 2, word] field (ENV_FIELD),
1576 REC_BUF : block [RECB_SIZE, word] field (RECB_FIELD),
1577 SND_BUF : vector [SNDB_SIZE, word],
1578 OUT$STD_BUF : blockvector [REC_ALLOCATE, 2, word] field (OUT$FIELD),
1579
1580 ! Miscellaneous external data declarations
1581
1582 NRD_SLOT : word, !Next receive descriptor slot
1583 NSD_SLOT : word, !Next send descriptor slot
1584 RDRX_ADDR : ref RDRX field (ISD_FIELD), !Controller reg access struct
1585 RET_STATUS : word, !Global return status location
1586 PID_SAVE : word, !Saves program indicator field
1587 VEC_ADDR : word, !RDRX interrupt vector address
1588 UC_VER : byte, !Micro code version storage
1589 R$VD_STRUCT : vector [4, word], !Stores init seq reserved fields
1590 ISD_STRUCT : blockvector [4, 2, word] field (ISD_FIELD), !Init seq data
1591 UNIT_NO : word, !P table unit number to format
1592 NXT_CRN : byte, !Stores next cmd ref number
1593
1594 ! Error Messages Structures
1595
1596 PFE_STRUCT : vector [22], !Port fatal error msg struct
1597 EMSG_STRUCT : vector [22]; !Error message structure
1598
1599 !
```

ZRQB4 RDRX DISK FORMATTER
REV A PATCH 00 GLOBAL ROUTINE DECLARATIONS

B 8
28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (5)

SEQ 92
Page 5

; 1600 %sbttl 'GLOBAL ROUTINE DECLARATIONS'

```
1601 global routine GET_NSD =      !Chooses the next send slot
1602
1603 !++
1604 Functional Description :
1605     This routine will determine which send ring descriptor the
1606     port/controller is polling and returns that dsc slot number
1607     to the calling routine. This host program will call this
1608     routine each time it wishes to deposit another command into the
1609     command (send) ring.
1610
1611 Formal Parameters :
1612     none
1613
1614 Implicit Inputs :
1615     NSD_SLOT :      Global storage for the next send descriptor slot.
1616                   Stores where the host should place this command for
1617                   processing by the port/controller.
1618
1619 Implicit Outputs :
1620     The global storage 'Nsd_slot' is updated to the
1621     present send slot where the port/controller is polling.
1622
1623 Completion Codes :
1624     Returns the contents of 'Nsd_slot' to the calling routine.
1625
1626 Side Effects :
1627     none
1628 --
1629
1630 begin
1631     Increment the next send descriptor_slot by one
1632     NSD_SLOT = .NSD_SLOT + 1;
1633     Set the slot pointer back to zero if it wraps around
1634     to the top of the ring.
1635
1636     if .NSD_SLOT gtru SND_ALLOCATE - 1 then NSD_SLOT = ZERO;
1637
1638     Return the next send descriptor_slot to the caller
1639
1640     return .NSD_SLOT;
1641 end;
```

```
.TITLE ZRQB4 RDRX DISK FORMATTER
.IDENT /REV A /
.ENABL AMA

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

ZRQB4 RDRX DISK FORMATTER
REV A PATCH 00 GLOBAL ROUTINE DECLARATIONS

D 8

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (6)

SEQ 94
Page 7

.GLOBL SEND.RING, REC.ENVELOPE, SND.ENVELOPE
.GLOBL RET.ENSAD, 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
.PSECT \$CODE\$, RO

000000

000000 005237 000000G

GET.NSD::

INC NSD.SLOT ;
CMP NSD.SLOT,#3 ;
BLOS 1\$;
CLR NSD.SLOT ;
1\$: MOV NSD.SLOT,RO ;
RTS PC ;

1634
1640

000004 023727 000000G 000003

000012 101402

000014 005037 000000G

000020 013700 000000G

000024 000207

1630
1601

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

: 1647


```

: 1648 global routine GET_NRD =           !Chooses the next receive slot
: 1649
: 1650 !++
: 1651 Functional Description :
: 1652 This routine will determine which receive ring descriptor the
: 1653 port/controller is polling and returns that dsc slot number to
: 1654 the calling routine. This host program will call this routine
: 1655 each time it wishes to process another receive ring descriptor.
: 1656
: 1657 Formal Parameters :
: 1658 none
: 1659
: 1660 Implicit Inputs :
: 1661 NRD_SLOT : Global storage for the next receive descriptor slot.
: 1662 Stores where the port should return this commands
: 1663 response indicator.
: 1664
: 1665 Implicit Outputs :
: 1666 The global storage 'Nrd_slot' is updated to the
: 1667 present receive slot where the port/controller
: 1668 is polling.
: 1669
: 1670 Completion Codes :
: 1671 Returns the contents of 'Nrd_slot' to the calling routine.
: 1672
: 1673 Side Effects :
: 1674 none
: 1675 !--
: 1676
: 1677 begin
: 1678 |
: 1679 | Increment the next receive descriptor_slot by one
: 1680 |
: 1681 | NRD_SLOT = .NRD_SLOT + 1;
: 1682 |
: 1683 | Set the slot pointer back to zero if it wraps around
: 1684 | to the top of the ring.
: 1685 |
: 1686 |
: 1687 | if .NRD_SLOT gtru REC_ALLOCATE - 1 then NRD_SLOT = ZERO;
: 1688 |
: 1689 |
: 1690 | Return the next receive descriptor_slot to the caller
: 1691 |
: 1692 | return .NRD_SLOT;
: 1693 | end;

```

```

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

```

1681
1687

ZRQB4 RDRX DISK FORMATTER
REV A PATCH 00 GLOBAL ROUTINE DECLARATIONS

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (7)

000014	005037	000000G		CLR	NRD.SLOT		
000020	013700	000000G	1\$:	MOV	NRD.SLOT,R0	:	1677
000024	000207			RTS	PC	:	1648

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

: 1694

1695 global routine LOAD_OUT\$STD_BUF (REF_NUM) = !Load out\$std_buffer with this command

1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747

!++

Functional Description :

The outstanding command buffer "out\$std_buf" is used by this host program to determine if an outstanding command issued to the port has been processed yet. This is done by examining the receive flag 'Rec_flg' in a buffer slot for a '1' which is set by the interrupt service routine during response ring interrupts.

This buffer can be looked at as a window between the port driver receiving & processing the response envelopes and the host class driver issuing commands to the port.

This routine loads into an empty out\$std_buf slot the following values:

1. This commands reference number.
2. Clears 'rec_flg' indicating this command is outstanding.
3. Clears out the second word in slot where the returned envelope address will go.

IMPORTANT NOTE:

To quarentee a command loaded into the out\$std_buffer will never be lost (i.e. having this routine return a buffer slot not yet received by the interrupt service routine), only the cto_wait (controller time out wait) routine is permitted to return a out\$std_buf slot to the unused pool (i.e. by loading a slots first word with %o'100000'). This routine is therefore quarenteed to return an unused out\$std_buffer slot when this unique value of %o'100000' is found. To further quarentee this, unique command ref numbers will never use zero as a reference number.

Formal Parameters :

REF_NUM This is the unique reference number of this command set to the port.

Implicit Inputs :

none

Implicit Outputs :

none

Completion Codes :

The out\$standing buffer slot index where this command was put is routines value and is returned to the caller.

Side Effects :

none

```

1748 |
1749 | |
1750 | |
1751 | begin
1752 | |
1753 | |
1754 | | Search through the out standing command buffer
1755 | | and look for the first open slot. Return this
1756 | | first slot index to the caller if one is found.
1757 | | If no open slots are found then return an error code.
1758 | |
1759 | |
1760 | incru i from 0 to REC_ALLOCATE - 1 do      !Find the first open slot
1761 | |
1762 | | An open slot is by definition the value
1763 | | %o'100000' in the slots first word.
1764 | |
1765 | |
1766 | | if .OUT$STD_BUF [.i, CMD_WRD] eqlu %o'100000'  !Is this slot open
1767 | | then
1768 | | | begin
1769 | | | OUT$STD_BUF [.i, REC_FLG] = FALSE; !Clear the received flag
1770 | | | OUT$STD_BUF [.i, CMD_REF] = .REF_NUM; !Load this cmd's ref num
1771 | | | OUT$STD_BUF [.i, ENV_ADR] = ZERO; !Clear the previous env adr
1772 | | | return .i; !Return buffer index to caller
1773 | | | end;
1774 | | |
1775 | | |
1776 | | | The buffer is full if the code reaches
1777 | | | here. This should never happen so
1778 | | | report an error for debug purposes.
1779 | | |
1780 | | return RET_STATUS = OBF_CODE; !Report an 'out$std buffer full'' error
1781 | | end;

```

Address	Offset	Label	Instruction	Comment	Line No.
000000	004137	000000G	.SBTTL LOAD.OUT\$STD.BUF GLOBAL ROUTINE DECLARATIONS		
			JSR R1,\$SAVE2		1695
000004	005001		CLR R1	: I	1760
000006	010100		1\$: MOV R1,R0	: I,*	1766
000010	006300		ASL R0		
000012	006300		ASL R0		
000014	012702	000000G	MOV #OUT\$STD.BUF,R2		
000020	060002		ADD R0,R2		
000022	021227	100000	CMP (R2), #-100000		
000026	001010		BNE 2\$		
000030	042712	100000	BIC #100000,(R2)	: REF.NUM,*	1769
000034	116612	000010	MOVB 10(SP),(R2)		1770
000040	005060	000002G	CLR OUT\$STD.BUF+2(R0)		1771
000044	010100		MOV R1,R0	: I,*	1768
000046	000207		RTS PC		
000050	005201		2\$: INC R1	: I	1760

ZRQB4 RDRX DISK FORMATTER
REV A PATCH 00 GLOBAL ROUTINE DECLARATIONS

I 8
28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (8)

SEQ 99
Page 12

000052	020127	000003	CMP	R1,#3	:	I,*	
000056	101753		BLOS	1\$:		1780
000060	012700	002001	MOV	#2001,R0	:		
000064	010037	000000G	MOV	R0,RET.STATUS	:		1695
000070	000207		RTS	PC	:		

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

: 1782

```
1783 global routine GET_CMD$REF = !Gets next unique cmd ref number
1784
1785 !++
1786 Functional Description :
1787 A 32 bit unique non-zero number used to identify host commands.
1788 Class drivers should supply a unique reference number in each
1789 command that they send to a DUP server. A class driver may supply
1790 a zero reference number if it does not need to associate a command
1791 with its end message.
1792
1793 Command reference numbers must be unique across all commands that
1794 are outstanding on the same connection i.e., they must be unique
1795 across all outstanding commands issued by a single class driver
1796 (Host) to a single DUP server. The class driver may re-use a
1797 command reference number when the command is no longer
1798 outstanding -- i.e., after receiving the command's end message or
1799 after re-synchronizing with the DUP server. Command reference
1800 numbers need not be unique for commands issued by different class
1801 drivers --- i.e. commands issued by different host or commands for
1802 different DUP servers from the same host. Therefore controllers
1803 must internally use the combination of a command reference number
1804 and the connection on which the command was received as the unique
1805 identifier of an outstanding command.
1806
1807 This routine will generate a unique command reference number and
1808 will search the outstanding command buffer to see if already used.
1809 The first unused unique command reference found will be returned
1810 to the calling routine.
1811
1812 Formal Parameters :
1813 none
1814
1815 Implicit Inputs :
1816 NXT_CRN This global location stores the next unique cmd
1817 reference number to be used.
1818
1819 Implicit Outputs :
1820 NXT_CRN This global location is loaded with the next
1821 unique command reference number.
1822
1823 Completion Codes :
1824 The contents of .NXT_CRN is returned to the calling routine.
1825
1826 Side Effects :
1827 none
1828 --
1829
1830 begin
1831 local
1832 DONE;
1833 !
1834
1835
```

```

1836 ! Increment the global unique command
1837 ! reference number before anything is done.
1838 !
1839 NXT_CRN = .NXT_CRN + 1;
1840 !
1841 !+
1842 ! Repeat generating and searching the out$standing
1843 ! command buffer until a unique command reference
1844 ! number is found.
1845 !-
1846
1847 do
1848   begin
1849     !
1850     ! Wrap this next command reference number around
1851     ! back to one if it is greater than 255 (decimal).
1852     !
1853     !
1854     if .NXT_CRN gtr 255 then NXT_CRN = 1;
1855     !
1856     DONE = TRUE; !Clear the all done indicator flag
1857     !
1858     !+
1859     ! Now search the out$std_buffer for this command
1860     ! reference number. If not there then done stays
1861     ! true and the loop will end else increment to the
1862     ! next unique command ref number and make done false
1863     ! to continue the loop.
1864     !-
1865     !
1866     incru i from 0 to REC_ALLOCATE - 1 do !Search buffer for this cmd ref num
1867       !
1868       if .OUT$STD_BUF [.i, CMD_REF] eqlu .NXT_CRN !Does it already exist
1869       then
1870         begin !It already exists
1871           NXT_CRN = .NXT_CRN + 1; !Try the next sequential cmd ref num
1872           DONE = FALSE; !Make code loop again
1873           exitloop; !Exit this incr loop
1874         end;
1875       !
1876     end
1877 until .DONE; !Repeat loop until done
1878 !
1879 ! Return the unique command reference number to the caller.
1880 !
1881 return .NXT_CRN;
1882 end;
1883

```

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (9)

ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

000002	105237	000000G		INCB	NXT.CRN	:		1839
000006	012701	000001	1\$:	MOV	#1,R1	:	*,DONE	1856
000012	005000			CLR	R0	:	I	1866
000014	126037	000000G 000000G	2\$:	CMPB	OUT\$STD.BUF(R0),NXT.CRN	:	*(I),*	1868
000022	001004			BNE	3\$			
000024	105237	000000G		INCB	NXT.CRN	:		1871
000030	005001			CLR	R1	:	DONE	1872
000032	000405			BR	4\$:		1870
000034	062700	000004	3\$:	ADD	#4,R0	:	*,I	1866
000040	020027	000014		CMP	R0,#14	:	I,*	
000044	101763			BLOS	2\$			
000046	006001		4\$:	ROR	R1	:	DONE	1877
000050	103356			BCC	1\$			
000052	005000			CLR	R0	:		1830
000054	153700	000000G		BISB	NXT.CRN,R0	:		
000060	012601			MOV	(SP)+,R1	:		1783
000062	000207			RTS	PC	:		

; Routine Size: 26 words, Routine Base: \$CODE\$ + 0146
; Maximum stack depth per invocation: 2 words

; 1884


```
1885 global routine DECODE : novalue =      !Decodes failing SA reg data
1886
1887 !++
1888 Functional Description :
1889 Due to the implimentation of the DUP and UQ Port protocol there
1890 are two levels at which an issued command to a port/controller
1891 can fail and they are:
1892
1893     1. The issued command can time out.
1894
1895     2. An error can be posted in SA register bit 15 by the port to
1896        report an error.
1897
1898     3. The issued command to the port/controller can be executed
1899        correctly without any errors but the response packet status
1900        field could have an error or status other than success posted.
1901
1902 These errors or status's returned are all returned to the host
1903 routine which queued the DUP command via the global storage
1904 "RET_STATUS". The host to port/controller communications
1905 connection having the highest priority (ie. if the SA Reg error
1906 bit is set the DUP interrupt routine returns a PFE_CODE and this
1907 code is passed up to the calling host routine with out being
1908 intercepted by any routine on the way up).
1909
1910 This routine will then be called when the return from a queued
1911 command comes back with an error code or non successfull status
1912 code. This is by definition when bit 0 in the returned status
1913 is equal to 1.
1914
1915 An appropriate recovery action will be done for each individual
1916 error.
1917
1918 Formal Parameters :
1919     none
1920
1921 Implicit Inputs :
1922     RET_STATUS:      Stored in this global storage is the returned error
1923                     code or non-successful status code from a queued
1924                     command.
1925
1926 Implicit Outputs :
1927     none
1928
1929 Completion Codes :
1930     none
1931
1932 Side Effects :
1933     All formatter errors are fatal, therefor after execution
1934     of this routine the RDRX controller is initialized
1935     aborting any DM code running in the controller.
1936
1937 !--
```

```
1938 begin
1939
1940 |+
1941 | Use the contents of "RET_STATUS" to select what
1942 | type error or non-successful status code is to
1943 | be processed.
1944 |-
1945
1946 selectoneu .RET_STATUS of
1947 set
1948 |
1949 | "Communication area initialize" error code
1950 |
1951 | This error code indicates that the port did
1952 | not init the com area in the host memory after
1953 | step 2 of the initialization sequence.
1954 |
1955
1956 [CIE_CODE] : !Code equals %'01'
1957 | begin
1958 | PRINTF (.EMSG_STRUCT [MSG4]);
1959 | end;
1960 |
1961 | "Port/Controller time out" error code
1962 |
1963 | Port/Controller timed out after the specified
1964 | time out interval.
1965 |
1966
1967 [CTO_CODE] : !Code equals %'11'
1968 | begin
1969 | PRINTF (.EMSG_STRUCT [MSG21]);
1970 | end;
1971 |
1972 | "Port fatal error" code
1973 |
1974 | The error bit in the SA Register was set when
1975 | examined in the DUP$I_SERVICE routine. This
1976 | error indicates a Port fatal error code.
1977 |
1978
1979 [PFE_CODE] : !Code equals %'21'
1980 | begin
1981 | PRINTF (.PFE_STRUCT [.RDRX_ADDR [RCSA, ERR_CODE]]);
1982 | end;
1983 |
1984 | "Return status error" code
1985 |
1986 | This indicates that a non-successful return status
1987 | code was returned from an issued command.
1988 |
1989
1990 [RSE_CODE] : !Code equals %'31'
```

```
1991      begin
1992      PRINTF (.EMSG_STRUCT [MSG0]);
1993      end;
1994      |
1995      | 'Port Portocol violation' error code
1996      |
1997      | A protocol violation error was detected during
1998      | host processing of an issued command.
1999      |
2000      |
2001      [PVE_CODE] :                               !Code equals %o'41'
2002      begin
2003      PRINTF (.EMSG_STRUCT [MSG1]);
2004      end;
2005      |
2006      | 'Remote program died' error code
2007      |
2008      | This indicates that the remote program running
2009      | in the DM machine did not responded within the
2010      | designated time out interval and that the progress
2011      | indicator was not increase after subsiquent time out
2012      | delays. It is assumed that the remote program is dead
2013      | and is treated as a fatal error.
2014      |
2015      |
2016      [RPD_CODE] :                               !Code equals %o'51'
2017      begin
2018      PRINTF (.EMSG_STRUCT [MSG2]);
2019      end;
2020      |
2021      | 'Port to host synchronious error'' code
2022      |
2023      |
2024      [PSE_CODE] :                               !Code equals %o'61'
2025      begin
2026      PRINTF (.EMSG_STRUCT [MSG5]);
2027      end;
2028      |
2029      | 'Message length error'' code
2030      |
2031      |
2032      [MLE_CODE] :                               !Code equals %o'71'
2033      begin
2034      PRINTF (.EMSG_STRUCT [MSG6]);
2035      end;
2036      |
2037      | 'Unknown end code'' error code
2038      |
2039      |
2040      [UEC_CODE] :                               !Code equals %o'101'
2041      begin
2042      PRINTF (.EMSG_STRUCT [MSG7]);
2043      end;
```

```
2044 |
2045 | "Adaptor purge request" error code
2046 |
2047 |
2048 [APR_CODE] : !Code equals %o'201'
2049 |   begin
2050 |   PRINTF (.EMSG_STRUCT [MSG8]);
2051 |   end;
2052 |
2053 | "Unknown interrupt" error code
2054 |
2055 |
2056 [UIN_CODE] : !Code equals %o'301'
2057 |   begin
2058 |   PRINTF (.EMSG_STRUCT [MSG9]);
2059 |   end;
2060 |
2061 | "ATTENTION MSG ENDCODE" error code
2062 |
2063 |
2064 [ATN_CODE] : !Code equals %o'401'
2065 |   begin
2066 |   PRINTF (.EMSG_STRUCT [MSG12]);
2067 |   end;
2068 |
2069 | "COMMAND MSG ENDCODE" error code
2070 |
2071 |
2072 [CMD_CODE] : !Code equals %o'501'
2073 |   begin
2074 |   PRINTF (.EMSG_STRUCT [MSG13]);
2075 |   end;
2076 |
2077 | "SERIOUS EXCEPTION" error code
2078 |
2079 |
2080 [SEX_CODE] : !Code equals %o'601'
2081 |   begin
2082 |   PRINTF (.EMSG_STRUCT [MSG14]);
2083 |   end;
2084 |
2085 | "INVALID COMMAND" error code
2086 |
2087 |
2088 [IVC_CODE] : !Code equals %o'701'
2089 |   begin
2090 |   PRINTF (.EMSG_STRUCT [MSG15]);
2091 |   end;
2092 |
2093 | "UNKNOWN MESSAGE TYPE" error code
2094 |
2095 |
2096 [UMT_CODE] : !Code equals %o'1001'
```


ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

E 9
28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (10)

SEQ 108
Page 21

: 2150
: 2151 end:

			.SBTTL	DECODE GLOBAL ROUTINE DECLARATIONS	
000000	010146		DECODE::	MOV R1,-(SP)	1885
000002	024646			CMP -(SP),-(SP)	
000004	013701	000000G		MOV RET.STATUS,R1	1946
000010	020127	000001		CMP R1,#1	
000014	001007			BNE 1\$	
000016	013746	000010G		MOV EMSG.STRUCT+10,-(SP)	1958
000022	012746	000001		MOV #1,-(SP)	
000026	010600			MOV SP,R0	: SP,*
000030	104417			TRAP 17	
000032	000567			BR 12\$	1946
000034	020127	000011	1\$:	CMP R1,#11	
000040	001007			BNE 2\$	
000042	013746	000052G		MOV EMSG.STRUCT+52,-(SP)	1969
000046	012746	000001		MOV #1,-(SP)	
000052	010600			MOV SP,R0	: SP,*
000054	104417			TRAP 17	
000056	000567			BR 14\$	1946
000060	020127	000021	2\$:	CMP R1,#21	
000064	001021			BNE 3\$	
000066	013700	000000G		MOV RDRX.ADDR,R0	1981
000072	016066	000002	000002	MOV 2(R0),2(SP)	: *,RCSS.REG
000100	016600	000002		MOV 2(SP),R0	: RCSS.REG,*
000104	042700	174000		BIC #174000,R0	
000110	006300			ASL R0	
000112	016046	000000G		MOV PFE.STRUCT(R0),-(SP)	
000116	012746	000001		MOV #1,-(SP)	
000122	010600			MOV SP,R0	: SP,*
000124	104417			TRAP 17	
000126	000567			BR 17\$	1946
000130	020127	000031	3\$:	CMP R1,#31	
000134	001007			BNE 4\$	
000136	013746	000000G		MOV EMSG.STRUCT,-(SP)	1992
000142	012746	000001		MOV #1,-(SP)	
000146	010600			MOV SP,R0	: SP,*
000150	104417			TRAP 17	
000152	000567			BR 19\$	1946
000154	020127	000041	4\$:	CMP R1,#41	
000160	001007			BNE 5\$	
000162	013746	000002G		MOV EMSG.STRUCT+2,-(SP)	2003
000166	012746	000001		MOV #1,-(SP)	
000172	010600			MOV SP,R0	: SP,*
000174	104417			TRAP 17	
000176	000567			BR 21\$	1946
000200	020127	000051	5\$:	CMP R1,#51	
000204	001007			BNE 6\$	
000206	013746	000004G		MOV EMSG.STRUCT+4,-(SP)	2018
000212	012746	000001		MOV #1,-(SP)	
000216	010600			MOV SP,R0	: SP,*

ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

000220	104417		TRAP	17			
000222	000567		BR	23\$:		1946
000224	020127	000061	6\$:	CMP	R1,#61		
000230	001007		BNE	7\$			
000232	013746	000012G	MOV	EMSG.STRUCT+12,-(SP)	:		2026
000236	012746	000001	MOV	#1,-(SP)			
000242	010600		MOV	SP,R0	:	SP,*	
000244	104417		TRAP	17			
000246	000567		BR	25\$:		1946
000250	020127	000071	7\$:	CMP	R1,#71		
000254	001007		BNE	8\$			
000256	013746	000014G	MOV	EMSG.STRUCT+14,-(SP)	:		2034
000262	012746	000001	MOV	#1,-(SP)			
000266	010600		MOV	SP,R0	:	SP,*	
000270	104417		TRAP	17			
000272	000576		BR	28\$:		1946
000274	020127	000101	8\$:	CMP	R1,#101		
000300	001007		BNE	9\$			
000302	013746	000016G	MOV	EMSG.STRUCT+16,-(SP)	:		2042
000306	012746	000001	MOV	#1,-(SP)			
000312	010600		MOV	SP,R0	:	SP,*	
000314	104417		TRAP	17			
000316	000564		BR	28\$:		1946
000320	020127	000201	9\$:	CMP	R1,#201		
000324	001007		BNE	10\$			
000326	013746	000020G	MOV	EMSG.STRUCT+20,-(SP)	:		2050
000332	012746	000001	MOV	#1,-(SP)			
000336	010600		MOV	SP,R0	:	SP,*	
000340	104417		TRAP	17			
000342	000552		BR	28\$:		1946
000344	020127	000301	10\$:	CMP	R1,#301		
000350	001007		BNE	11\$			
000352	013746	000022G	MOV	EMSG.STRUCT+22,-(SP)	:		2058
000356	012746	000001	MOV	#1,-(SP)			
000362	010600		MOV	SP,R0	:	SP,*	
000364	104417		TRAP	17			
000366	000540		BR	28\$:		1946
000370	020127	000401	11\$:	CMP	R1,#401		
000374	001007		BNE	13\$			
000376	013746	000030G	MOV	EMSG.STRUCT+30,-(SP)	:		2066
000402	012746	000001	MOV	#1,-(SP)			
000406	010600		MOV	SP,R0	:	SP,*	
000410	104417		TRAP	17			
000412	000526		BR	28\$:		1946
000414	020127	000501	12\$:	CMP	R1,#501		
000420	001007		BNE	15\$			
000422	013746	000032G	MOV	EMSG.STRUCT+32,-(SP)	:		2074
000426	012746	000001	MOV	#1,-(SP)			
000432	010600		MOV	SP,R0	:	SP,*	
000434	104417		TRAP	17			
000436	000514		BR	28\$:		1946
000440	020127	000601	13\$:	CMP	R1,#601		
000444	001007		BNE	16\$			

ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (10)

000446	013746	000034G		MOV	EMSG.STRUCT+34,-(SP)	:	2082
000452	012746	000001		MOV	#1,-(SP)	:	
000456	010600			MOV	SP,R0	: SP,*	
000460	104417			TRAP	17	:	
000462	000502			BR	28\$:	1946
000464	020127	000701	16\$:	CMP	R1,#701	:	
000470	001007			BNE	18\$:	
000472	013746	000036G		MOV	EMSG.STRUCT+36,-(SP)	:	2090
000476	012746	000001		MOV	#1,-(SP)	:	
000502	010600			MOV	SP,R0	: SP,*	
000504	104417			TRAP	17	:	
000506	000470		17\$:	BR	28\$:	1946
000510	020127	001001	18\$:	CMP	R1,#1001	:	
000514	001007			BNE	20\$:	
000516	013746	000040G		MOV	EMSG.STRUCT+40,-(SP)	:	2098
000522	012746	000001		MOV	#1,-(SP)	:	
000526	010600			MOV	SP,R0	: SP,*	
000530	104417			TRAP	17	:	
000532	000456		19\$:	BR	28\$:	1946
000534	020127	004001	20\$:	CMP	R1,#4001	:	
000540	001007			BNE	22\$:	
000542	013746	000046G		MOV	EMSG.STRUCT+46,-(SP)	:	2106
000546	012746	000001		MOV	#1,-(SP)	:	
000552	010600			MOV	SP,R0	: SP,*	
000554	104417			TRAP	17	:	
000556	000444		21\$:	BR	28\$:	1946
000560	020127	002001	22\$:	CMP	R1,#2001	:	
000564	001007			BNE	24\$:	
000566	013746	000042G		MOV	EMSG.STRUCT+42,-(SP)	:	2114
000572	012746	000001		MOV	#1,-(SP)	:	
000576	010600			MOV	SP,R0	: SP,*	
000600	104417			TRAP	17	:	
000602	000432		23\$:	BR	28\$:	1946
000604	020127	003001	24\$:	CMP	R1,#3001	:	
000610	001007			BNE	26\$:	
000612	013746	000044G		MOV	EMSG.STRUCT+44,-(SP)	:	2122
000616	012746	000001		MOV	#1,-(SP)	:	
000622	010600			MOV	SP,R0	: SP,*	
000624	104417			TRAP	17	:	
000626	000420		25\$:	BR	28\$:	1946
000630	020127	005001	26\$:	CMP	R1,#5001	:	
000634	001007			BNE	27\$:	
000636	013746	000050G		MOV	EMSG.STRUCT+50,-(SP)	:	2130
000642	012746	000001		MOV	#1,-(SP)	:	
000646	010600			MOV	SP,R0	: SP,*	
000650	104417			TRAP	17	:	
000652	000406			BR	28\$:	1946
000654	013746	000006G	27\$:	MOV	EMSG.STRUCT+6,-(SP)	:	2139
000660	012746	000001		MOV	#1,-(SP)	:	
000664	010600			MOV	SP,R0	: SP,*	
000666	104417			TRAP	17	:	
000670	017766	000000G 000004	28\$:	MOV	@RDRX.ADDR,4(SP)	: *,RCSS.REG	2148
000676	012700	177777		MOV	#-1,R0	: *,RC\$M.REG	

ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (10)

000702 010077 000000G
000706 104444
000710 062706 000010
000714 012601
000716 000207

MOV R0, @RDRX.ADDR
TRAP 44
ADD #10, SP
MOV (SP)+, R1
RTS PC

; RCSM.REG, *

;

1885

: Routine Size: 232 words, Routine Base: \$CODES + 0232
: Maximum stack depth per invocation: 7 words

: 2152

2153 global routine DUP\$SERVICE : INT_LNK\$TYP novalue = !Signals receive queue entry

2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205

!++

Functional Description :

The transmission of a message will result in a host interrupt if and only if interrupts were armed suitably during initialization and one of the following conditions has been met:

1. The message was a command with F=1 and the port's fetching it caused the command ring to transition from full to not-full. (This interrupt means that the host may place another command in the ring.)
2. The message was a response with F=1 and the port's depositing it caused the response ring to transition from empty to not-empty. (This interrupt means that there is a response for the host to process.)
3. The port is interfaced to the host via a bus adapter and a command requires the port/controller to re-access a given location during data transfer. (This interrupt means that the port/controller is requesting the host to purge the indicated channel of the bus adapter.)

This interrupt service routine is entered when any of the above conditions occur. When entered it will be determined what type interrupt was executed and take the necessary action.

Formal Parameters :

none

Implicit Inputs :

Nrd_slot:

A global flag which points to the next receive descriptor slot where the port/controller should be polling on and where to expect the first response packet to process.

Implicit Outputs :

Ret_status:

This global flag is the mechanism by which these DUP and UQ Port protocol routines pass status code back to the host routine's requesting communications over the established connections. The status returned is decoded by the caller to determine if an error or bad response packet status was discovered.

Out\$std_buf:

This buffer is used to save all commands issued to the port and are considered outstanding when in this buffer. This interrupt service routine will indicate this command is no longer outstanding by setting the rec_flg in the slot matching this response envelope command ref number.

Completion Codes :

none

```
2206 | Side Effects :
2207 |         none
2208 | --
2209
2210 | begin
2211
2212 | local
2213 |     TEMP,                !Holds nrd_slot + 1 reference
2214 |     FOUND_CMD,          !Found command flag
2215 |     REF_NUM;            !Stores response packets cmd ref number
2216
2217 | +
2218 | | Before this interrupt service routine does anything
2219 | | look at the SA register for any port fatal errors
2220 | | posted. If there are errors posted then report the
2221 | | error and kick the bucket.
2222 | -
2223
2224 | if .RDRX_ADDR [RCSA, ERR_BIT]      !Are there any errors posted
2225 | then
2226 |     begin
2227 |         RET_STATUS = PFE_CODE;    !Indicate the error type
2228 |         DECODE ();                !Decode and print the error
2229 |         return;                   !Just for show. Decode will kill it
2230 |     end;
2231
2232 | +
2233 | | See what kind of interrupt got us here.
2234 | | We could have a:
2235 | |
2236 | | 1. Response ring transition interrupt.
2237 | | 2. Send ring transition interrupt.
2238 | | 3. A adaptor purge request interrupt (which is
2239 | |    illegal running under the PDP-11 Diagnostic
2240 | |    supervisor and is flagged as a fatal controller
2241 | |    error.)
2242 | | 4. Or an unknown interrupt not known by this program
2243 | |    which also results in a fatal controller error.
2244 | |
2245 | | -
2246 | |
2247 | |
2248 | |
2249 | |
2250 | | Check to see if we get here because of a response ring
2251 | | transition interrupt. This is more likely to be the
2252 | | most frequent interrupt so check it first.
2253 | |
2254 | |
2255 | if .HEAD_AREA [RSP_INT]
2256 | then
2257 |     begin
2258 |         HEAD_AREA [RSP_INT] = ZEROS;    !Clear the interrupt indicator location
```

```

2259 GET_NRD (); !Get the resp slot location to process
2260
2261 ! Check the host protocol for being out of sequence
2262 ! with the controller by making sure that this slot
2263 ! is owned by the host (meaning that there is a resp
2264 ! envelope at this ring slot to process.
2265
2266
2267 if .RECEIVE_RING [.NRD_SLOT, OWN_BIT] nequ HOST_OWNED !Is this owned by host
2268 then
2269 begin !Host/port is out of sequence
2270 RET_STATUS = PSE_CODE; !Load a "Port sync error" code
2271 DECODE (); !Report the error and kick the bucket
2272 return; !Just for show Decode kills it
2273 end;
2274
2275 !+
2276 ! Per DUP protocol once interrupted due to a response ring
2277 ! interrupt, the host code should process all response packets
2278 ! found in the response ring. This while loop will continue
2279 ! to process the response packets in the response ring until
2280 ! none remain.
2281 !-
2282
2283 while TRUE do !Process all response packets in ring
2284 begin
2285 BREAK; !Look for control c's
2286
2287 ! Load the Reference structure "Ret_en$ad" with the address
2288 ! of this response envelope to process (The minus %o'4' is
2289 ! done to address the first word in the envelope packet
2290 ! and is equal to location "text-4').
2291
2292 RET_EN$AD = (.RECEIVE_RING [.NRD_SLOT, LO_EN$AD]) - %o'4';
2293
2294 !+
2295 ! Test the end packet for its possible three end types.
2296
2297 ! End message opcodes (also called endcodes) are formed by adding the end
2298 ! message flag to the command opcode. For example, a READ commands end
2299 ! message contains the value OP.RED + OP.END in its opcode field. The Invalid
2300 ! command end message contains just the end message flag (i.e., OP.END) in
2301 ! its opcode field. The serious exception opcode shown above (i.e. OP.SEX +
2302 ! OP.END) in its opcode field.
2303
2304 ! Commands opcode bits 6 and 7 indicate the type of message (command, end or
2305 ! attention message. Command opcodes bits 3 through 5 indicate the command
2306 ! category (immediate, sequential or no-sequential) and whether or not the
2307 ! command includes a buffer descriptor.
2308
2309 ! See MSCP document appendix "A-1 NOTE:" for more information on this topic.
2310 !-
2311

```

```
2312 selectoneu .RET_EN$AD [TYP$MSG] of !Select the endpacket size
2313 set
2314
2315 [END$MSG] : !Is this an end code packet
2316 begin
2317
2318 |
2319 | +
2320 | Select off of the endcode to make sure the communications
2321 | mechanism transfered the correct number of byte for this
2322 | end packet. If this number of bytes transfered is not
2323 | correct for the commands end packet then load the error
2324 | code into return status, call decode to report the
2325 | error and kick the bucket and die. This endcode is formed
2326 | by adding the end message flag %'200' to the commands
2327 | opcode.
2328 | -
2329
2330 selectoneu .RET_EN$AD [ENDCODE] of
2331 set
2332 | "RECEIVE DATA" command end packet
2333 |
2334
2335 [EOP_RED] :
2336 begin
2337
2338 if .RET_EN$AD [MSG_LENGTH] nequ ESZ_RED !Is the byte count correct
2339 then
2340 begin
2341 RET_STATUS = MLE_CODE; !Return a 'message length error code'
2342 DECODE (); !Report the error and kick the bucket
2343 return; !Just for show. Decode kills it
2344 end;
2345
2346 end;
2347
2348 | "SEND DATA" command end packet
2349 | "SET CONTROLLER CHAR" command end packet (same opcode)
2350 |
2351
2352 [EOP_SED] :
2353 ! [EOP_SCC] :
2354 begin
2355
2356 if (not ((.RET_EN$AD [MSG_LENGTH] eqlu ESZ_SED) or !Is the byte count correct
2357 (.RET_EN$AD [MSG_LENGTH] eqlu ESZ_SCC)) )
2358 then
2359 begin
2360 RET_STATUS = MLE_CODE; !Return a 'message length error code'
2361 DECODE (); !Report the error and kick the bucket
2362 return; !Just for show. Decode kills it
2363 end;
2364
```

```
2365         end;
2366         |
2367         | "GET DUST STATUS" command end packet
2368         |
2369         |
2370         [EOP_GDS] :
2371         begin
2372         if .RET_EN$AD [MSG_LENGTH] nequ ESZ_GDS      !Is the byte count correct
2373         then
2374         begin
2375         RET_STATUS = MLE_CODE; !Return a 'message length error code'
2376         DECODE ();           !Report the error and kick the bucket
2377         return;              !Just for show. Decode kills it
2378         end;
2379         end;
2380         |
2381         |
2382         | %  
C         | "EXECUTE SUPPLIED PROGRAM" command end packet
2383         |
2384         |
2385         |
2386         [EOP_ESP] :
2387         begin
2388         if .RET_EN$AD [MSG_LENGTH] nequ ESZ_ESP      !Is the byte count correct
2389         then
2390         begin
2391         RET_STATUS = MLE_CODE; !Return a 'message length error code'
2392         DECODE ();           !Report the error and kick the bucket
2393         return;              !Just for show. Decode kills it
2394         end;
2395         end;
2396         |
2397         |
2398         | )%  
C         | "EXECUTE LOCAL PROGRAM" command end packet
2399         |
2400         |
2401         |
2402         [EOP_ELP] :
2403         begin
2404         if .RET_EN$AD [MSG_LENGTH] nequ ESZ_ELP      !Is the byte count correct
2405         then
2406         begin
2407         RET_STATUS = MLE_CODE; !Return a 'message length error code'
2408         DECODE ();           !Report the error and kick the bucket
2409         return;              !Just for show. Decode kills it
2410         end;
2411         end;
2412         |
2413         |
2414         |
2415         | "ABORT PROGRAM" command end packet
2416         |
2417         |
```

```
2418 [EOP_ABT] :
2419   begin
2420
2421   if .RET_EN$AD [MSG_LENGTH] nequ ESZ_ABT    !Is the byte count correct
2422   then
2423     begin
2424       RET_STATUS = MLE_CODE; !Return a 'message length error code'
2425       DECODE ();    !Report the error and kick the bucket
2426       return;      !Just for show. Decode kills it
2427     end;
2428
2429   end;
2430   %(  
C 2431   ! 'ON LINE' command end packet
C 2432   !
C 2433   !
C 2434   !
C 2435   !
C 2436   !
C 2437   !
C 2438   !
C 2439   !
C 2440   !
C 2441   !
C 2442   !
C 2443   !
C 2444   !
C 2445   !
C 2446   ! )%
2447   ! The 'OP_END' end message flag all by its self tells
2448   ! us that the controller is flagging us of an illegal
2449   ! command sent over the connection. Error and kick the
2450   ! bucket.
2451   !
2452   !
2453   [OP_END] :
2454     begin
2455       RET_STATUS = IVC_CODE;
2456       DECODE ();
2457       return;
2458     end;
2459   !
2460   ! The controller is telling us that a serious exception
2461   ! has occured. Error and kick the bucket.
2462   !
2463   !
2464   [EOP_SEX] :
2465     begin
2466       RET_STATUS = SEX_CODE;
2467       DECODE ();
2468       return;
2469     end;
2470   !
```

```
2471      ! Unknown end packet endcode type
2472      !
2473
2474      [otherwise] :
2475      begin
2476      RET_STATUS = UEC_CODE;      !Return an 'unknown end code'
2477      DECODE ();      !Report the error and kick the bucket
2478      return;      !Just for show. Decode kills it
2479      end;
2480      tes;
2481
2482      !
2483      ! The port/controller sent the endpacket over the connection
2484      ! with out any problems. Now find this commands owner in the
2485      ! out$standing buffer and indicate to them that the command
2486      ! has been received.
2487
2488      REF_NUM = .RET_EN$AD [CMD_LREF];      !Get this rec packets cmd ref number
2489
2490      ! Search the outstanding command buffer for this commands
2491      ! reference number.
2492
2493      ! If found, load the buffer location with the ret_en$ad
2494      ! and set the received flag to signify that this command
2495      ! has been received by this interrupt service routine.
2496
2497      FOUND_CMD = FALSE;      !Clear the found cmd flag
2498
2499      incru i from 0 to REC_ALLOCATE - 1 do      !Search the buffer
2500
2501      if .OUT$STD_BUF [.i, CMD_REF] eqlu .REF_NUM      !Is this the cmd ref
2502      then
2503      begin
2504      OUT$STD_BUF [.i, REC_FLG] = TRUE;      !Indicate command is received
2505      OUT$STD_BUF [.i, ENV_ADR] = .RET_EN$AD;      !Return envelope adrs
2506      FOUND_CMD = TRUE;      !Indicate it was found
2507      exitloop;      !Exit the loop
2508      end;
2509
2510      !
2511      ! If the search through the command ref
2512      ! buffer failed to find this commands cmd
2513      ! reference number then die.
2514
2515
2516      if not .FOUND_CMD
2517      then
2518      begin
2519      RET_STATUS = PSE_CODE;
2520      DECODE ();
2521      return;
2522      end;
2523
```



```
2524      end;                                !End of ENDSMSG processing
2525      |
2526      | The set controller characteristics command
2527      | disabled the reporting of attentions messages
2528      | so treat this as a fatal error and die.
2529      |
2530      |
2531      [ATT$MSG] :                            !Attention end message type
2532      begin
2533      RET STATUS = ATN_CODE;
2534      DECODE ();
2535      return;
2536      end;
2537      |
2538      | It doesn't make any since for this end message
2539      | packet not to have the end message flag added
2540      | to this command opcode as treat it as a fatal
2541      | error and die.
2542      |
2543      |
2544      [CMD$MSG] :
2545      begin
2546      RET STATUS = CMD_CODE;
2547      DECODE ();
2548      return;
2549      end;
2550      |
2551      | This end code type is of unknown origin so
2552      | treat is as a fatal error and die.
2553      |
2554      |
2555      [otherwise] :
2556      begin
2557      RET STATUS = UEC_CODE;                !Unknown message type code received
2558      DECODE ();
2559      return;
2560      end;
2561      tes;
2562      |
2563      |
2564      | Before we leave put this receive envelope message length
2565      | field back to the envelope size, in bytes (Per UQ Spec).
2566      | This size does not include the 2 UQ's words preceeding the
2567      | command text area.
2568      |
2569      RET_EN$AD [MSG_LENGTH] = RB_SIZE*2;
2570      |
2571      | Return this receive slot descriptor back to
2572      | the port to fullfill my part of the protocol.
2573      |
2574      RECEIVE_RING [.NRD_SLOT, OWN_BIT] = PORT_OWNED;
2575      |
2576      | Look at the next response ring descriptor. If its
```

```
2577      ! host owned then continue this process else exit the
2578      ! loop. First see if the ring reference has wrapped
2579      ! around to the top of the ring.
2580
2581
2582      if .NRD_SLOT + 1 gtru REC_ALLOCATE - 1      !Has the ring ref wrapped around
2583      then
2584          TEMP = ZERO      !Wrap it back to zero dsc slot
2585      else
2586          TEMP = .NRD_SLOT + 1;      !Look at the next seq dsc slot
2587
2588
2589      ! Now see if the next receive descriptor slot is
2590      ! host owned.
2591
2592
2593      if .RECEIVE_RING [.TEMP, OWN_BIT] eqlu HOST_OWNED      !Are we done yet
2594      then
2595          GET_NRD ( )      !Get the next resp desc to process
2596      else
2597          exitloop;      !No more to do so exit
2598
2599      end;      !End of WHILE LOOP
2600
2601
2602      ! All response ring descriptors have been with out any
2603      ! detected errors so return to the main host code with
2604      ! an pass return code.
2605
2606      return RET_STATUS = PAS_CODE;      !Return a 'pass code
2607      end;
2608
2609      !*****END OF RESPONSE RING INTERRUPT PROCESSING*****
2610
2611      !+
2612      ! A send ring transition interrupt could happen if at
2613      ! some date only one descriptor slot is allocated for
2614      ! commands.
2615
2616      ! Clear the interrupt indicator if this is true and do
2617      ! a return with no errors.
2618      !-
2619
2620      if .HEAD_AREA [CMD_INT]      !Is this a com ring transition interrupt
2621      then
2622          begin
2623              HEAD_AREA [CMD_INT] = ZERO;      !Clear out the indicator
2624              return;      !Continue on with the host code
2625          end;
2626
2627      !*****END OF COMMAND RING INTERRUPT PROCESSING*****
2628
2629      !+
```

ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (11)

```

2630      ! Check to see if an adaptor purge is being requested
2631      ! by the port/controller in order to complete excution
2632      ! of a issued command. Remember that this is illegal
2633      ! during PDP-11 formatting and is concidered to be a
2634      ! fatal error.
2635      !-
2636
2637      if .HEAD_AREA [ADP_CH] nequ ZERO          !Is the an adaptor purge request?
2638      then
2639      begin
2640      RET_STATUS = APR_CODE;                    !Indicate the error code
2641      DECODE ();                               !Report the error and kick the bucket
2642      return;                                  !Just for show. Decode kills it
2643      end;
2644
2645      !*****END OF ADAPTOR PURGE INTERRUPT PROCESSING*****
2646
2647      !+
2648      ! The host program has been interrupted by an unknown interrupt
2649      ! source if the routine program flow reaches here.
2650
2651      ! Load the error code into return status and call decode to take
2652      ! appropriate action.
2653      !-
2654
2655      RET_STATUS = UIN_CODE;
2656      DECODE ();
2657      return;
2658      end;

```

```

000000 010046      .SBTTL DUPSI.SERVICE GLOBAL ROUTINE DECLARATIONS
000002 010146      DUPSI.SERVICE::
000004 010246      MOV      R0,-(SP)      ;
000006 010346      MOV      R1,-(SP)      ;
000010 010446      MOV      R2,-(SP)      ;
000012 010546      MOV      R3,-(SP)      ;
000014 013700 000000G  MOV      R4,-(SP)      ;
000020 016046 000002   MOV      R5,-(SP)      ;
000024 100004      BPL      1$            ;
000026 012737 000021 000000G  MOV      RDRX.ADDR,R0  ;
000034 000553      BR       13$          ;
000036 013700 000000G 1$:  MOV      2(R0),-(SP)  ;
000042 032760 000001 000006   BPL      #21,RET.STATUS ;
000050 001002      BIT      #1,6(R0)    ;
000052 000137 001776'   BNE      2$            ;
000056 005060 000006'   JMP      25$          ;
000062 004737 000026'   CLR      6(R0)        ;
000066 013700 000000G  JSR      PC,GET.NRD   ;
000072 006300      MOV      NRD.SLOT,R0 ;
000074 006300      ASL      R0          ;

```

ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (11)

000076	063700	000000G		ADD	RECEIVE.RING,RO		
000102	032760	100000	000002	BIT	#100000,2(RO)		
000110	001404			BEQ	4\$		
000112	012737	000061	000000G	3\$: MOV	#61,RET.STATUS	:	2270
000120	000573			BR	20\$:	2271
000122	104422			4\$: TRAP	22	:	2284
000124	013700	000000G		MOV	NRD.SLOT,RO	:	2292
000130	006300			ASL	RO		
000132	006300			ASL	RO		
000134	063700	000000G		ADD	RECEIVE.RING,RO		
000140	011037	000000G		MOV	(RO),RET.ENSAD		
000144	162737	000004	000000G	SUB	#4,RET.ENSAD		
000152	013701	000000G		MOV	RET.ENSAD,R1	:	2312
000156	116100	000014		MOV#	14(R1),RO		
000162	006200			ASR	RO		
000164	006200			ASR	RO		
000166	006200			ASR	RO		
000170	006200			ASR	RO		
000172	006200			ASR	RO		
000174	006200			ASR	RO		
000176	042700	177774		BIC	#177774,RO		
000202	020027	000002		CMP	RO,#2		
000206	001124			BNE	18\$		
000210	005002			CLR	R2	:	2329
000212	156102	000014		BISB	14(R1),R2		
000216	020227	000205		CMP	R2,#205		
000222	001007			BNE	7\$		
000224	021127	000020		5\$: CMP	(R1),#20	:	2338
000230	001456			6\$: BEQ	14\$:	2341
000232	012737	000071	000000G	MOV	#71,RET.STATUS	:	2342
000240	000523			BR	20\$:	2329
000242	020227	000204		7\$: CMP	R2,#204	:	
000246	001006			BNE	8\$:	2356
000250	021127	000020		CMP	(R1),#20	:	2357
000254	001444			BEQ	14\$:	2329
000256	021127	000034		CMP	(R1),#34	:	2373
000262	000762			BR	6\$:	2329
000264	020227	000201		8\$: CMP	R2,#201	:	2405
000270	001003			BNE	9\$:	2329
000272	021127	000026		CMP	(R1),#26	:	2421
000276	000754			BR	6\$:	2329
000300	020227	000203		9\$: CMP	R2,#203	:	2405
000304	001747			BEQ	5\$:	2329
000306	020227	000206		CMP	R2,#206	:	
000312	001003			BNE	10\$:	2421
000314	021127	000014		CMP	(R1),#14	:	2329
000320	000743			BR	6\$:	
000322	020227	000200		10\$: CMP	R2,#200	:	2455
000326	001004			BNE	11\$:	2456
000330	012737	000701	000000G	MOV	#701,RET.STATUS	:	2329
000336	000557			BR	28\$:	
000340	020227	000207		11\$: CMP	R2,#207	:	
000344	001004			BNE	12\$:	

ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

000346	012737	000601	000000G		MOV	#601,RET.STATUS	:	2466
000354	000550				BR	28\$:	2467
000356	012737	000101	000000G	12\$:	MOV	#101,RET.STATUS	:	2476
000364	000544			13\$:	BR	28\$:	2477
000366	013700	000000G		14\$:	MOV	RET.ENSAD,R0	:	2488
000372	016005	000004			MOV	4(R0),R5	:	
000376	005004				CLR	R4	:	2497
000400	005000				CLR	R0	:	2499
000402	005001			15\$:	CLR	R1	:	2501
000404	156001	000000G			BISB	OUT\$STD.BUF(R0),R1	:	
000410	020105				CMP	R1,R5	:	
000412	001011				BNE	16\$:	
000414	052760	100000	000000G		BIS	#100000,OUT\$STD.BUF(R0)	:	2504
000422	013760	000000G	000002G		MOV	RET.ENSAD,OUT\$STD.BUF+2(R0)	:	2505
000430	012704	000001			MOV	#1,R4	:	2506
000434	000405				BR	17\$:	2503
000436	062700	000004		16\$:	ADD	#4,R0	:	2499
000442	020027	000014			CMP	R0,#14	:	
000446	101755				BLOS	15\$:	
000450	032704	000001		17\$:	BIT	#1,R4	:	2516
000454	001016				BNE	21\$:	
000456	000615				BR	3\$:	2519
000460	020027	000001		18\$:	CMP	R0,#1	:	2312
000464	001004				BNE	19\$:	
000466	012737	000401	000000G		MOV	#401,RET.STATUS	:	2533
000474	000500				BR	28\$:	2534
000476	005700			19\$:	TST	R0	:	2312
000500	001326				BNE	12\$:	
000502	012737	000501	000000G		MOV	#501,RET.STATUS	:	2546
000510	000472			20\$:	BR	28\$:	2547
000512	012777	000074	000000G	21\$:	MOV	#74,@RET.ENSAD	:	2569
000520	013700	000000G			MOV	NRD.SLOT,R0	:	2574
000524	006300				ASL	R0	:	
000526	006300				ASL	R0	:	
000530	063700	000000G			ADD	RECEIVE.RING,R0	:	
000534	052760	100000	000002		BIS	#100000,2(R0)	:	
000542	013700	000000G			MOV	NRD.SLOT,R0	:	2582
000546	005200				INC	R0	:	
000550	020027	000003			CMP	R0,#3	:	
000554	101402				BLOS	22\$:	
000556	005003				CLR	R3	:	2584
000560	000401				BR	23\$:	2582
000562	010003			22\$:	MOV	R0,R3	:	2586
000564	010300			23\$:	MOV	R3,R0	:	2593
000566	006300				ASL	R0	:	
000570	006300				ASL	R0	:	
000572	063700	000000G			ADD	RECEIVE.RING,R0	:	
000576	032760	100000	000002		BIT	#100000,2(R0)	:	
000604	001004				BNE	24\$:	
000606	004737	000026'			JSR	PC,GET.NRD	:	2595
000612	000137	001274'			JMP	4\$:	2593
000616	005037	000000G		24\$:	CLR	RET.STATUS	:	2606
000622	000427				BR	29\$:	2257

ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (11)

000624	013700	000000G		25\$:	MOV	HEAD.AREA,R0	:	2620
000630	032760	000001 000004			BIT	#1,4(R0)		
000636	001403				BEQ	26\$		
000640	005060	000004			CLR	4(R0)	:	2623
000644	000416				BR	29\$:	2622
000646	013700	000000G		26\$:	MOV	HEAD.AREA,R0	:	2637
000652	105760	000003			TSTB	3(R0)		
000656	001404				BEQ	27\$		
000660	012737	000201 000000G			MOV	#201,RET.STATUS	:	2640
000666	000403				BR	28\$:	2641
000670	012737	000301 000000G		27\$:	MOV	#301,RET.STATUS	:	2655
000676	004737	000232'		28\$:	JSR	PC,DECODE	:	2656
000702	005726			29\$:	TST	(SP)+	:	2153
000704	012605				MOV	(SP)+,R5		
000706	012604				MOV	(SP)+,R4		
000710	012603				MOV	(SP)+,R3		
000712	012602				MOV	(SP)+,R2		
000714	012601				MOV	(SP)+,R1		
000716	012600				MOV	(SP)+,R0		
000720	000002				RTI			

: Routine Size: 233 words, Routine Base: \$CODE\$ + 1152
: Maximum stack depth per invocation: 13 words

: 2659

2660 global routine CTO_WAIT (TO_VALUE, REF_NUM, BUF\$LOC) = !Controller time out wait

2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712

!++
Functional Description :
This routine is called to wait for the port/controller to either complete the queued command or time out the command.

Formal Parameters :
TO_VALUE Indicate the time-out interval for this command.
REF_NUM This argument contains the unique reference number assigned to this command being timed out by this routine.
BUF\$LOC This argument points to the out\$std_buf location where this command is saved. At this location the received flag 'rec_flg' bit is examined within the timeout loop and when it equals true will signal that this command has been received by the interrupt service routine.

Implicit Inputs :
none

Implicit Outputs :
The command word in the out\$std_buffer 'word zero of a command slot' is cleared out with the value %o'100000' to indicate this is an unused out\$std_buffer slot and that it can be reused.

Completion Codes :
There are two levels of return status returned by this routine.
1. The DUP interrupt service returns to this routine a status code to indicate the success of the connection/communications mechanism to complete the queued command. If the port/controller does not time out then this return status is returned as this routines return status code.
2. If the port/controller times out then the SA Register error bit is examined for the error bit set. If set then an port fatal error code is returned to the calling routine else a controller time out error code is returned.

In all cases, if an error code is returned (bit 0 = 1) then the routine decode is called to decode the error code and does the necessary recovery actions.

At the next higher level of return from this routine is another level of return status returned. This level test the success of the connection and also test the status field in the returned response envelope for the success of the controller to successfully complete the requested command.

Side Effects :

```
2713 | none
2714 |--
2715 |
2716 | begin
2717 |
2718 | +
2719 | | Before doing the timeout wait make sure that this buffer location
2720 | | that we're suppose to time out actually contains the command ref
2721 | | number that was sent to us via the formal argument. Error and
2722 | | kick the bucket if not the same.
2723 | | -
2724 | |
2725 | | if .OUT$STD_BUF [.BUF$LOC, CMD_REF] nequ .REF_NUM !Is this the same ref_num
2726 | | then
2727 | | | begin
2728 | | | RET_STATUS = OSE_CODE; !Indicate the error code
2729 | | | DECODE (); !Call decode to report the error
2730 | | | end;
2731 | |
2732 | | +
2733 | | | Loop on a one micro second delay for the number of times
2734 | | | requested by the caller. After each delay see if the flag
2735 | | | 'rec_flg' has been set yet. Return 'Ret_status' and clear the
2736 | | | command word to %o'100000' to indicate this command has been
2737 | | | received if this flag gets set before the timer expires.
2738 | | | Return a error code if the timer expires before the flag gets set.
2739 | | | -
2740 | | |
2741 | | | incru i from 0 to .TO_VALUE do !Loop for time-out_value
2742 | | | | begin
2743 | | | | DELAY (C_US); !Do the one micro second delay
2744 | | | | |
2745 | | | | | Exit routine with the DUP interrupt service
2746 | | | | | routines 'ret_status' if 'rec_flg' got set before
2747 | | | | | the timer expires.
2748 | | | | |
2749 | | | | |
2750 | | | | if .OUT$STD_BUF [.BUF$LOC, REC_FLG] eqlu TRUE !Is this command received yet
2751 | | | | then
2752 | | | | | begin
2753 | | | | | OUT$STD_BUF [.BUF$LOC, CMD_WRD] = %o'100000'; !Return the slot to the unused state
2754 | | | | | return .RET_STATUS; !Return the interrupt service status
2755 | | | | | end;
2756 | | | |
2757 | | | | BREAK; !Service any control C's
2758 | | | | end;
2759 | | | |
2760 | | | |
2761 | | | | | The port/controller timed out if the code
2762 | | | | | reached here. Return an error code to
2763 | | | | | the caller and exit the routine.
2764 | | | | |
2765 | | | | |
```


ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

K 10

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (12)

SEQ 127
Page 40

```

:      2766      if .RDRX_ADDR [RCSA, ERR_BIT]
:      2767      then
:      2768          return RET_STATUS = PFE_CODE
:      2769      else
:      2770          return RET_STATUS = CTO_CODE;
:      2771
:      2772      end;

```

```

!Is the SA error bit set
!Port timed out with fatal error
!Port just timed out

```

.GLOBL LSDLY

			.SBTTL	CTO.WAIT GLOBAL ROUTINE DECLARATIONS	
000000	004137	000000G	CTO.WAIT::	JSR R1,\$SAVE3	2660
000004	024646			CMP -(SP),-(SP)	
000006	016600	000016		MOV 16(SP),R0	2725
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	2728
000044	004737	000232		JSR PC,DECODE	2729
000050	005002		1\$:	CLR R2	2741
000052	000425			BR 8\$	
000054	012701	000001	2\$:	MOV #1,R1	2743
000060	001411		3\$:	BEQ 6\$	
000062	013700	000000G		MOV LSDLY,R0	: *,\$STMP1
000066	001404			BEQ 5\$	
000070	005066	000002	4\$:	CLR 2(SP)	: \$STMP
000074	005300			DEC R0	: \$STMP1
000076	001374			BNE 4\$	
000100	005301		5\$:	DEC R1	: \$STMP2
000102	000766			BR 3\$	
000104	005713		6\$:	TST (R3)	2750
000106	100005			BPL 7\$	
000110	012713	100000		MOV #-100000,(R3)	2753
000114	013700	000000G		MOV RET.STATUS,R0	2752
000120	000421			BR 11\$	
000122	104422		7\$:	TRAP 22	2755
000124	005202			INC R2	2741
000126	020266	000022	8\$:	CMP R2,22(SP)	: I,TO.VALUE
000132	101750			BLOS 2\$	
000134	013700	000000G		MOV RDRX.ADDR,R0	2766
000140	016016	000002		MOV 2(R0),(SP)	: *,RCSS.REG
000144	100003			BPL 9\$	
000146	012700	000021		MOV #21,R0	2768
000152	000402			BR 10\$	
000154	012700	000011	9\$:	MOV #11,R0	2770

ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (12)

000160	010037	000000G	10\$:	MOV	RO,RET,STATUS		
000164	022626		11\$:	CMP	(SP)+,(SP)+	:	2660
000166	000207			RTS	PC		

: Routine Size: 60 words, Routine Base: \$CODE\$ + 2074
 : Maximum stack depth per invocation: 8 words

: 2773

```
2774 global routine ABORT =          !Aborts remote program
2775
2776 !++
2777 Functional Description :
2778 The abort program command is used to terminate the execution of a
2779 remote program in an orderly fashion. When a successful response
2780 is received to this command the remote program has stopped
2781 executing and the server is in idle state. Note that the sending
2782 of this command does not preclude further send data or receive data
2783 exchanges: On the contrary, the remote program may be designed to
2784 send out termination status and possibly even ask questions during
2785 its forced-exit sequence. The time out for this command is a fixed
2786 10 seconds and if a response is not received by then the
2787 connection to the dust should be terminated. This command is only
2788 legal if the dust is in active state.
2789
2790 Formal Parameters :
2791 none
2792
2793 Implicit Inputs :
2794 NSD_SLOT          This global storage gets loaded by the routine
2795                   'Get_nsd' and in it is stored the next send ring
2796                   descriptor slot where the port/controller should
2797                   be polling on and the place to put this commands
2798                   command packet.
2799
2800 Implicit Outputs :
2801 none
2802
2803 Completion Codes :
2804 RET_STATUS:      Return status passes back to the calling routine
2805                  the status of the just issued command.
2806
2807 Side Effects :
2808 Any remote program running in the controllers DM machine will
2809 be aborted.
2810 --
2811
2812 begin
2813
2814 local
2815     REF_NUM,          !Stores unique cmd ref number
2816     ABO_BUF$LOC,     !Stores outstanding cmd buffer location
2817     TEMP;            !A place to put the read IP register data
2818
2819 !
2820 ! Before we load up the command packet up
2821 ! with all this good information get the
2822 ! next send descriptor slot and a unique
2823 ! command reference number.
2824
2825 GET_NSD ();          !Get the next send desc slot
2826 REF_NUM = GET_CMD$REF (); !Get a unique command ref num
```

```
2827 |
2828 | UQ Port command envelope Header field definition
2829 |
2830 | SND_ENVELOPE [.NSD_SLOT, MSG_LENGTH] = SZ_ABT; !Load the length of envelope
2831 | SND_ENVELOPE [.NSD_SLOT, CREDITS] = ONE; !Load credits
2832 | SND_ENVELOPE [.NSD_SLOT, MSG_TYPE] = 0; !Load message type
2833 | SND_ENVELOPE [.NSD_SLOT, CONN_ID] = 2; !Load connection ID
2834 |
2835 | DUP command envelope field definition
2836 |
2837 | SND_ENVELOPE [.NSD_SLOT, CMD_LREF] = .REF_NUM; !Define reference number
2838 | SND_ENVELOPE [.NSD_SLOT, CMD_HREF] = ZERO; !Hi order ref number
2839 | SND_ENVELOPE [.NSD_SLOT, UN_USED] = ZERO; !Unused low order
2840 | SND_ENVELOPE [.NSD_SLOT, UN_HUSED] = ZERO; !Unused hi order
2841 | SND_ENVELOPE [.NSD_SLOT, OPCODE] = OP_ABT; !Load opcod
2842 | SND_ENVELOPE [.NSD_SLOT, RSVD] = ZERO; !Reserved field
2843 | SND_ENVELOPE [.NSD_SLOT, MODIFIER] = ZERO;
2844 |
2845 | Call the load outstanding command buffer routine
2846 | and load this command into the buffer. The return
2847 | from this routine will point us to the buffer location
2848 | where this command is stored. Later we can look at
2849 | this location to see if the interrupt service routine
2850 | has received and process it.
2851 |
2852 | ABO_BUF$LOC = LOAD_OUT$STD_BUF (.REF_NUM); !Load the command
2853 |
2854 | if .ABO_BUF$LOC eqlu OBF_CODE then DECODE (); !Error if buffer is full
2855 |
2856 |
2857 | Set the ownership bit to 1 giving this slot
2858 | to the port/controller
2859 |
2860 | SEND_RING [.NSD_SLOT, OWN_BIT] = PORT_OWNED;
2861 |
2862 | Read the IP register to stimulate port polling
2863 |
2864 | TEMP = .RDRX_ADDR [RCIP, RC_ALL];
2865 |
2866 | Time out the port/controller processing the command.
2867 |
2868 | The first test tests the connections ability to
2869 | respond to this command without any errors in the SA
2870 | register and for the command not timing out.
2871 |
2872 | The second tests the DUP server for good status. If
2873 | bad status is sent back then an error code is returned
2874 | to the calling routine where the routine "decode" will
2875 | decode and take the appropriate recovery. The time
2876 | out routine will loop on delaying and checking the hi
2877 | bit of the first word in the out$std_buf for a true.
2878 | When true signals us that the interrupt service routine
2879 | has received the endpacket and no connection errors
```

```

: 2880      | were detected.
: 2881      |
: 2882      |
: 2883      | if CTO_WAIT (3000, .REF_NUM, .ABO_BUF$LOC) then DECODE (); !Is return an error
: 2884      |
: 2885      |
: 2886      | Get the return envelope address from the out$std_buf
: 2887      | at this commands buffer location and check the packet
: 2888      | for good status error and die if bad status was returned
: 2889      |
: 2890      | RET_EN$AD = .OUT$STD_BUF [.ABO_BUF$LOC, ENV_ADR]; !Get the ret env adr
: 2891      |
: 2892      | Now test for good status
: 2893      |
: 2894      |
: 2895      | if .RET_EN$AD [STATUS] nequ ZERO          !Test the status
: 2896      | then
: 2897      |     return RET_STATUS = RSE_CODE         !Return a 'Response status err'' code
: 2898      | else
: 2899      |     return .RET_STATUS;                 !This ret_status is good or bad
: 2900      |
: 2901      | end;

```

```

000000 004137 000000G      .SBTTL  ABORT GLOBAL ROUTINE DECLARATIONS
000004 005746      ABORT:: JSR  R1,$SAVE2          ; 2774
000006 004737 000000'      TST    -(SP)                  ;
000012 004737 000146'      JSR    PC,GET.NSD             ; 2825
000016 010002      JSR    PC,GET.CMD$REF         ; 2826
000020 013746 000000G      MOV    R0,R2                  ; *,REF.NUM
000024 012746 000054      MOV    NSD.SLOT,-(SP)         ;
000030 004737 000000G      JSR    PC,BLSMUL              ;
000034 012760 000014 000000G  MOV    #14,SND.ENVELOPE(R0)   ;
000042 012701 000002G      MOV    #SND.ENVELOPE+2,R1    ; 2831
000046 060001      ADD    R0,R1                  ;
000050 112711 000001      MOVB   #1,(R1)                ; 2832
000054 112761 000002 000001  MOVB   #2,1(R1)               ; 2833
000062 010260 000004G      MOV    R2,SND.ENVELOPE+4(R0) ; REF.NUM,* 2837
000066 005060 000006G      CLR   SND.ENVELOPE+6(R0)     ; 2838
000072 005060 000010G      CLR   SND.ENVELOPE+10(R0)    ; 2839
000076 005060 000012G      CLR   SND.ENVELOPE+12(R0)    ; 2840
000102 112760 000006 000014G  MOVB   #6,SND.ENVELOPE+14(R0) ; 2841
000110 105060 000015G      CLRB  SND.ENVELOPE+15(R0)    ; 2842
000114 005060 000016G      CLR   SND.ENVELOPE+16(R0)    ; 2843
000120 010216      MOV    R2,(SP)                ; REF.NUM,* 2852
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$LOC,* 2854
000134 001002      BNE   1$                      ;
000136 004737 000232'      JSR    PC,DECODE             ;
000142 013700 000000G      1$:  MOV    NSD.SLOT,R0        ; 2860
000146 006300      ASL    R0                      ;

```

ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (13)

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\$\$,REG	2864
000172	016600	000004	MOV	4(SP),R0	: RC\$\$,REG,TEMP	
000176	012716	005670	MOV	#5670,(SP)	: REF.NUM,*	2883
000202	010246		MOV	R2,-(SP)	: ABO.BUF\$LOC,*	
000204	010146		MOV	R1,-(SP)		
000206	004737	002074'	JSR	PC,CTO.WAIT		
000212	022626		CMP	(SP)+,(SP)+		
000214	006000		ROR	R0		
000216	103002		BCC	2\$		
000220	004737	000232'	JSR	PC,DECODE	: ABO.BUF\$LOC,*	2890
000224	010100		MOV	R1,R0		
000226	006300		ASL	R0		
000230	006300		ASL	R0		
000232	016037	000002G 000000G	MOV	OUT\$STD.BUF+2(R0),RET.ENSAD		2895
000240	016000	000002G	MOV	OUT\$STD.BUF+2(R0),R0	:	
000244	005760	000016	TST	16(R0)		
000250	001405		BEQ	3\$		
000252	012700	000031	MOV	#31,R0	:	2897
000256	010037	000000G	MOV	R0,RET.STATUS	:	2812
000262	000402		BR	4\$:	
000264	013700	000000G	MOV	RET.STATUS,R0	:	2774
000270	062706	000006	ADD	#6,SP		
000274	000207		RTS	PC		

: Routine Size: 95 words, Routine Base: \$CODE\$ + 2264
: Maximum stack depth per invocation: 9 words

: 2902

```

2903 global routine GET_DUST_STATUS = !Gets DUP server status
2904
2905 !++
2906 Functional Description :
2907 This command allows the host program to interrogate the DUP server
2908 to determine its characteristics, its state and the state of the
2909 program currently running (if any). It is legal in either idle or
2910 active state and does not affect the state of server. It has a
2911 fixed timeout interval of 3 seconds. If the response times out, the
2912 host should break the connection.
2913
2914 Formal Parameters :
2915 none
2916
2917 Implicit Inputs :
2918 NSD_SLOT This global storage gets loaded by the routine
2919 'Get_nsd' and in it is stored the next send ring
2920 descriptor slot where the port/controller should
2921 be polling on and the place to put this commands
2922 command packet.
2923
2924 Implicit Outputs :
2925 none
2926
2927 Completion Codes :
2928 RET_STATUS: Return status passes back to the calling routine
2929 the status of the just issued command.
2930
2931 Side Effects :
2932 --
2933
2934 begin
2935
2936 local
2937 REF_NUM, !Stores unique cmd ref number
2938 GDS_BUF$LOC, !Stores outstanding cmd buffer location
2939 TEMP; !A place to put the IP read data
2940
2941 |
2942 | Before we load up the command packet up
2943 | with all this good information get the
2944 | next send descriptor slot and a unique
2945 | command reference number.
2946
2947 GET_NSD (); !Get the next send desc slot
2948 REF_NUM = GET_CMD$REF (); !Get a unique command ref num
2949
2950 | UQ Port command envelope Header field definition
2951
2952 SND_ENVELOPE [.NSD_SLOT, MSG_LENGTH] = SZ_GDS; !Load the envelope size
2953 SND_ENVELOPE [.NSD_SLOT, CREDITS] = ONE; !Load the credit size
2954 SND_ENVELOPE [.NSD_SLOT, MSG_TYPE] = 0; !Load the message type (Sequential)
2955 SND_ENVELOPE [.NSD_SLOT, CONN_ID] = 2; !Load the connection ID (DUP)

```

```
2956 |
2957 | DUP generic command envelope field definition
2958 |
2959 | SND_ENVELOPE [.NSD_SLOT, CMD_LREF] = .REF_NUM;      !Load command reference number
2960 | SND_ENVELOPE [.NSD_SLOT, CMD_HREF] = ZERO;        !Command reference low order
2961 | SND_ENVELOPE [.NSD_SLOT, UN_USED] = ZERO;        !Low order unused
2962 | SND_ENVELOPE [.NSD_SLOT, UN_HUSED] = ZERO;       !Hi order unused
2963 | SND_ENVELOPE [.NSD_SLOT, OPCODE] = OP_GDS;       !Load opcode
2964 | SND_ENVELOPE [.NSD_SLOT, RSVD] = ZERO;           !Reserved field
2965 | SND_ENVELOPE [.NSD_SLOT, MODIFIER] = ZERO;       !Load modifier field
2966 |
2967 | Call the load outstanding command buffer routine
2968 | and load this command into the buffer. The return
2969 | from this routine will point us to the buffer location
2970 | where this command is stored. Later we can look at
2971 | this location to see if the interrupt service routine
2972 | has received and process it.
2973 |
2974 | GDS_BUF$LOC = LOAD_OUT$STD_BUF (.REF_NUM); !Load the command
2975 |
2976 | if .GDS_BUF$LOC eqlu OBF_CODE then DECODE ();      !Error if buffer is full
2977 |
2978 |
2979 | Set the ownership bit to 1 giving this slot
2980 | to the port/controller
2981 |
2982 | SEND_RING [.NSD_SLOT, OWN_BIT] = PORT_OWNED;
2983 |
2984 | Read the IP register to stimulate port polling
2985 |
2986 | TEMP = .RDRX_ADDR [RCIP, RC_ALL];
2987 |
2988 | Time out the port/controller processing the command.
2989 |
2990 | The first test tests the connections ability to
2991 | respond to this command without any errors in the SA
2992 | register and for the command not timing out.
2993 |
2994 | The second tests the DUP server for good status. If
2995 | bad status is sent back then an error code is returned
2996 | to the calling routine where the routine "decode" will
2997 | decode and take the appropriate recovery. The time
2998 | out routine will loop on delaying and checking the hi
2999 | bit of the first word in the out$std_buf for a true.
3000 | When true signals us that the interrupt service routine
3001 | has received the endpacket and no connection errors
3002 | were detected.
3003 |
3004 |
3005 | if CTO_WAIT (%o'3000', .REF_NUM, .GDS_BUF$LOC) then DECODE (); !Is return an error
3006 |
3007 |
3008 | Get the return envelope address from the out$std_buf
```



```

: 3009      | at this commands buffer location and check the packet
: 3010      | for good status error and die if bad status was returned
: 3011
: 3012      RET_EN$AD = .OUT$STD_BUF [.GDS_BUF$LOC, ENV_ADR];      !Get the ret env adr
: 3013
: 3014      | Now test for good status
: 3015
: 3016
: 3017      if .RET_EN$AD [STATUS] nequ ZERO                        !Test the status
: 3018      then
: 3019          return RET_STATUS = RSE_CODE                        !Return a 'Response status err' code
: 3020      else
: 3021          return .RET_STATUS;                                  !This ret_status is good or bad
: 3022
: 3023      end;

```

000000	004137	000000G	.SBTTL GET.DUST.STATUS GLOBAL ROUTINE DECLARATIONS	
			GET.DUST.STATUS::	
			JSR R1,\$SAVE2	2903
000004	005746		TST -(SP)	
000006	004737	000000'	JSR PC,GET.NSD	2947
000012	004737	000146'	JSR PC,GET.CMD\$REF	2948
000016	010002		MOV R0,R2	*.REF.NUM
000020	013746	000000G	MOV NSD.SLOT, -(SP)	2952
000024	012746	000054	MOV #54, -(SP)	
000030	004737	000000G	JSR PC,BLSMUL	
000034	012760	000014 000000G	MOV #14, SND.ENVELOPE(R0)	
000042	012701	000002G	MOV #SND.ENVELOPE+2, R1	2953
000046	060001		ADD R0, R1	
000050	112711	000001	MOVB #1, (R1)	2954
000054	112761	000002 000001	MOVB #2, 1(R1)	2955
000062	010260	000004G	MOV R2, SND.ENVELOPE+4(R0)	REF.NUM, *
000066	005060	000006G	CLR SND.ENVELOPE+6(R0)	2959
000072	005060	000010G	CLR SND.ENVELOPE+10(R0)	2960
000076	005060	000012G	CLR SND.ENVELOPE+12(R0)	2961
000102	112760	000001 000014G	MOVB #1, SND.ENVELOPE+14(R0)	2962
000110	105060	000015G	CLRB SND.ENVELOPE+15(R0)	2963
000114	005060	000016G	CLR SND.ENVELOPE+16(R0)	2964
000120	010216		MOV R2, (SP)	REF.NUM, *
000122	004737	000054'	JSR PC,LOAD.OUT\$STD.BUF	2974
000126	010001		MOV R0, R1	*.GDS.BUF\$LOC
000130	020127	002001	CMP R1, #2001	GDS.BUF\$LOC, *
000134	001002		BNE 1\$	
000136	004737	000232'	JSR PC,DECODE	
000142	013700	000000G	MOV NSD.SLOT, R0	2982
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)	*.RCSS.REG
000172	016600	000004	MOV 4(SP), R0	RCSS.REG, TEMP
000176	012716	003000	MOV #3000, (SP)	3005

ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (14)

000202	010246		MOV	R2,-(SP)	:	REF.NUM,*	
000204	010146		MOV	R1,-(SP)	:	GDS.BUF\$LOC,*	
000206	004737	002074'	JSR	PC,CTO.WAIT			
000212	022626		CMP	(SP)+,(SP)+			
000214	006000		ROR	R0			
000216	103002		BCC	2\$			
000220	004737	000232'	JSR	PC,DECODE			
000224	010100		MOV	R1,R0	:	GDS.BUF\$LOC,*	3012
000226	006300		ASL	R0			
000230	006300		ASL	R0			
000232	016037	000002G 000000G	MOV	OUT\$STD.BUF+2(R0),RET.ENSAD			
000240	016000	000002G	MOV	OUT\$STD.BUF+2(R0),R0	:		3017
000244	005760	000016	TST	16(R0)			
000250	001405		BEQ	3\$			
000252	012700	000031	MOV	#31,R0	:		3019
000256	010037	000000G	MOV	R0,RET.STATUS			
000262	000402		BR	4\$:		2934
000264	013700	000000G	MOV	RET.STATUS,R0			
000270	062706	000006	ADD	#6,SP	:		2903
000274	000207		RTS	PC			

: Routine Size: 95 words, Routine Base: \$CODE\$ + 2562
: Maximum stack depth per invocation: 9 words

: 3024

```

:
: C 3025  %(  
: C 3026  global routine EX_SUP_PROG =      !Executes supplied program  
: C 3027  
: C 3028  !++  
: C 3029  Functional Description :  
: C 3030  This command causes the server to transfer the program from host  
: C 3031  memory to an area in the controller and start its execution. The  
: C 3032  host supplies the address and length (in bytes) of a buffer  
: C 3033  containing the program header and initial load; the starting  
: C 3034  of the program, its memory requirements and any relocation information  
: C 3035  needed to run under the server are in the program header in a format  
: C 3036  which is none of the host business. This command is only legal when  
: C 3037  the server is in the idle state and return of a successful end packet  
: C 3038  puts the server into to active state.  
: C 3039  
: C 3040  The time out interval for this command is 30 seconds.  
: C 3041  
: C 3042  Formal Parameters :  
: C 3043  none  
: C 3044  
: C 3045  Implicit Inputs :  
: C 3046  NSD_SLOT      This global storage gets loaded by the routine  
: C 3047  'Get_nsd' and in it is stored the next send ring  
: C 3048  descriptor slot where the port/controller should  
: C 3049  be polling on and the place to put this commands  
: C 3050  command packet.  
: C 3051  
: C 3052  
: C 3053  Implicit Outputs :  
: C 3054  AZFMTR:      Azfmtr is the vector produced by DMCONV program and  
: C 3055  is decalared in module AZKEL6.  
: C 3056  DMSA:      These three bound addresses point to specific area  
: C 3057  OVSA:      in the DM code buffer 'azfmtr' and are used to  
: C 3058  HDSA:      define the buffer descriptors within this command.  
: C 3059  
: C 3060  Completion Codes :  
: C 3061  RET_STATUS:  Return status passes back to the calling routine  
: C 3062  the status of the just issued command.  
: C 3063  
: C 3064  Side Effects :  
: C 3065  The DM machine in the controller goes from the idle state to the  
: C 3066  active state on return of a successful return packet.  
: C 3067  --  
: C 3068  
: C 3069  begin  
: C 3070  
: C 3071  local  
: C 3072  REF_NUM,      !Stores unique cmd ref number  
: C 3073  ESP_BUF$LOC, !Stores out$standing cmd buffer location  
: C 3074  TEMP;        !A place to put the read IP register data  
: C 3075  
: C 3076  !  
: C 3077  ! Before we load up the command packet up
```

```

C 3078      | with all this good information get the
C 3079      | next send descriptor slot and a unique
C 3080      | command reference number.
C 3081
C 3082      | GET_NSD ();                                !Get the next send desc slot
C 3083      | REF_NUM = GET_CMDSREF ();                  !Get a unique command ref num
C 3084
C 3085      | UQ Port command envelope Header field definition
C 3086
C 3087      | SND_ENVELOPE [ .NSD_SLOT, MSG_LENGTH ] = SZ_ESP;      !Load the message length
C 3088      | SND_ENVELOPE [ .NSD_SLOT, CREDITS ] = ONE;            !Load the credits field
C 3089      | SND_ENVELOPE [ .NSD_SLOT, MSG_TYPE ] = 0;             !Define the msg type 'Sequential'
C 3090      | SND_ENVELOPE [ .NSD_SLOT, CONN_ID ] = 2;             !Define the conection ID as DUP
C 3091
C 3092      | DUP generic command envelope field definition
C 3093
C 3094      | SND_ENVELOPE [ .NSD_SLOT, CMD_LREF ] = .REF_NUM;     !Load command ref number
C 3095      | SND_ENVELOPE [ .NSD_SLOT, CMD_HREF ] = ZERO;        !Zero Hi order word of cmd ref
C 3096      | SND_ENVELOPE [ .NSD_SLOT, UN_USED ] = ZERO;         !Not used in DUP implimentation
C 3097      | SND_ENVELOPE [ .NSD_SLOT, UN_HUSED ] = ZERO;        !Not used in DUP implimentation
C 3098      | SND_ENVELOPE [ .NSD_SLOT, OP_CODE ] = OP_ESP;       !Load the command op-code
C 3099      | SND_ENVELOPE [ .NSD_SLOT, RSVD ] = ZERO;            !Not used
C 3100      | SND_ENVELOPE [ .NSD_SLOT, MODIFIER ] = ZERO;
C 3101
C 3102      | Command specfic command envelope field definition
C 3103
C 3104
C 3105      | Byte count of initial transfer (from bytes 0-3
C 3106      | of the program header).
C 3107
C 3108      | SND_ENVELOPE [ .NSD_SLOT, BLO_CNT ] = .AZFMTR [WRD0]; !Byte count low word
C 3109      | SND_ENVELOPE [ .NSD_SLOT, BHI_CNT ] = .AZFMTR [WRD1]; !Byte count high word
C 3110
C 3111      | Buffer descriptor definition for initial load. First
C 3112      | byte of this buffer is byte 0 of program header.
C 3113
C 3114      | SND_ENVELOPE [ .NSD_SLOT, BPA_LO ] = HDSA;          !Low unibus adrs <0-15>
C 3115      | SND_ENVELOPE [ .NSD_SLOT, BPA_HI ] = ZERO;         !Unibus adrs bits <16-17>
C 3116      | SND_ENVELOPE [ .NSD_SLOT, QBUS_EXT ] = ZERO;       !Q_bus extention adrs
C 3117      | SND_ENVELOPE [ .NSD_SLOT, RSV ] = ZERO;            !Reserved field
C 3118      | SND_ENVELOPE [ .NSD_SLOT, UBA_CHAN ] = ZERO;       !Unibus adaptor channel number
C 3119      | SND_ENVELOPE [ .NSD_SLOT, RSV0 ] = ZERO;          !These next four words are not
C 3120      | SND_ENVELOPE [ .NSD_SLOT, RSV1 ] = ZERO;          !used in the DUP implementation
C 3121      | SND_ENVELOPE [ .NSD_SLOT, RSV2 ] = ZERO;
C 3122      | SND_ENVELOPE [ .NSD_SLOT, RSV3 ] = ZERO;
C 3123
C 3124      | These next field definitions are the same
C 3125      | as above except they are for the overlay
C 3126      | buffer descriptors. To make life easy for
C 3127      | me I'll use the same names and just prefix
C 3128      | them with a $ for uniqueness.
C 3129
C 3130      | The overlay area immediately follows the

```

```

C 3131      ! initial load image in the program image.
C 3132
C 3133      SND_ENVELOPE [.NSD_SLOT, $BPA_LO] = .OVSA;      !Low unibus adrs <0-15>
C 3134      SND_ENVELOPE [.NSD_SLOT, $BPA_HI] = ZERO;      !Unibus adrs bits <16-17>
C 3135      SND_ENVELOPE [.NSD_SLOT, $QBUS_EXT] = ZERO;     !Q_bus extention adrs
C 3136      SND_ENVELOPE [.NSD_SLOT, $RSV] = ZERO;         !Reserved field
C 3137      SND_ENVELOPE [.NSD_SLOT, $UBA_CHAN] = ZERO;    !Unibus adaptor channel number
C 3138      SND_ENVELOPE [.NSD_SLOT, $RSV0] = ZERO;        !These next four words are not
C 3139      SND_ENVELOPE [.NSD_SLOT, $RSV1] = ZERO;        !used in the DUP implementation
C 3140      SND_ENVELOPE [.NSD_SLOT, $RSV2] = ZERO;
C 3141      SND_ENVELOPE [.NSD_SLOT, $RSV3] = ZERO;
C 3142
C 3143      ! Call the load out$standing command buffer routine
C 3144      ! and load this command into the buffer. The return
C 3145      ! from this routine will point us to the buffer location
C 3146      ! where this command is stored. Later we can look at
C 3147      ! this location to see if the interrupt service routine
C 3148      ! has received and process it.
C 3149
C 3150      ESP_BUF$LOC = LOAD_OUT$STD_BUF (.REF_NUM);      !Load the command
C 3151
C 3152      if .ESP_BUF$LOC eqlu OBF_CODE then DECODE ();    !Error if buffer is full
C 3153
C 3154      !
C 3155      ! Set the ownership bit to 1 giving this slot
C 3156      ! to the port/controller
C 3157
C 3158      SEND_RING [.NSD_SLOT, OWN_BIT] = PORT_OWNED;
C 3159
C 3160      ! Read the IP register to stimulate port polling
C 3161
C 3162      TEMP = .RDRX_ADDR [RCIP, RC_ALL];
C 3163
C 3164      ! Time out the port/controller processing the command.
C 3165
C 3166      ! The first test tests the connections ability to
C 3167      ! respond to this command without any errors in the SA
C 3168      ! register and for the command not timing out.
C 3169
C 3170      ! The second tests the DUP server for good status. If
C 3171      ! bad status is sent back then an error code is returned
C 3172      ! to the calling routine where the routine "decode" will
C 3173      ! decode and take the appropriate recovery. The time
C 3174      ! out routine will loop on delaying and checking the hi
C 3175      ! bit of the first word in the out$std_buf for a true.
C 3176      ! When true signals us that the interrupt service routine
C 3177      ! has received the endpacket and no connection errors
C 3178      ! were detected.
C 3179
C 3180
C 3181      if CTO_WAIT (%o'3000', .REF_NUM, .ESP_BUF$LOC) then DECODE ();    !Is return an error
C 3182
C 3183      !

```

ZROB4
REV A PATCH 00RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS28-Jun-1983 13:04:36
27-Jun-1983 18:06:48VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZROB4.B16;1 (15)SEQ 140
Page 53

```

: C 3184      ! Get the return envelope address from the out$std_buf
: C 3185      ! at this commands buffer location and check the packet
: C 3186      ! for good status error and die if bad status was returned
: C 3187
: C 3188      RET_EN$AD = .OUT$STD_BUF [.ESP_BUF$LOC, ENV_ADR];  !Get the ret env adr
: C 3189
: C 3190      ! Now test for good status
: C 3191
: C 3192
: C 3193      if .RET_EN$AD [STATUS] nequ ZERO                    !Test the status
: C 3194      then
: C 3195          return RET_STATUS = RSE_CODE                    !Return a 'Response status err'' code
: C 3196      else
: C 3197          return .RET_STATUS;                               !This ret_status is good or bad
: C 3198
: C 3199      end;
: C 3200      )%

```

```

3201 global routine EX_LOC_PROG = !Executes local program
3202
3203 !++
3204 Functional Description :
3205 Receipt of this command causes the controller to search its local
3206 media for the named program, load and execute it. Receipt of a
3207 successful response by the host means that the program is executing
3208 and the server is in the active state. The time out value for this
3209 command is specified in the get dust response
3210
3211 Formal Parameters :
3212 none
3213
3214 Implicit Inputs :
3215 NSD_SLOT This global storage gets loaded by the routine
3216 'Get_nsd' and in it is stored the next send ring
3217 descriptor slot where the port/controller should
3218 be polling on and the place to put this commands
3219 command packet.
3220
3221 Implicit Outputs :
3222 none
3223
3224 Completion Codes :
3225 RET_STATUS: Return status passes back to the calling routine
3226 the status of the just issued command.
3227
3228 Side Effects :
3229 The DM machine in the controller goes from the idle state to the
3230 active state on return of a successful return packet.
3231 !--
3232
3233 begin
3234
3235 local
3236 REF_NUM, !Stores unique cmd ref number
3237 ELP_BUF$LOC, !Stores outstanding cmd buffer location
3238 TEMP; !A place to store the read IP register data
3239
3240 !
3241 ! Before we load up the command packet up
3242 ! with all this good information get the
3243 ! next send descriptor slot and a unique
3244 ! command reference number.
3245
3246 GET_NSD (); !Get the next send desc slot
3247 REF_NUM = GET_CMD$REF (); !Get a unique command ref num
3248
3249 ! UQ Port command envelope Header field definition
3250
3251 SND_ENVELOPE [.NSD_SLOT, MSG_LENGTH] = SZ_ELP; !Load the message size
3252 SND_ENVELOPE [.NSD_SLOT, CREDITS] = ONE; !Load the credit size
3253 SND_ENVELOPE [.NSD_SLOT, MSG_TYPE] = 0; !Define the msg typ 'Sequential'

```

```
3254 SND_ENVELOPE [.NSD_SLOT, CONN_ID] = 2;      !Define the connectio ID 'DUP'
3255
3256 ! DUP generic command envelope field definition
3257
3258 SND_ENVELOPE [.NSD_SLOT, CMD_LREF] = .REF_NUM;      !Load the command ref number
3259 SND_ENVELOPE [.NSD_SLOT, CMD_HREF] = ZERO;      !Zero the Hi order cmd ref num
3260 SND_ENVELOPE [.NSD_SLOT, UN_USED] = ZERO;      !Not used in DUP implimentation
3261 SND_ENVELOPE [.NSD_SLOT, UN_HUSED] = ZERO;      !Not used in DUP implimentation
3262 SND_ENVELOPE [.NSD_SLOT, OP_CODE] = OP_ELP;      !Load this commands op-code
3263 SND_ENVELOPE [.NSD_SLOT, RSVD] = ZERO;      !Not used field
3264 SND_ENVELOPE [.NSD_SLOT, MODIFIER] = DUP_STND; !Define modifiers for this cmd
3265
3266 ! Command specfic command envelope field definition
3267
3268 SND_ENVELOPE [.NSD_SLOT, PN_0] = 'FO';      !Program name word 0
3269 SND_ENVELOPE [.NSD_SLOT, PN_1] = 'RM';      !Program name word 1
3270 SND_ENVELOPE [.NSD_SLOT, PN_2] = 'AT';      !Program name word 2
3271
3272 ! Call the load out$standing command buffer routine
3273 ! and load this command into the buffer. The return
3274 ! from this routine will point us to the buffer location
3275 ! where this command is stored. Later we can look at
3276 ! this location to see if the interrupt service routine
3277 ! has received and process it.
3278
3279 ELP_BUF$LOC = LOAD_OUT$STD_BUF (.REF_NUM); !Load the command
3280
3281 if .ELP_BUF$LOC eqlu OBF_CODE then DECODE ();      !Error if buffer is full
3282
3283
3284 ! Set the ownership bit to 1 giving this slot
3285 ! to the port/controller
3286
3287 SEND_RING [.NSD_SLOT, OWN_BIT] = PORT_OWNED;
3288
3289 ! Read the IP register to stimulate port polling
3290
3291 TEMP = .RDRX_ADDR [RCIP, RC_ALL];
3292
3293 ! Time out the port/controller processing the command.
3294
3295 ! The first test tests the connections ability to
3296 ! respond to this command without any errors in the SA
3297 ! register and for the command not timing out.
3298
3299 ! The second tests the DUP server for good status. If
3300 ! bad status is sent back then an error code is returned
3301 ! to the calling routine where the routine "decode" will
3302 ! decode and take the appropriate recovery. The time
3303 ! out routine will loop on delaying and checking the hi
3304 ! bit of the first word in the out$std_buf for a true.
3305 ! When true signals us that the interrupt service routine
3306 ! has received the endpacket and no connection errors
```



```

: 3307      ! were detected.
: 3308      !
: 3309
: 3310      if CTO_WAIT (3000, .REF_NUM, .ELP_BUF$LOC) then DECODE (); !Is return an error
: 3311
: 3312      !
: 3313      ! Get the return envelope address from the out$std_buf
: 3314      ! at this commands buffer location and check the packet
: 3315      ! for good status error and die if bad status was returned
: 3316
: 3317      RET_EN$AD = .OUT$STD_BUF [.ELP_BUF$LOC, ENV_ADR]; !Get the ret env adr
: 3318
: 3319      ! Now test for good status
: 3320
: 3321
: 3322      if .RET_EN$AD [STATUS] nequ ZERO          !Test the status
: 3323      then
: 3324          return RET_STATUS = RSE_CODE          !Return a 'Response status err'' code
: 3325      else
: 3326          return .RET_STATUS;                  !This ret_status is good or bad
: 3327
: 3328      end;

```

000000	004137	000000G	EX.LOC.PROG	.SBTTL	EX.LOC.PROG GLOBAL ROUTINE DECLARATIONS	
			EX.LOC.PROG::	JSR	R1,\$SAVE2	3201
000004	005746			TST	-(SP)	
000006	004737	000000'		JSR	PC,GET.NSD	3246
000012	004737	000146'		JSR	PC,GET.CMD\$REF	3247
000016	010002			MOV	R0,R2	
000020	013746	000000G		MOV	NSD.SLOT,-(SP)	3251
000024	012746	000054		MOV	#54,-(SP)	
000030	004737	000000G		JSR	PC,BLSMUL	
000034	012760	000060 000000G		MOV	#60,SND.ENVELOPE(R0)	
000042	012701	000002G		MOV	#SND.ENVELOPE+2,R1	3252
000046	060001			ADD	R0,R1	
000050	112711	000001		MOVB	#1,(R1)	3253
000054	112761	000002 000001		MOVB	#2,1(R1)	3254
000062	010260	000004G		MOV	R2,SND.ENVELOPE+4(R0)	3258
000066	005060	000006G		CLR	SND.ENVELOPE+6(R0)	3259
000072	005060	000010G		CLR	SND.ENVELOPE+10(R0)	3260
000076	005060	000012G		CLR	SND.ENVELOPE+12(R0)	3261
000102	112760	000003 000014G		MOVB	#3,SND.ENVELOPE+14(R0)	3262
000110	105060	000015G		CLRB	SND.ENVELOPE+15(R0)	3263
000114	012760	000001 000016G		MOV	#1,SND.ENVELOPE+16(R0)	3264
000122	012760	047506 000020G		MOV	#47506,SND.ENVELOPE+20(R0)	3268
000130	012760	046522 000022G		MOV	#46522,SND.ENVELOPE+22(R0)	3269
000136	012760	052101 000024G		MOV	#52101,SND.ENVELOPE+24(R0)	3270
000144	010216			MOV	R2,(SP)	3279
000146	004737	000054'		JSR	PC,LOAD.OUT\$STD.BUF	
000152	010001			MOV	R0,R1	
000154	020127	002001		CMP	R1,#2001	3281

ZRQB4 RDRX DISK FORMATTER
REV A PATCH 00 GLOBAL ROUTINE DECLARATIONS

000160	001002			BNE	1\$		
000162	004737	000232'		JSR	PC,DECODE		
000166	013700	000000G	1\$:	MOV	NSD.SLOT,R0	:	3287
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)	: *RCSS.REG	3291
000216	016600	000004		MOV	4(SP),R0	: RCSS.REG,TEMP	
000222	012716	005670		MOV	#5670,(SP)	:	3310
000226	010246			MOV	R2,-(SP)	: REF.NUM,*	
000230	010146			MOV	R1,-(SP)	: ELP.BUF\$LOC,*	
000232	004737	002074'		JSR	PC,CTO.WAIT		
000236	022626			CMP	(SP)+,(SP)+		
000240	006000			ROR	R0		
000242	103002			BCC	2\$		
000244	004737	000232'	2\$:	JSR	PC,DECODE	: ELP.BUF\$LOC,*	3317
000250	010100			MOV	R1,R0		
000252	006300			ASL	R0		
000254	006300			ASL	R0		
000256	016037	000002G 000000G		MOV	OUT\$STD.BUF+2(R0),RET.ENSAD		
000264	016000	000002G		MOV	OUT\$STD.BUF+2(R0),R0	:	3322
000270	005760	000016		TST	16(R0)		
000274	001405			BEQ	3\$		
000276	012700	000031		MOV	#31,R0	:	3324
000302	010037	000000G		MOV	R0,RET.STATUS	:	
000306	000402			BR	4\$:	3233
000310	013700	000000G	3\$:	MOV	RET.STATUS,R0		
000314	062706	000006	4\$:	ADD	#6,SP	:	3201
000320	000207			RTS	PC		

: Routine Size: 105 words, Routine Base: \$CODE\$ + 3060
: Maximum stack depth per invocation: 9 words

: 3329

```

3330 global routine SEND_DATA =      !Performs host-->port communications
3331
3332 !++
3333 Functional Description :
3334 These commands are used to communicate between the initiating host
3335 program and the remote program. Both send and receive commands
3336 specify a host buffer descriptor and a byte count. In the case of
3337 send data, the information in the buffer is read by the remote program
3338 and a send data response sent back to the host to acknowledge receipt.
3339 In the case of receive data the remote program writes data into the
3340 buffer up to the amount specified by the byte count and then sends a
3341 receive data response to the host to notify it of the transmission.
3342
3343 The send data and receive data commands are only legal when the
3344 server is in the active state. If the remote program terminates
3345 abnormally, putting the server back in the idle stae, outstanding
3346 send data and receive data commands may be lost. In the event that
3347 the specified timeout interval is exceeded, the host program should
3348 issue a get dust staus command to see if the remote program is
3349 still running (is. the dup server is active); if it is, the
3350 progress indicator should be remembered and the timeout interval
3351 should be re-installed. If the second timeout expires without a
3352 response and a second get dust status shows the remote program
3353 having made no progress in the interim then the program should be
3354 considered broken and should be aborted.
3355
3356 Formal Parameters :
3357 none
3358
3359 Implicit Inputs :
3360 NSD_SLOT      This global storage gets loaded by the routine
3361                'Get_nsd' and in it is stored the next send ring
3362                descriptor slot where the port/controller should
3363                be polling on and the place to put this commands
3364                command packet.
3365
3366 Implicit Outputs :
3367 none
3368
3369 Completion Codes :
3370 RET_STATUS:   Return status passes back to the calling routine
3371                the status of the just issued command.
3372
3373 Side Effects :
3374 none
3375 !--
3376
3377 begin
3378
3379 local
3380     REF_NUM,          !Stores unique cmd ref number
3381     SND_BUF$LOC,     !Stores outstanding cmd buffer location
3382     TEMP;            !A place to put read !? register data

```

```

3383
3384
3385     | Before we load up the command packet up
3386     | with all this good information get the
3387     | next send descriptor slot and a unique
3388     | command reference number.
3389
3390     | GET_NSD ();                !Get the next send desc slot
3391     | REF_NUM = GET_CMDSREF (); !Get a unique command ref num
3392
3393     | UQ Port command envelope Header field definition
3394
3395     | SND_ENVELOPE [.NSD_SLOT, MSG_LENGTH] = SZ_SED;    !Load the message size
3396     | SND_ENVELOPE [.NSD_SLOT, CREDITS] = ONE;         !Load the credit size
3397     | SND_ENVELOPE [.NSD_SLOT, MSG_TYPE] = 0;         !Define the message typ 'Sequential
3398     | SND_ENVELOPE [.NSD_SLOT, CONN_ID] = 2;         !Define the connection ID 'DUP'
3399
3400     | DUP generic command envelope field definition
3401
3402     | SND_ENVELOPE [.NSD_SLOT, CMD_LREF] = .REF_NUM;   !Load command reference number
3403     | SND_ENVELOPE [.NSD_SLOT, CMD_HREF] = ZERO;      !Zero Hi order cmd ref number
3404     | SND_ENVELOPE [.NSD_SLOT, UN_USED] = ZERO;      !Not used in DUP implimentation
3405     | SND_ENVELOPE [.NSD_SLOT, UN_HUSED] = ZERO;     !Not used in DUP implimentation
3406     | SND_ENVELOPE [.NSD_SLOT, OP_CODE] = OP_SED;    !Load this commands op-code
3407     | SND_ENVELOPE [.NSD_SLOT, RSV0] = ZERO;        !Not used field
3408     | SND_ENVELOPE [.NSD_SLOT, MODIFIER] = ZERO;     !Define the commands modifiers
3409
3410     | Command specific command envelope field definition
3411
3412     | Byte count of transfer
3413
3414     | SND_ENVELOPE [.NSD_SLOT, BLO_CNT] = SNDB_COUNT; !Byte count low word
3415     | SND_ENVELOPE [.NSD_SLOT, BHI_CNT] = ZERO;      !Byte count high word
3416
3417     | Buffer descriptor definition
3418
3419     | SND_ENVELOPE [.NSD_SLOT, BPA_LO] = SND_BUF;    !Buffer physical adrs <0-15>
3420     | SND_ENVELOPE [.NSD_SLOT, BPA_HI] = ZERO;      !Buffer physical adrs bits <16-17>
3421     | SND_ENVELOPE [.NSD_SLOT, QBUS_EXT] = ZERO;    !Q bus extention adrs
3422     | SND_ENVELOPE [.NSD_SLOT, RSV] = ZERO;        !Reserved field
3423     | SND_ENVELOPE [.NSD_SLOT, UBA_CHAN] = ZERO;    !Unibus adaptor channel number
3424     | SND_ENVELOPE [.NSD_SLOT, RSV0] = ZERO;      !These next four words are not
3425     | SND_ENVELOPE [.NSD_SLOT, RSV1] = ZERO;      !used in the UQ Port implementation
3426     | SND_ENVELOPE [.NSD_SLOT, RSV2] = ZERO;
3427     | SND_ENVELOPE [.NSD_SLOT, RSV3] = ZERO;
3428     |
3429
3430     | Call the load out$standing command buffer routine
3431     | and load this command into the buffer. The return
3432     | from this routine will point us to the buffer location
3433     | where this command is stored. Later we can look at
3434     | this location to see if the interrupt service routine
3435     | has received and process it.

```

```
3436 |  
3437 |SND_BUF$LOC = LOAD_OUT$STD_BUF (.REF_NUM); !Load the command  
3438 |  
3439 |if .SND_BUF$LOC eqlu OBF_CODE then DECODE (); !Error if buffer is full  
3440 |  
3441 |  
3442 |Set the ownership bit to 1 giving this slot  
3443 |to the port/controller  
3444 |  
3445 |SEND_RING [.NSD_SLOT, OWN_BIT] = PORT_OWNED;  
3446 |  
3447 |Read the IP register to stimulate port polling  
3448 |  
3449 |TEMP = .RDRX_ADDR [RCIP, RC_ALL];  
3450 |  
3451 |  
3452 |Time out the port/controller for the response from this  
3453 |command.  
3454 |  
3455 |If the controller times out then:  
3456 | 1. See what kind of error was returned. If the error  
3457 | is a type other than a CTO_CODE (controller time out)  
3458 | then call routine Decode which does the appropriate  
3459 | action based on the error.  
3460 |  
3461 | 2. If the returned error is an CTO_CODE then do a get  
3462 | dust status and check the progress indicator to look  
3463 | for an increase, indicating that the remote program is  
3464 | still running and is not dead.  
3465 |  
3466 | If the indicator hasn't changed then assume that the  
3467 | remote program is dead and return an error code of  
3468 | RPD_CODE (remote program dead code) and exit.  
3469 |  
3470 | If the indicator has changed then assume that the  
3471 | remote program is still running, save a copy of its  
3472 | value and reinstate the controller time out delay and  
3473 | repeat the loop.  
3474 |  
3475 | As long as the progress indicator in the remote program  
3476 | is still increasing this loop will be repeated for ever.  
3477 |  
3478 | If the controller doesn't time then return with the return  
3479 | code returned from routine CTO_WAIT () which could be either  
3480 | a success or error code by definition of this host code.  
3481 |  
3482 |  
3483 |while TRUE do !Repeat for ever  
3484 |begin  
3485 |BREAK; !Flag control C's  
3486 |  
3487 |Do a controller time out and determine if the controller  
3488 | has processed the command or if a fatal error has occurred.
```

```
3489      !
3490
3491      if CTO_WAIT (3000, .REF_NUM, .SND_BUF$LOC)      !Is return an error
3492      then
3493          begin
3494              !
3495              ! If the return status code eql's a CTO_CODE
3496              ! then see if the remote program is still
3497              ! running. If it is then save the progress
3498              ! indicator and repeat the loop else call
3499              ! routine Decode ().
3500              !
3501              !
3502              if .RET_STATUS eqlu CTO_CODE      !Is this a controller time out
3503              then
3504                  begin
3505                      REF_NUM = .REF_NUM;
3506
3507                      !
3508                      !
3509                      if GET_DUST_STATUS () then DECODE ();      !Get the dust status
3510
3511                      !
3512                      !
3513                      if .RET_EN$AD [PLO_IND] gtru .PID_SAVE      !Any progress been made
3514                      then
3515                          PID_SAVE = .RET_EN$AD [PLO_IND]      !Still running save Pid
3516                      else
3517                          return RET_STATUS = RPD_CODE;      !No progress so flag error
3518                      end
3519                  else
3520                      !
3521                      ! The return status code was not a controller time
3522                      ! out code so something else is wrong. Call the
3523                      ! routine Decode () to find out what went wrong.
3524                      !
3525                      DECODE ()
3526                  end
3527              else
3528                  begin
3529                      !
3530                      ! The command has been received by the interrupt service.
3531                      !
3532                      ! Get this commands return envelope address out of the
3533                      ! out$std_buf and check for good return status error and
3534                      ! die if bad status.
3535                      !
3536                      !
3537                      RET_EN$AD = .OUT$STD_BUF [.SND_BUF$LOC, ENV_ADR];      !Get the ret env adr
3538                      !
3539                      ! Test for good status
3540                      !
3541                  end
```

```

: 3542      if .RET_EN$AD [STATUS] nequ ZERO      !Test the status
: 3543      then
: 3544          return RET_STATUS = RSE_CODE      !Return a 'Response status err' code
: 3545      else
: 3546          return .RET_STATUS;              !This ret_status is good or bad
: 3547
: 3548      end;
: 3549
: 3550      end;
: 3551
: 3552      return .RET_STATUS;                    !It won't compile without this here
: 3553      end;

```

			.SBTTL	SEND.DATA GLOBAL ROUTINE DECLARATIONS	
000000	004137	000000G	SEND.DATA::		
			JSR	R1,\$SAVE2	3330
000004	005746		TST	-(SP)	
000006	004737	000000'	JSR	PC,GET.NSD	3390
000012	004737	000146'	JSR	PC,GET.CMD\$REF	3391
000016	010002		MOV	R0,R2	* ,REF.NUM
000020	013746	000000G	MOV	NSD.SLOT,-(SP)	3395
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	3396
000046	060001		ADD	R0,R1	
000050	112711	000001	MOVB	#1,(R1)	3397
000054	112761	000002 000001	MOVB	#2,1(R1)	3398
000062	010260	000004G	MOV	R2,SND.ENVELOPE+4(R0)	REF.NUM,*
000066	005060	000006G	CLR	SND.ENVELOPE+6(R0)	3402
000072	005060	000010G	CLR	SND.ENVELOPE+10(R0)	3403
000076	005060	000012G	CLR	SND.ENVELOPE+12(R0)	3404
000102	112760	000004 000014G	MOVB	#4,SND.ENVELOPE+14(R0)	3405
000110	105060	000015G	CLRB	SND.ENVELOPE+15(R0)	3406
000114	005060	000016G	CLR	SND.ENVELOPE+16(R0)	3407
000120	012760	000020 000020G	MOV	#20,SND.ENVELOPE+20(R0)	3408
000126	005060	000022G	CLR	SND.ENVELOPE+22(R0)	3415
000132	012760	000000G 000024G	MOV	#SND.BUF,SND.ENVELOPE+24(R0)	3416
000140	012701	000026G	MOV	#SND.ENVELOPE+26,R1	3420
000144	060001		ADD	R0,R1	3421
000146	105011		CLRB	(R1)	3423
000150	105061	000001	CLRB	1(R1)	3424
000154	005060	000030G	CLR	SND.ENVELOPE+30(R0)	3425
000160	005060	000032G	CLR	SND.ENVELOPE+32(R0)	3426
000164	005060	000034G	CLR	SND.ENVELOPE+34(R0)	3427
000170	005060	000036G	CLR	SND.ENVELOPE+36(R0)	3428
000174	010216		MOV	R2,(SP)	REF.NUM,*
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,*
000210	001002		BNE	1\$	
000212	004737	000232'	JSR	PC,DECODE	3439

ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

000216	013700	000000G	1\$:	MOV	NSD.SLOT,R0	:	3445
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\$\$REG	3449
000246	016600	000004		MOV	4(SP),R0	: RC\$\$REG,TEMP	
000252	104422		2\$:	TRAP	22	:	3484
000254	012716	005670		MOV	#5670,(SP)	:	3491
000260	010246			MOV	R2,-(SP)	:	
000262	010146			MOV	R1,-(SP)	: REF.NUM,*	
000264	004737	002074'		JSR	PC,CTO.WAIT	: SND.BUF\$LOC,*	
000270	022626			CMP	(SP)+,(SP)+	:	
000272	006000			ROR	R0	:	
000274	103032			BCC	6\$:	
000276	023727	000000G 000011		CMP	RET.STATUS,#11	:	3502
000304	001023			BNE	5\$:	
000306	004737	002562'		JSR	PC,GET.DUST.STATUS	:	3509
000312	006000			ROR	R0	:	
000314	103002			BCC	3\$:	
000316	004737	000232'		JSR	PC,DECODE	:	
000322	013700	000000G	3\$:	MOV	RET.ENSAD,R0	:	3513
000326	026037	000024 000000G		CMP	24(R0),PID.SAVE	:	
000334	101404			BLOS	4\$:	
000336	016037	000024 000000G		MOV	24(R0),PID.SAVE	:	3515
000344	000742			BR	2\$:	3513
000346	012700	000051	4\$:	MOV	#51,R0	:	3517
000352	000420			BR	7\$:	
000354	004737	000232'	5\$:	JSR	PC,DECODE	:	3525
000360	000734			BR	2\$:	3491
000362	010100		6\$:	MOV	R1,R0	: SND.BUF\$LOC,*	3537
000364	006300			ASL	R0	:	
000366	006300			ASL	R0	:	
000370	016037	000002G 000000G		MOV	OUT\$STD.BUF+2(R0),RET.ENSAD	:	
000376	016000	000002G		MOV	OUT\$STD.BUF+2(R0),R0	:	3542
000402	005760	000016		TST	16(R0)	:	
000406	001405			BEQ	8\$:	
000410	012700	000031		MOV	#31,R0	:	3544
000414	010037	000000G	7\$:	MOV	R0,RET.STATUS	:	
000420	000402			BR	9\$:	3529
000422	013700	000000G	8\$:	MOV	RET.STATUS,R0	:	
000426	062706	000006	9\$:	ADD	#6,SP	:	3330
000432	000207			RTS	PC	:	

: Routine Size: 142 words, Routine Base: \$CODE\$ + 3402
: Maximum stack depth per invocation: 9 words

: 3554


```
3555 global routine REC_DATA =      !Performs host-->port communications
3556
3557 !++
3558 ! Functional Description :
3559 ! These commands are used to communicate between the initiating host
3560 ! program and the remote program. Both send and receive commands
3561 ! specify a host buffer descriptor and a byte count. In the case of
3562 ! send data, the information in the buffer is read by the remote program
3563 ! and a send data response sent back to the host to acknowledge receipt.
3564 ! In the case of receive data the remote program writes data into the
3565 ! buffer up to the amount specified by the byte count and then sends a
3566 ! receive data response to the host to notify it of the transmission.
3567
3568 ! The send data and receive data commands are only legal when the
3569 ! server is in the active state. If the remote program terminates
3570 ! abnormally, putting the server back in the idle state, outstanding
3571 ! send data and receive data commands may be lost. In the event that
3572 ! the specified timeout interval is exceeded, the host program should
3573 ! issue a get dust status command to see if the remote program is
3574 ! still running (is. the dup server is active); if it is, the
3575 ! progress indicator should be remembered and the timeout interval
3576 ! should be re-installed. If the second timeout expires without a
3577 ! response and a second get dust status shows the remote program
3578 ! having made no progress in the interim then the program should be
3579 ! considered broken and should be aborted.
3580
3581 ! Formal Parameters :
3582 ! none
3583
3584 ! Implicit Inputs :
3585 ! NSD_SLOT
3586 ! This global storage gets loaded by the routine
3587 ! 'Get_nsd' and in it is stored the next send ring
3588 ! descriptor slot where the port/controller should
3589 ! be polling on and the place to put this commands
3590 ! command packet.
3591
3592 ! Implicit Outputs :
3593 ! none
3594
3595 ! Completion Codes :
3596 ! RET_STATUS:
3597 ! Return status passes back to the calling routine
3598 ! the status of the just issued command.
3599
3600 ! Side Effects :
3601 ! none
3602
3603 ! --
3604
3605 ! begin
3606 ! local
3607 ! REF_NUM, !Stores unique cmd ref number
3608 ! REC_BUF$LOC, !Stores outstanding cmd buffer location
3609 ! TEMP; !A place to put read IP register data
```

```

3608
3609
3610 : Before we load up the command packet up
3611 : with all this good information get the
3612 : next send descriptor slot and a unique
3613 : command reference number.
3614
3615 GET_NSD (); !Get the next send desc slot
3616 REF_NUM = GET_CMDSREF (); !Get a unique command ref num
3617
3618 : UQ Port command envelope Header field definition
3619
3620 SND_ENVELOPE [.NSD_SLOT, MSG_LENGTH] = SZ_RED; !Load message length
3621 SND_ENVELOPE [.NSD_SLOT, CREDITS] = ONE; !Load credit size
3622 SND_ENVELOPE [.NSD_SLOT, MSG_TYPE] = 0; !Define message type 'Sequential'
3623 SND_ENVELOPE [.NSD_SLOT, CONN_ID] = 2; !Define connection ID 'DUP'
3624
3625 : DUP generic command envelope field definition
3626
3627 SND_ENVELOPE [.NSD_SLOT, CMD_LREF] = .REF_NUM; !Load command reference number
3628 SND_ENVELOPE [.NSD_SLOT, CMD_HREF] = ZERO; !Zero Hi order cmd ref num
3629 SND_ENVELOPE [.NSD_SLOT, UN_USED] = ZERO; !Not used in DUP implimentation
3630 SND_ENVELOPE [.NSD_SLOT, UN_HUSED] = ZERO; !Not used in DUP implimentation
3631 SND_ENVELOPE [.NSD_SLOT, OP_CODE] = OP_RED; !Load this commands op-code
3632 SND_ENVELOPE [.NSD_SLOT, RSV0] = ZERO; !Not used field
3633 SND_ENVELOPE [.NSD_SLOT, MODIFIER] = ZERO; !Define this commands modifiers
3634
3635 : Command specfic command envelope field definition
3636
3637
3638 : Byte count of transfer
3639
3640 SND_ENVELOPE [.NSD_SLOT, BLO_CNT] = RECB_SIZE; !Byte count low word
3641 SND_ENVELOPE [.NSD_SLOT, BHI_CNT] = ZERO; !Byte count high word
3642
3643 : Buffer descriptor definition
3644
3645 SND_ENVELOPE [.NSD_SLOT, BPA_LO] = REC_BUF; !Low unibus adrs <0-15>
3646 SND_ENVELOPE [.NSD_SLOT, BPA_HI] = ZERO; !Unibus adrs bits <16-17>
3647 SND_ENVELOPE [.NSD_SLOT, QBUS_EXT] = ZERO; !Q_bus extention adrs
3648 SND_ENVELOPE [.NSD_SLOT, RSV] = ZERO; !Reserved field
3649 SND_ENVELOPE [.NSD_SLOT, UBA_CHAN] = ZERO; !Unibus adaptor channel number
3650 SND_ENVELOPE [.NSD_SLOT, RSV0] = ZERO; !These next four words are not
3651 SND_ENVELOPE [.NSD_SLOT, RSV1] = ZERO; !used in the UQ Port implementation
3652 SND_ENVELOPE [.NSD_SLOT, RSV2] = ZERO;
3653 SND_ENVELOPE [.NSD_SLOT, RSV3] = ZERO;
3654
3655 : Call the load out$standing command buffer routine
3656 : and load this command into the buffer. The return
3657 : from this routine will point us to the buffer location
3658 : where this command is stored. Later we can look at
3659 : this location to see if the interrupt service routine
3660 : has received and process it.

```

```
3661 !
3662 REC_BUF$LOC = LOAD_OUT$STD_BUF (.REF_NUM); !Load the command
3663
3664 if .REC_BUF$LOC eqlu OBF_CODE then DECODE (); !Error if buffer is full
3665
3666 !
3667 ! Set the ownership bit to 1 giving
3668 ! this slot to the port/controller
3669
3670 SEND_RING [.NSD_SLOT, OWN_BIT] = PORT_OWNED;
3671
3672 ! Read the IP register to stimulate port polling
3673
3674 TEMP = .RDRX_ADDR [RCIP, RC_ALL];
3675
3676 !+
3677 ! Time out the port/controller for the response from this command.
3678
3679 ! If the controller times out then:
3680 ! 1. See what kind of error was returned. If the error
3681 ! is a type other than a CTO_CODE (controller time out)
3682 ! then call routine Decode which does the appropriate
3683 ! action based on the error.
3684
3685 ! 2. If the returned error is an CTO_CODE then do a get
3686 ! dust status and check the progress indicator to look
3687 ! for an increase, indicating that the remote program is
3688 ! still running and is not dead.
3689
3690 ! If the indicator hasn't changed then assume that the
3691 ! remote program is dead and return an error code of
3692 ! RPD_CODE (remote program dead code) and exit.
3693
3694 ! If the indicator has changed then assume that the
3695 ! remote program is still running, save a copy of its
3696 ! value and reinstate the controller time out delay and
3697 ! repeat the loop.
3698
3699 ! As long as the progress indicator in the remote program
3700 ! is still increasing this loop will be repeated for ever.
3701
3702 ! If the controller doesn't time then return with the return code
3703 ! returned from routine CTO_WAIT () which could be either a success
3704 ! or error code by definition of this host code.
3705 !-
3706
3707 while TRUE do !Repeat for ever
3708     begin
3709     BREAK; !Flag control C's
3710
3711     ! Do a controller time out and determine if the controller
3712     ! has processed the command or if a fatal error has occurred.
3713
```

```
3714 if CTO_WAIT (3000, .REF_NUM, .REC_BUF$LOC)      !Is return an error
3715 then
3716 begin
3717     |
3718     | If the return status code eql's a CTO_CODE
3719     | then see if the remote program is still
3720     | running. If it is then save the progress
3721     | indicator and repeat the loop else call
3722     | routine Decode ().
3723     |
3724     |
3725     |
3726 if .RET_STATUS eqlu CTO_CODE      !Is this a controller time out
3727 then
3728 begin
3729     REF_NUM = .REF_NUM;
3730
3731     |
3732     | if GET_DUST_STATUS () then DECODE ();      !Get the dust status
3733     |
3734     | if .RET_EN$AD [PLO_IND] neq .PID_SAVE      !Any progress been made
3735     | then
3736     |     PID_SAVE = .RET_EN$AD [PLO_IND]      !Still running save Pid
3737     | else
3738     |     return RET_STATUS = RPD_CODE;      !No progress so flag error
3739     |
3740     |
3741     | end
3742 else
3743     |
3744     | The return status code was not a controller time
3745     | out code so something else is wrong. Call the
3746     | routine Decode () to find out what went wrong.
3747     |
3748     | DECODE ()
3749     |
3750 end
3751 else
3752 begin
3753     |
3754     | The command has been received by the interrupt service.
3755     |
3756     | Get this commands return envelope address out of the
3757     | out$std_buf and check for good return status error and
3758     | die if bad status.
3759     |
3760     | RET_EN$AD = .OUT$STD_BUF [.REC_BUF$LOC, ENV_ADR];      !Get the ret env adr
3761     |
3762     | Test for good status
3763     |
3764     |
3765     | if .RET_EN$AD [STATUS] nequ ZERO      !Test the status
3766     | then
```

ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

M 12
28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (18)

SEQ 155
Page 68

```

:          3767          return RET_STATUS = RSE_CODE      !Return a 'Response status err' code
:          3768          else
:          3769          return .RET_STATUS;                !This ret_status is good or bad
:          3770
:          3771          end;
:          3772
:          3773          end;
:          3774
:          3775          return .RET_STATUS;                !It won't compile without this here
:          3776          end;

```

			.SBTTL	REC.DATA GLOBAL ROUTINE DECLARATIONS	
000000	004137	000000G	REC.DATA::		
			JSR	R1,\$SAVE2	3555
			TST	-(SP)	
000004	005746		JSR	PC,GET.NSD	3615
000006	004737	000000'	JSR	PC,GET.CMD\$REF	3616
000012	004737	000146'	MOV	R0,R2	
000016	010002		MOV	NSD.SLOT,-(SP)	*,REF.NUM
000020	013746	000000G	MOV	#54,-(SP)	3620
000024	012746	000054	JSR	PC,BL\$MUL	
000030	004737	000000G	MOV	#34,SND.ENVELOPE(R0)	
000034	012760	000034 000000G	MOV	#SND.ENVELOPE+2,R1	3621
000042	012701	000002G	ADD	R0,R1	
000046	060001		MOVB	#1,(R1)	3622
000050	112711	000001	MOVB	#2,1(R1)	3623
000054	112761	000002 000001	MOV	R2,SND.ENVELOPE+4(R0)	REF.NUM,*
000062	010260	000004G	CLR	SND.ENVELOPE+6(R0)	3627
000066	005060	000006G	CLR	SND.ENVELOPE+10(R0)	3628
000072	005060	000010G	CLR	SND.ENVELOPE+12(R0)	3629
000076	005060	000012G	CLR	SND.ENVELOPE+14(R0)	3630
000102	112760	000005 000014G	MOVB	#5,SND.ENVELOPE+15(R0)	3631
000110	105060	000015G	CLRB	SND.ENVELOPE+16(R0)	3632
000114	005060	000016G	CLR	SND.ENVELOPE+17(R0)	3633
000120	012760	000170 000020G	MOV	#170,SND.ENVELOPE+20(R0)	3640
000126	005060	000022G	CLR	SND.ENVELOPE+22(R0)	3641
000132	012760	000000G 000024G	MOV	#REC.BUF,SND.ENVELOPE+24(R0)	3645
000140	012701	000026G	MOV	#SND.ENVELOPE+26,R1	3646
000144	060001		ADD	R0,R1	
000146	105011		CLRB	(R1)	3648
000150	105061	000001	CLRB	1(R1)	3649
000154	005060	000030G	CLR	SND.ENVELOPE+30(R0)	3650
000160	005060	000032G	CLR	SND.ENVELOPE+32(R0)	3651
000164	005060	000034G	CLR	SND.ENVELOPE+34(R0)	3652
000170	005060	000036G	CLR	SND.ENVELOPE+36(R0)	3653
000174	010216		MOV	R2,(SP)	REF.NUM,*
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,*
000210	001002		BNE	1\$	3664
000212	004737	000232'	JSR	PC,DECODE	
000216	013700	000000G	MOV	NSD.SLOT,R0	3670
000222	006300		ASL	R0	

ZRQB4 RDRX DISK FORMATTER
REV A PATCH 00 GLOBAL ROUTINE DECLARATIONS

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\$\$REG 3674
000246	016600	000004		MOV	4(SP),R0	:	RC\$\$REG,TEMP 3708
000252	104422			TRAP	22	:	3715
000254	012716	005670	2\$:	MOV	#5670,(SP)	:	
000260	010246			MOV	R2,-(SP)	:	REF.NUM,*
000262	010146			MOV	R1,-(SP)	:	REC.BUF\$LOC,*
000264	004737	002074'		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	:	3726
000304	001023			BNE	5\$		
000306	004737	002562'		JSR	PC,GET.DUST.STATUS	:	3733
000312	006000			ROR	R0		
000314	103002			BCC	3\$		
000316	004737	000232'		JSR	PC,DECODE		
000322	013700	000000G	3\$:	MOV	RET.EN\$AD,R0	:	3735
000326	026037	000024	000000G	CMP	24(R0),PID.SAVE		
000334	001404			BEQ	4\$		
000336	016037	000024	000000G	MOV	24(R0),PID.SAVE	:	3737
000344	000742			BR	2\$:	3735
000346	012700	000051	4\$:	MOV	#51,R0	:	3739
000352	000420			BR	7\$		
000354	004737	000232'	5\$:	JSR	PC,DECODE	:	3748
000360	000734			BR	2\$:	3715
000362	010100		6\$:	MOV	R1,R0	:	REC.BUF\$LOC,* 3760
000364	006300			ASL	R0		
000366	006300			ASL	R0		
000370	016037	000002G	000000G	MOV	OUT\$STD.BUF+2(R0),RET.EN\$AD	:	3765
000376	016000	000002G		MOV	OUT\$STD.BUF+2(R0),R0	:	
000402	005760	000016		TST	16(R0)		
000406	001405			BEQ	8\$		
000410	012700	000031		MOV	#31,R0	:	3767
000414	010037	000000G	7\$:	MOV	R0,RET.STATUS	:	3752
000420	000402			BR	9\$:	
000422	013700	000000G	8\$:	MOV	RET.STATUS,R0	:	
000426	062706	000006	9\$:	ADD	#6,SP	:	3555
000432	000207			RTS	PC		

: Routine Size: 142 words, Routine Base: \$CODE\$ + 4036
: Maximum stack depth per invocation: 9 words

: 3777

```
3778 global routine SET_CNTRLR_CHAR =          !Sets control characteristics
3779
3780 !++
3781 | Functional Description :
3782 | The SET CONTROLLER CHARACTERISTICS command is used to set host
3783 | settable unit characteristics and obtain those unit
3784 | characteristics that are essential for proper class driver
3785 | operation. This command never alters the unit's state
3786 | ('unit-online', 'unit-available', 'unit-offline'). It is
3787 | meaningless to set host settable characteristics for a unit
3788 | that is 'unit-available' or 'unit-offline'.
3789
3790 | Formal Parameters :
3791 | none
3792
3793 | Implicit Inputs :
3794 | NSD_SLOT          This global storage gets loaded by the routine
3795 |                   'Get_nsd' and in it is stored the next send ring
3796 |                   descriptor slot where the port/controller should
3797 |                   be polling on and the place to put this commands
3798 |                   command packet.
3799
3800 | Implicit Outputs :
3801 | none
3802
3803 | Completion Codes :
3804 | RET_STATUS:      Return status passes back to the calling routine
3805 |                   the status of the just issued command.
3806
3807 | Side Effects :
3808 | Any previously defined controller characteristics will possibly
3809 | be altered after execution of this command.
3810 |
3811 | --
3812
3813 | begin
3814 |
3815 | local
3816 |     REF_NUM,          !Stores unique cmd ref number
3817 |     SCC_BUF$LOC,     !Stores outstanding cmd buffer location
3818 |     TEMP;            !A place to put read IP register data
3819 |
3820 |
3821 | | Before we load up the command packet up
3822 | | with all this good information get the
3823 | | next send descriptor slot and a unique
3824 | | command reference number.
3825 |
3826 | GET_NSD ();          !Get the next send desc slot
3827 | REF_NUM = GET_CMD$REF (); !Get a unique command ref num
3828 |
3829 | | UQ Port command envelope Header field definition
3830 |
```

```
3831 SND_ENVELOPE [.NSD_SLOT, MSG_LENGTH] = SZ_SCC;      !Load message length
3832 SND_ENVELOPE [.NSD_SLOT, CREDITS] = ONE;           !Load credit size
3833 SND_ENVELOPE [.NSD_SLOT, MSG_TYPE] = 0;           !Define message type 'Sequential'
3834 SND_ENVELOPE [.NSD_SLOT, CONN_ID] = 0;           !Define connection ID 'DUP'
3835
3836 : MSCP generic command envelope field definition
3837
3838 SND_ENVELOPE [.NSD_SLOT, CMD_LREF] = .REF_NUM;      !Load command reference number
3839 SND_ENVELOPE [.NSD_SLOT, CMD_HREF] = ZERO;        !Zero Hi order cmd ref num
3840 SND_ENVELOPE [.NSD_SLOT, UN_USED] = ZERO;         !Not used in DUP implimentation
3841 SND_ENVELOPE [.NSD_SLOT, UN_HUSED] = ZERO;       !Not used in DUP implimentation
3842 SND_ENVELOPE [.NSD_SLOT, OP_CODE] = OP_SCC;      !Load this commands op-code
3843 SND_ENVELOPE [.NSD_SLOT, RSVD] = ZERO;           !Not used field
3844 SND_ENVELOPE [.NSD_SLOT, MODIFIER] = ZERO;       !Define this commands modifiers
3845
3846 : Command specific command envelope field definition
3847
3848 SND_ENVELOPE [.NSD_SLOT, MSCP_VER] = ZERO;        !MSCP version
3849 SND_ENVELOPE [.NSD_SLOT, CTL_FLAGS] = ZERO;      !Controller flags
3850 SND_ENVELOPE [.NSD_SLOT, HOST_TOV] = ZERO;       !Host time out value
3851 SND_ENVELOPE [.NSD_SLOT, RSVD] = ZERO;           !Reserved
3852 SND_ENVELOPE [.NSD_SLOT, TSD_0] = ZERO;          !Time and Date word 0
3853 SND_ENVELOPE [.NSD_SLOT, TSD_1] = ZERO;          !Time and Date word 1
3854 SND_ENVELOPE [.NSD_SLOT, TSD_2] = ZERO;          !Time and Date word 2
3855 SND_ENVELOPE [.NSD_SLOT, TSD_3] = ZERO;          !Time and Date word 3
3856 SND_ENVELOPE [.NSD_SLOT, CDP_LO] = ZERO;        !Cntlr dep parameter lo word
3857 SND_ENVELOPE [.NSD_SLOT, CDP_HI] = ZERO;        !Cntlr dep parameter hi wrd
3858
3859 : Call the load out$standing command buffer routine
3860 : and load this command into the buffer. The return
3861 : from this routine will point us to the buffer location
3862 : where this command is stored. Later we can look at
3863 : this location to see if the interrupt service routine
3864 : has received and process it.
3865
3866 SCC_BUF$LOC = LOAD_OUT$STD_BUF (.REF_NUM);      !Load the command
3867
3868 if .SCC_BUF$LOC eqlu OBF_CODE then DECODE ();    !Error if buffer is full
3869
3870
3871 : Set the ownership bit to 1 giving this slot
3872 : to the port/controller
3873
3874 SEND_RING [.NSD_SLOT, OWN_BIT] = PORT_OWNED;
3875
3876 : Read the IP register to stimulate port polling
3877
3878 TEMP = .RDRX_ADDR [RCIP, RC_ALL];
3879
3880 : Time out the port/controller processing the command.
3881
3882 : The first test tests the connections ability to
3883 : respond to this command without any errors in the SA
```



```

3884 | register and for the command not timing out.
3885 |
3886 | The second tests the DUP server for good status. If
3887 | bad status is sent back then an error code is returned
3888 | to the calling routine where the routine "decode" will
3889 | decode and take the appropriate recovery. The time
3890 | out routine will loop on delaying and checking the hi
3891 | bit of the first word in the out$std_buf for a true.
3892 | When true signals us that the interrupt service routine
3893 | has received the endpacket and no connection errors
3894 | were detected.
3895 |
3896 |
3897 | if CTO_WAIT (3000, .REF_NUM, .SCC_BUF$LOC) then DECODE (); !Is return an error
3898 |
3899 |
3900 | Get the return envelope address from the out$std_buf
3901 | at this commands buffer location and check the packet
3902 | for good status error and die if bad status was returned
3903 |
3904 | RET_EN$AD = .OUT$STD_BUF [.SCC_BUF$LOC, ENV_ADR]; !Get the ret env adr
3905 |
3906 | Now test for good status
3907 |
3908 |
3909 | if .RET_EN$AD [STATUS] nequ ZERO !Test the status
3910 | then
3911 | return RET_STATUS = RSE_CODE !Return a 'Response status err' code
3912 | else
3913 | return .RET_STATUS; !This ret_status is good or bad
3914 |
3915 | end;

```

000000	004137	000000G	.SBTTL SET.CNTRL.CHAR GLOBAL ROUTINE DECLARATIONS		
			SET.CNTRL.CHAR::		
			JSR R1,\$SAVE2	:	3778
000004	005746		TST -(SP)	:	
000006	004737	000000'	JSR PC,GET.NSD	:	3826
000012	004737	000146'	JSR PC,GET.CMD\$REF	:	3827
000016	010002		MOV R0,R2	: *,.REF.NUM	
000020	013746	000000G	MOV NSD.SLOT,-(SP)	:	3831
000024	012746	000054	MOV #54,-(SP)	:	
000030	004737	000000G	JSR PC,BLSMUL	:	
000034	012760	000040 000000G	MOV #40,SND.ENVELOPE(R0)	:	
000042	012701	000002G	MOV #SND.ENVELOPE+2,R1	:	3832
000046	060001		ADD R0,R1	:	
000050	112711	000001	MOVB #1,(P1)	:	3833
000054	105061	000001	CLRB 1(R1)	:	3834
000060	010260	000004G	MOV R2,SND.ENVELOPE+4(R0)	: REF.NUM,*	3838
000064	005060	000006G	CLR SND.ENVELOPE+6(R0)	:	3839
000070	005060	000010G	CLR SND.ENVELOPE+10(R0)	:	3840
000074	005060	000012G	CLR SND.ENVELOPE+12(R0)	:	3841

000100	112760	000004	000014G	MOV	#4,SND.ENVELOPE+14(R0)	:	3842
000106	105060	000015G		CLRB	SND.ENVELOPE+15(R0)	:	3843
000112	005060	000016G		CLR	SND.ENVELOPE+16(R0)	:	3844
000116	005060	000020G		CLR	SND.ENVELOPE+20(R0)	:	3848
000122	005060	000022G		CLR	SND.ENVELOPE+22(R0)	:	3849
000126	005060	000024G		CLR	SND.ENVELOPE+24(R0)	:	3850
000132	005060	000026G		CLR	SND.ENVELOPE+26(R0)	:	3851
000136	005060	000030G		CLR	SND.ENVELOPE+30(R0)	:	3852
000142	005060	000032G		CLR	SND.ENVELOPE+32(R0)	:	3853
000146	005060	000034G		CLR	SND.ENVELOPE+34(R0)	:	3854
000152	005060	000036G		CLR	SND.ENVELOPE+36(R0)	:	3855
000156	005060	000040G		CLR	SND.ENVELOPE+40(R0)	:	3856
000162	005060	000042G		CLR	SND.ENVELOPE+42(R0)	:	3857
000166	010216			MOV	R2,(SP)	: REF.NUM,*	3866
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,*	3868
000202	001002			BNE	1\$		
000204	004737	000232'		JSR	PC,DECODE		
000210	013700	000000G	1\$:	MOV	NSD.SLOT,R0	:	3874
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)	: *,RCSS.REG	3878
000240	016600	000004		MOV	4(SP),R0	: RCSS.REG,TEMP	
000244	012716	005670		MOV	#5670,(SP)	:	3897
000250	010246			MOV	R2,-(SP)	: REF.NUM,*	
000252	010146			MOV	R1,-(SP)	: SCC.BUF\$LOC,*	
000254	004737	002074'		JSR	PC,CTO.WAIT		
000260	022626			CMP	(SP)+,(SP)+		
000262	006000			ROR	R0		
000264	103002			BCC	2\$		
000266	004737	000232'		JSR	PC,DECODE		
000272	010100		2\$:	MOV	R1,R0	: SCC.BUF\$LOC,*	3904
000274	006300			ASL	R0		
000276	006300			ASL	R0		
000300	016037	000002G	000000G	MOV	OUT\$STD.BUF+2(R0),RET.ENSAD		
000306	016000	000002G		MOV	OUT\$STD.BUF+2(R0),R0	:	3909
000312	005760	000016		TST	16(R0)		
000316	001405			BEQ	3\$		
000320	012700	000031		MOV	#31,R0	:	3911
000324	010037	000000G		MOV	R0,RET.STATUS		
000330	000402			BR	4\$:	3813
000332	013700	000000G	3\$:	MOV	RET.STATUS,R0		
000336	062706	000006	4\$:	ADD	#6,SP	:	3778
000342	000207			RTS	PC		

: Routine Size: 114 words, Routine Base: \$CODE\$ + 4472
: Maximum stack depth per invocation: 9 words

: 3916

```

C 3917  %(  

C 3918  global routine ON_LINE =          !Makes a unit come online to a host  

C 3919  

C 3920  !++  

C 3921  Functional Description :  

C 3922  The online command is used to bring a unit 'unit-online, set  

C 3923  host settable unit characteristics and obtain those unit  

C 3924  characteristics that are essential for proper class driver  

C 3925  operation. The unit is spun-up, if necessary, and is heads  

C 3926  are loaded prior to returning the online command's end  

C 3927  message. Host settable characteristics are set exactly as if  

C 3928  a set unit characteristics command were issued. Host settable  

C 3929  characteristics are set after the unit has been successfully spun-up  

C 3930  and any other validity checks have succeeded. Note that the unit's  

C 3931  host settable characteristics are not altered if the unit is already  

C 3932  'unit-online'.  

C 3933  

C 3934  Formal Parameters :  

C 3935  none  

C 3936  

C 3937  Implicit Inputs :  

C 3938  NSD_SLOT          This global storage gets loaded by the routine  

C 3939                   'Get_nsd' and in it is stored the next send ring  

C 3940                   descriptor slot where the port/controller should  

C 3941                   be polling on and the place to put this commands  

C 3942                   command packet.  

C 3943  

C 3944  Implicit Outputs :  

C 3945  none  

C 3946  

C 3947  Completion Codes :  

C 3948  RET_STATUS:      Return status passes back to the calling routine  

C 3949                   the status of the just issued command.  

C 3950  

C 3951  Side Effects :  

C 3952  Any previously defined controller characteristics will possibly  

C 3953  be altered after execution of this command.  

C 3954  --  

C 3955  

C 3956  begin  

C 3957  

C 3958  local  

C 3959  REF_NUM,          !Stores unique cmd ref number  

C 3960  ONL_BUF$LOC,     !Stores outstanding cmd buffer location  

C 3961  TEMP;           !A place to put read IP register data  

C 3962  

C 3963  |  

C 3964  | Before we load up the command packet up  

C 3965  | with all this good information get the  

C 3966  | next send descriptor slot and a unique  

C 3967  | command reference number.  

C 3968  |  

C 3969  GET_NSD ();      !Get the next send desc slot
```

```

C 3970 REF_NUM = GET_CMD$REF (); !Get a unique command ref num
C 3971
C 3972 ! UQ Port command envelope Header field definition
C 3973
C 3974 SND_ENVELOPE [.NSD_SLOT, MSG_LENGTH] = SZ_ONL; !Load message length
C 3975 SND_ENVELOPE [.NSD_SLOT, CREDITS] = ONE; !Load credit size
C 3976 SND_ENVELOPE [.NSD_SLOT, MSG_TYPE] = 0; !Define message type 'Sequential'
C 3977 SND_ENVELOPE [.NSD_SLOT, CONN_ID] = 0; !Define connection ID
C 3978
C 3979 ! MSCP generic command envelope field definition
C 3980
C 3981 SND_ENVELOPE [.NSD_SLOT, CMD_LREF] = .REF_NUM; !Load command reference number
C 3982 SND_ENVELOPE [.NSD_SLOT, CMD_HREF] = ZERO; !Zero Hi order cmd ref num
C 3983 SND_ENVELOPE [.NSD_SLOT, UNIT_NUM] = .UNIT_NO; !Select unit to bring online
C 3984 SND_ENVELOPE [.NSD_SLOT, UN_HUSED] = ZERO; !Not used in DUP implimentation
C 3985 SND_ENVELOPE [.NSD_SLOT, OPCODE] = OP_ONL; !Load this commands op-code
C 3986 SND_ENVELOPE [.NSD_SLOT, RSVD] = ZERO; !Not used field
C 3987 SND_ENVELOPE [.NSD_SLOT, MODIFIER] = ZERO; !Define this commands modifiers
C 3988
C 3989 ! Command specific command envelope field definition
C 3990
C 3991 SND_ENVELOPE [.NSD_SLOT, RSV$D] = ZERO; !Reserved
C 3992 SND_ENVELOPE [.NSD_SLOT, UNT_FLAGS] = ZERO; !Unit flag field
C 3993 SND_ENVELOPE [.NSD_SLOT, RSV$D0] = ZERO; !Reserved field
C 3994 SND_ENVELOPE [.NSD_SLOT, RSV$D1] = ZERO; !Reserved field
C 3995 SND_ENVELOPE [.NSD_SLOT, RSV$D2] = ZERO; !Reserved field
C 3996 SND_ENVELOPE [.NSD_SLOT, RSV$D3] = ZERO; !Reserved field
C 3997 SND_ENVELOPE [.NSD_SLOT, RSV$D4] = ZERO; !Reserved field
C 3998 SND_ENVELOPE [.NSD_SLOT, RSV$D5] = ZERO; !Reserved field
C 3999 SND_ENVELOPE [.NSD_SLOT, DDP_LO] = ZERO; !Device dependent parameter
C 4000 SND_ENVELOPE [.NSD_SLOT, DDP_HI] = ZERO; !Device dependent parameter
C 4001 SND_ENVELOPE [.NSD_SLOT, SHADOW_UNIT] = ZERO; !Shadow unit
C 4002 SND_ENVELOPE [.NSD_SLOT, COPY_SPEED] = ZERO; !Copy speed
C 4003
C 4004 ! Call the load outstanding command buffer routine
C 4005 ! and load this command into the buffer. The return
C 4006 ! from this routine will point us to the buffer location
C 4007 ! where this command is stored. Later we can look at
C 4008 ! this location to see if the interrupt service routine
C 4009 ! has received and process it.
C 4010
C 4011 ONL_BUF$LOC = LOAD_OUT$STD_BUF (.REF_NUM); !Load the command
C 4012
C 4013 if .ONL_BUF$LOC eqlu OBF_CODE then DECODE (); !Error if buffer is full
C 4014
C 4015 !
C 4016 ! Set the ownership bit to 1 giving this slot
C 4017 ! to the port/controller
C 4018
C 4019 SEND_RING [.NSD_SLOT, OWN_BIT] = PORT_OWNED;
C 4020
C 4021 ! Read the IP register to stimulate port polling
C 4022

```

```

C 4023     TEMP = .RDRX_ADDR [RCIP, RC_ALL];
C 4024     |
C 4025     | Time out the port/controller processing the command.
C 4026     |
C 4027     | The first test tests the connections ability to
C 4028     | respond to this command without any errors in the SA
C 4029     | register and for the command not timing out.
C 4030     |
C 4031     | The second tests the DUP server for good status. If
C 4032     | bad status is sent back then an error code is returned
C 4033     | to the calling routine where the routine "decode" will
C 4034     | decode and take the appropriate recovery. The time
C 4035     | out routine will loop on delaying and checking the hi
C 4036     | bit of the first word in the out$std_buf for a true.
C 4037     | When true signals us that the interrupt service routine
C 4038     | has received the endpacket and no connection errors
C 4039     | were detected.
C 4040     |
C 4041     |
C 4042     | if CTO_WAIT (3000, .REF_NUM, .ONL_BUF$LOC) then DECODE (); !Is return an error
C 4043     |
C 4044     |
C 4045     | Get the return envelope address from the out$std_buf
C 4046     | at this commands buffer location and check the packet
C 4047     | for good status error and die if bad status was returned
C 4048     |
C 4049     | RET_EN$AD = .OUT$STD_BUF [.ONL_BUF$LOC, ENV_ADR]; !Get the ret env adr
C 4050     |
C 4051     | Now test for good status
C 4052     |
C 4053     |
C 4054     | if .RET_EN$AD [STATUS] nequ ZERO           !Test the status
C 4055     | then
C 4056     |     return RET_STATUS = RSE_CODE         !Return a "Response status err" code
C 4057     | else
C 4058     |     return .RET_STATUS;                   !This ret_status is good or bad
C 4059     |
C 4060     | end;
C 4061     |)%

```

```

: 4062 global routine INT$I_SERVICE : INT_LNK$TYP novalue = !Init sequence interrupt catcher
: 4063
: 4064 |++
: 4065 | Functional Description :
: 4066 | During the initialization sequence the IE bit is defined to be
: 4067 | a zero. This means that the host is not requesting interrupts at
: 4068 | the completion of steps 1-3.
: 4069 |
: 4070 | Note that no interrupt will be generated at the completion of
: 4071 | step 4 since this step requires only a small number of time.
: 4072 |
: 4073 | This interrupt service routine serves to catch any interrupts that the
: 4074 | controller might issue during the initialization sequence. The
: 4075 | interrupt is ignored and control is returned.
: 4076 |
: 4077 | Formal Parameters :
: 4078 | none
: 4079 |
: 4080 | Implicit Inputs :
: 4081 | none
: 4082 |
: 4083 | Implicit Outputs :
: 4084 | none
: 4085 |
: 4086 | Completion Codes :
: 4087 | none
: 4088 |
: 4089 | Side Effects :
: 4090 | none
: 4091 |--
: 4092
: 4093 begin
: 4094 return;
: 4095 end;

```

```

000000 000002 .SBTTL INT$I.SERVICE GLOBAL ROUTINE DECLARATIONS
INT$I.SERVICE:: RTI ; 4062

; Routine Size: 1 word, Routine Base: $CODE$ + 5036
; Maximum stack depth per invocation: 0 words

; 4096

```

```

4097 global routine IS_TIMER (SEQ_NO) =      !Init sequence time out
4098
4099 !++
4100 ! Functional Description :
4101 !   Steps 1-3 of the init sequence, each are required to
4102 !   complete within 10 seconds. If any of these steps
4103 !   fails to complete within that period, this is to be
4104 !   treated as a host detected fatal error.
4105
4106 !   This routine will do one us delays for a total of 10
4107 !   seconds. After each delay the step field is examined
4108 !   to see if this init sequence has completed.
4109
4110 ! Formal Parameters :
4111 !   SEQ_NO: indicated which init step is presently being
4112 !   performed within the RDRX init sequence.
4113
4114 ! Implicit Inputs :
4115 !   none
4116
4117 ! Implicit Outputs :
4118 !   none
4119
4120 ! Completion Codes :
4121 !   TRUE: Indicates to the calling routine that the
4122 !   indicated init sequence step has timed out.
4123
4124 !   FALSE: Indicates to the calling routine that the
4125 !   indicated init sequence step has not timed
4126 !   out.
4127
4128 ! Side Effects :
4129 !   If the init sequence step times out and an error
4130 !   is posted in the sa register then the routine
4131 !   decode will be call.
4132 !--
4133
4134 begin
4135
4136 local
4137     STEP_VAL : word;                !Temp storage of step value
4138
4139 STEP_VAL = ZERO;                   !Make sure the loc is zeroed out
4140
4141 !+
4142 ! Select the step value expected from
4143 ! this init sequence step.
4144 !-
4145
4146 selectoneu .SEQ_NO of              !Select the binary step value
4147     set
4148
4149     [0] :

```

ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

K 13

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (22)

SEQ 166
Page 79

```
: 4150          STEP_VAL = %b'0001';          !Step 1 binary value
: 4151
: 4152          [1] : STEP_VAL = %b'0010';          !Step 2 binary value
: 4153
: 4154          [2] : STEP_VAL = %b'0100';          !Step 3 binary value
: 4155
: 4156          [3] : STEP_VAL = %b'1000';          !Step 4 binary value
: 4157
: 4158          tes;
: 4159
: 4160
: 4161
: 4162          !+
: 4163          ! Loop on the 100 micro second delay until
: 4164          ! either the expected step field is read in
: 4165          ! the SA register or the step times out.
: 4166          !-
: 4167
: 4168          incru TIM_OUT from 0 to 15000 do          !Delay for 10 seconds
: 4169          begin
: 4170          DELAY (C_US);          !Do the delay
: 4171
: 4172          ! Check the step bit to see if it is set yet.
: 4173          ! If it is set then return a false indicating
: 4174          ! the completion else continue delaying.
: 4175          !
: 4176
: 4177          if .RDRX_ADDR [RCSA, STP_FIELD] eqlu .STEP_VAL then return FALSE;
: 4178
: 4179          BREAK;          !Service any control C's
: 4180          end;
: 4181
: 4182          !+
: 4183          ! This step has not completed within the specified
: 4184          ! 10 second time interval. Test the sa register
: 4185          ! for any errors posted and report errors if any.
: 4186          ! Return a true to the caller indicating the error.
: 4187          !-
: 4188
: 4189          if .RDRX_ADDR [RCSA, ERR_BIT]          !Is the error bit set
: 4190          then
: 4191          begin
: 4192          RET_STATUS = PFE_CODE;          !Indicate the port/fatal error code
: 4193          DECODE ();          !Report the error
: 4194          end;
: 4195
: 4196          return TRUE;          !Return a failure to the caller
: 4197          end;
```

000000 004137 000000G

```
.SBTTL IS.TIMER GLOBAL ROUTINE DECLARATIONS
IS.TIMER::
JSR R1,$SAVE3 ;
```

4097

ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (22)

000004	162706	000006		SUB	#6,SP				
000010	005002			CLR	R2	:	STEP.VAL		4139
000012	016600	000020		MOV	20(SP),R0	:	SEQ.NO,*		4146
000016	001003			BNE	1\$				
000020	012702	000001		MOV	#1,R2	:	*,STEP.VAL		4150
000024	000421			BR	4\$:			4146
000026	020027	000001	1\$:	CMP	R0,#1				
000032	001003			BNE	2\$				
000034	012702	000002		MOV	#2,R2	:	*,STEP.VAL		4153
000040	000413			BR	4\$:			4146
000042	020027	000002	2\$:	CMP	R0,#2				
000046	001003			BNE	3\$				
000050	012702	000004		MOV	#4,R2	:	*,STEP.VAL		4156
000054	000405			BR	4\$:			4146
000056	020027	000003	3\$:	CMP	R0,#3				
000062	001002			BNE	4\$				
000064	012702	000010		MOV	#10,R2	:	*,STEP.VAL		4159
000070	005003		4\$:	CLR	R3	:	TIM.OUT		4168
000072	012701	000001	5\$:	MOV	#1,R1	:	*,SSTMP2		4170
000076	001411		6\$:	BEQ	9\$				
000100	013700	000000G		MOV	LSDLY,R0	:	*,SSTMP1		
000104	001404			BEQ	8\$				
000106	005066	000004	7\$:	CLR	4(SP)	:	SSTMP		
000112	005300			DEC	R0	:	SSTMP1		
000114	001374			BNE	7\$				
000116	005301		8\$:	DEC	R1	:	SSTMP2		
000120	000766			BR	6\$				
000122	013700	000000G	9\$:	MOV	RDRX.ADDR,R0	:			4177
000126	016066	000002 000002		MOV	2(R0),2(SP)	:	*,RCSS.REG		
000134	010201			MOV	R2,R1	:	STEP.VAL,*		
000136	016600	000002		MOV	2(SP),R0	:	RCSS.REG,*		
000142	006200			ASR	R0				
000144	006200			ASR	R0				
000146	006200			ASR	R0				
000150	000300			SWAB	R0				
000152	042700	177760		BIC	#177760,R0				
000156	020001			CMP	R0,R1				
000160	001422			BEQ	11\$				
000162	104422			TRAP	22				
000164	005203			INC	R3	:	TIM.OUT		4168
000166	020327	035230		CMP	R3,#35230	:	TIM.OUT,*		
000172	101737			BLOS	5\$				
000174	013700	000000G		MOV	RDRX.ADDR,R0	:			4189
000200	016016	000002		MOV	2(R0),(SP)	:	*,RCSS.REG		
000204	100005			BPL	10\$				
000206	012737	000021 000000G		MOV	#21,RET.STATUS	:			4192
000214	004737	000232		JSR	PC,DECODE	:			4193
000220	012700	000001	10\$:	MOV	#1,R0	:			4134
000224	000401			BR	12\$				
000226	005000		11\$:	CLR	R0	:			4097
000230	062706	000006	12\$:	ADD	#6,SP				
000234	000207			RTS	PC				

M 13

ZRQB4 RDRX DISK FORMATTER
REV A PATCH 00 GLOBAL ROUTINE DECLARATIONS

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (22)

SEQ 168
Page 81

: Routine Size: 79 words, Routine Base: \$CODES + 5040
: Maximum stack depth per invocation: 9 words

: 4198

```
4199 global routine BOOT_RDRX =      !Performs RDRX init sequence
4200
4201 !++
4202 | Functional Description :
4203 |   This routine performs the initialization sequence of the RDRX
4204 |   RDRX controller.
4205 |
4206 |   The initialization procedure serves to:
4207 |
4208 |   1. Identify the parameters of the host-resident communications
4209 |      region to the port.
4210 |
4211 |   2. Provide a confidence check of port/controller integrity.
4212 |
4213 |   3. Bring the port/controller online to the host (note that the
4214 |      devices attached to the controller are not thereby brought
4215 |      online to the class driver.)
4216 |
4217 | Formal Parameters :
4218 |   none
4219 |
4220 | Implicit Inputs :
4221 |   ISD_STRUCT      Stores the init sequence read and write data defined
4222 |   for this program and controller.
4223 |
4224 | Implicit Outputs :
4225 |   none
4226 |
4227 | Completion Codes :
4228 |   Success: Is returned to the calling routine if this initialization
4229 |            sequence was executed successfully.
4230 |
4231 |   Failure: Is returned to the calling routine if this initialization
4232 |            sequence was not executed successfully.
4233 |
4234 | Side Effects :
4235 |   Any DM code that might have been running in the DM machine will be
4236 |   aborted.
4237 |
4238 |   Any outstanding commands or response pertaining to a process using
4239 |   the controller will be lost.
4240 | --
4241 |
4242 | begin
4243 |
4244 | local
4245 |   TEMP : word;          !Temporary storage location
4246 |
4247 | !+
4248 | | The host begins the initialization sequence
4249 | | either by issuing a bus init or by writing
4250 | | any value into the IP register; the port must
4251 | | guarantee that the host will read zeros in SA
```

```
4252 | on the next bus cycle. Initialization then
4253 | sequences through steps 1-4 as per UQSSP.DOC
4254 | Version 1.5.
4255 |
4256 | Write to the IP register and start the init
4257 | sequence going.
4258 | -
4259 |
4260 WRT_RDRX (RCIP, RC_ALL, ONES);           !Begin init sequence
4261 |
4262 | +
4263 | This incr loop performs all four steps of the
4264 | initialization sequence described above. The
4265 | SA write and read data is preset into the
4266 | structure ISD_STRUCT and stands for
4267 | "Initialization Sequence Data_STRUCT".
4268 | -
4269 |
4270 | If a step time out error occurs the test
4271 | invoking this routine will take the necessary
4272 | retry procedure. A return code of failure is
4273 | returned.
4274 |
4275 | If any SA register compare error is detected after
4276 | a step completion the routine Decode will decode the
4277 | error and load statistical tables up pertanate data.
4278 | -
4279 |
4280 |
4281 incru SEQ_NO from STEP1 to STEP4 do       !Do the four init seq steps
4282 | begin
4283 | |
4284 | | Wait for the controller to load the SA reg
4285 | | up with the step data.
4286 | |
4287 | |
4288 | | if IS_TIMER (.SEQ_NO)                   !Did the Controller time out
4289 | | then
4290 | | | begin
4291 | | | |
4292 | | | | DO SOME STAT TABLE UP DATA TO SHOW
4293 | | | | THE TIME OUT
4294 | | | |
4295 | | | | PRINTF (.EMSG_STRUCT [MSG10]);
4296 | | | | return FAILURE;                     !Notify DRS> init of the failure
4297 | | | | end;
4298 | | |
4299 | | |
4300 | | | The controller did not time out so read the SA register
4301 | | | for the expected step data and compare it to the good
4302 | | | data stored in ISD_STRUCT.
4303 | | |
4304 | | | If the read SA data is not what we expect then return a
```

```
4305      | failure code.
4306      |
4307      | Note that the reserved fields read in the SA register are
4308      | or'ed with all ones to mask out the field before compared
4309      | to the expected data stored in the structure "ISD_STRUCT".
4310      |
4311      | TEMP = ((.RDRX_ADDR [RCSA, RC_ALL]) or (.RSVD_STRUCT [.SEQ_NO]));
4312      |
4313      | if .TEMP nequ .ISD_STRUCT [.SEQ_NO, ISRD, ISR_ALL]      !Compare read to expected
4314      | then
4315      |     begin
4316      |         |
4317      |         | Load some satistical table up with
4318      |         | some data to indicate that the init
4319      |         | sequence had some trouble.
4320      |         |
4321      |         | PRINTF (.EMSG_STRUCT [MSG11]);
4322      |         | return FAILURE;                          !Return a failure code
4323      |         | end;
4324      |         |
4325      |         |
4326      |         | We need to save the micro code
4327      |         | version number if this is step4.
4328      |         | Save the ucode version in UC_VER
4329      |         | if this is step4.
4330      |         |
4331      |         |
4332      |         | if .SEQ_NO eqlu STEP4                      !Is this seq step 4
4333      |         | then
4334      |         |     begin
4335      |         |         UC_VER = .RDRX_ADDR [RCSA, S4R_UVER];      !Save the ucode version
4336      |         |     end;
4337      |         |
4338      |         |
4339      |         | This step read data is what we expected so
4340      |         | write the SA register with this steps write
4341      |         | data stored in ISD_STRUCT.
4342      |         |
4343      |         | WRT_RDRX (RCSA, RC_ALL,                      !Write the SA register
4344      |         |         .ISD_STRUCT [.SEQ_NO, ISWRT, ISW_ALL]);  !Locate the data
4345      |         | end;
4346      |         |
4347      |         | !+
4348      |         | The controller initialization sequence was
4349      |         | done successfully so return a success code.
4350      |         | !-
4351      |         |
4352      |         | return SUCCESS;
4353      |         | end;
```

ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (23)

000004	162706	000010		JSR	R1,\$SAVE4	:	4199
000010	017766	000000G	00000S	SUB	#10,SP	:	
000016	012700	177777		MOV	@RDRX.ADDR,6(SP)	: *,RCSS.REG	4260
000022	010077	000000G		MOV	#-1,R0	: *,RCSM.REG	
000026	005003			MOV	R0,@RDRX.ADDR	: RCSM.REG,*	
000030	010346			CLR	R3	: SEQ.NO	4281
000032	004737	005040'		MOV	R3,-(SP)	: SEQ.NO,*	4288
000036	005726			JSR	PC,IS.TIMER		
000040	006000			TST	(SP)+		
000042	103007			ROR	R0		
000044	013746	000024G		BCC	2\$		
000050	012746	000001		MOV	EMSG.STRUCT+24,-(SP)	:	4295
000054	010600			MOV	#1,-(SP)		
000056	104417			MOV	SP,R0	: SP,*	
000060	000427			TRAP	17		
000062	013700	000000G		BR	3\$:	4288
000066	016066	000002	000004	MOV	RDRX.ADDR,R0	:	4311
000074	010300			MOV	2(R0),4(SP)	: *,RCSS.REG	
000076	006300			MOV	R3,R0	: SEQ.NO,*	
000100	016604	000004		ASL	R0		
000104	056004	000000G		MOV	4(SP),R4	: RCSS.REG,TEMP	
000110	010302			BIS	RSVD.STRUCT(R0),R4	: *,TEMP	
000112	006302			MOV	R3,R2	: SEQ.NO,*	4313
000114	006302			ASL	R2		
000116	020462	000000G		ASL	R2		
000122	001410			MOV	R4,ISD.STRUCT(R2)	: TEMP,*	
000124	013746	000026G		CMP	4\$		
000130	012746	000001		BEQ	4\$		
000134	010600			MOV	EMSG.STRUCT+26,-(SP)	:	4321
000136	104417			MOV	#1,-(SP)		
000140	022626			MOV	SP,R0	: SP,*	
000142	000432			TRAP	17		
000144	020327	000003		CMP	(SP)+,(SP)+	:	4313
000150	001010			BR	6\$:	4315
000152	013700	000000G		CMP	R3,#3	: SEQ.NO,*	4332
000156	016066	000002	000002	BNE	5\$		
000164	116637	000002	000000G	MOV	RDRX.ADDR,R0	:	4335
000172	013700	000000G		MOV	2(R0),2(SP)	: *,RCSS.REG	
000176	016016	000002		MOVB	2(SP),UC.VER	: RCSS.REG,*	
000202	016201	000002G		MOV	RDRX.ADDR,R0	:	4344
000206	010160	000002		MOV	2(R0),(SP)	: *,RCSS.REG	
000212	005203			MOV	ISD.STRUCT+2(R2),R1	: *,RCSM.REG	
000214	020327	000003		MOV	R1,2(R0)	: RCSM.REG,*	
000220	101703			INC	R3	: SEQ.NO	4281
000222	012700	000001		CMP	R3,#3	: SEQ.NO,*	
000226	000401			BLOS	1\$		
000230	005000			MOV	#1,R0	:	4242
000232	062706	000010		BR	7\$		
000236	000207			CLR	R0	:	4199
				ADD	#10,SP		
				RTS	PC		

; Routine Size: 80 words, Routine Base: \$CODE\$ + 5276
; Maximum stack depth per invocation: 13 words

ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

: 4354

E 14

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (23)

SEQ 173
Page 86

4355 global routine INIT_COM_AREA = !Inits DUP Protocol communication area

4356

4357

4358

4359

4360

4361

4362

4363

4364

4365

4366

4367

4368

4369

4370

4371

4372

4373

4374

4375

4376

4377

4378

4379

4380

4381

4382

4383

4384

4385

4386

4387

4388

4389

4390

4391

4392

4393

4394

4395

4396

4397

4398

4399

4400

4401

4402

4403

4404

4405

4406

4407

!++

Functional Description :

After initialization step 3 the port controller clears out the communication area's ring buffers.

This routine first makes sure that this protocol is accomplished by the port before proceeding.

If the port did its part of the protocol then the communications area is initialized as follows:

1. Defines from the contiguous data storage structure "COM_AREA" the header area address, receive ring address and the send ring address (these structures are initially declared as reference structures and require an address to be defined as its value per BLISS language conventions).
2. Clears the interrupt indicators and adaptor purge (ring base -1, -2, -3, -4) defined as 'HEAD_AREA'.
3. Loads the receive and send descriptors with the values:
 - a. Envelope low, high and Q_bus address
 - b. Reserved field
 - c. Flag bit
 - d. Ownership bit
4. Load the receive envelope message length field with the buffer size in bytes.
5. Initialize the Outstanding command buffer to reflect that all slots are unused.

Formal Parameters :

none

Implicit Inputs :

HEAD_AREA, RECEIVE_RING, SEND_RING, COM_AREA

Implicit Outputs :

The communication area as a result of this routine will be initialized for host program to remote program communications per DUP and UQSSP specifications.

Completion Codes :

TRUE: Error code to indicate the port controller has not fulfilled its part of the DUP protocol.

FALSE: An error code to indicate the port controller has fulfilled its part of the DUP protocol.

ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

G 14

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (24)

SEQ 175
Page 88

```
4408 |
4409 | Side Effects :
4410 | none
4411 | --
4412 |
4413 | begin
4414 |
4415 | +
4416 | Make sure that the controller has done its part of
4417 | the DUP protocol by clearing out the ring buffers.
4418 | If the rings are not cleared out then return with
4419 | an error code of true.
4420 | -
4421 |
4422 | incru i from 2 to RING_SIZE - 1 do           !Test all blocks for zeros
4423 |     incru j from WRD0 to WRD1 do           !Test all words for zeros
4424 |         Test this word for zeros.  If not zeros then exit
4425 |         this routine with an "communication area init"
4426 |         error code to indicate the Protocol violation.
4427 |
4428 |
4429 |
4430 |
4431 |     if .COM_AREA [.i, .j, WORD_REF] nequ ZERO then return CIE_CODE;
4432 |
4433 | +
4434 | The port did its part of the protocol so now
4435 | define the address locations of the HEAD_AREA,
4436 | RECEIVE_RING and SEND_RING from the contiguous
4437 | storage declared by COM_AREA.
4438 | -
4439 |
4440 | HEAD_AREA = COM_AREA;                       !Define the Header area
4441 | RECEIVE_RING = COM_AREA [REC_BASE];         !Define the receive ring area
4442 | SEND_RING = COM_AREA [SND_BASE];           !Define the send ring area
4443 |
4444 | +
4445 | Not quite sure if the port has to clear out
4446 | the header area of the communications area
4447 | so I'll clear it out here just in case.
4448 | -
4449 |
4450 | incru i from WRD0 to WRD3 do
4451 |     HEAD_AREA [.i, WORD_REF] = ZEROS;
4452 |
4453 | +
4454 | Load up the Send Ring descriptors with an envelope address,
4455 | define the "Flag bit" to = 1 (interrupt requested), define
4456 | the "Ownership bit" to = 0 (owned by host) and load the
4457 | Reserved field with zeros (per DUP spec).
4458 | -
4459 |
4460 | incru i from 0 to SND_ALLOCATE - 1 do
```

```

4461     begin
4462     SEND_RING [.i, LO_EN$AD] = SND_ENVELOPE [.i, CMD_LREF]; !Low-order envelope adress for all sys
4463     SEND_RING [.i, HI_EN$AD] = ZERO;           !High-order portion of an 18-bit U/Q bus adrs
4464     SEND_RING [.i, QB_EXT] = ZERO;           !Q_bus extention
4465     SEND_RING [.i, D$RSVD] = ZERO;           !Reserved field
4466     SEND_RING [.i, FLAG_BIT] = SET_FLG;       !Flag bit whose meaning varies depending on dsc state
4467     SEND_RING [.i, OWN_BIT] = HOST_OWNED;     !Indicates whether dsc is host or port owned
4468     end;

```

```

4469
4470     !+
4471     ! Load up the Receive Ring descriptors with an envelope
4472     ! address, define the 'Ownership bit' = 1 (owned by port),
4473     ! define the 'Flag bit' to = 1 (Interrupts requested) and
4474     ! the reserved field set to zeros (per DUP spec).
4475     !-

```

```

4476
4477     incru i from 0 to REC_ALLOCATE - 1 do
4478     begin
4479     RECEIVE_RING [.i, LO_EN$AD] = REC_ENVELOPE [.i, CMD_LREF];
4480     RECEIVE_RING [.i, HI_EN$AD] = ZEROS;
4481     RECEIVE_RING [.i, QB_EXT] = ZEROS;
4482     RECEIVE_RING [.i, D$RSVD] = ZEROS;
4483     RECEIVE_RING [.i, FLAG_BIT] = SET_FLG;
4484     RECEIVE_RING [.i, OWN_BIT] = PORT_OWNED;
4485     end;

```

```

4486
4487     !+
4488     ! Reset the communications area pointer to
4489     ! their initial state.
4490     !-

```

```

4491     NRD_SLOT = -1;           !Start ring pointer at zero
4492     NSD_SLOT = -1;           !Start ring pointer at zero
4493     NXT_CRN = ZERO;         !Start unique cmd ref num at one

```

```

4494
4495     !+
4496     ! Set the response envelope message length size equal
4497     ! to the buffer size in bytes starting at text + 0.
4498     !-

```

```

4499
4500     incru i from 0 to REC_ALLOCATE - 1 do
4501     REC_ENVELOPE [.i, MSG_LENGTH] = RB_SIZE*2;           !Convert to bytes before loading

```

```

4502
4503     !+
4504     ! Init the outstanding command buffer as follows:
4505     ! 1. Indicate that all slots are unused by loading
4506     !    the unique value %o'100000'.
4507     ! 2. Clear the envelope adrs words to zero.
4508     !-

```

```

4509
4510     incru i from 0 to REC_ALLOCATE - 1 do
4511     begin
4512     OUT$STD_BUF [.i, CMD_WRD] = %o'100000'; !Define the slot as unused
4513     end;

```

ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (24)

```

:      4514      OUT$STD_BUF [.i, ENV_ADR] = ZERO;
:      4515      end;
:      4516
:      4517      !+
:      4518      ! No errors detected by this routine so
:      4519      ! return with an non-error code of false.
:      4520      !-
:      4521
:      4522      return PAS_CODE;
:      4523      end;

```

!Clear out the envelope adrs field

		.SBTTL INIT.COM.AREA GLOBAL ROUTINE DECLARATIONS		
000000	004137	000000G	INIT.COM.AREA::	
			JSR R1,\$SAVE3	4355
000004	012701	000004	MOV #4,R1	4422
000010	005002		1\$: CLR R2	4424
000012	010100		2\$: MOV R1,R0	4431
000014	060200		ADD R2,R0	
000016	006300		ASL R0	
000020	005760	000000G	TST COM.AREA(R0)	
000024	001403		BEQ 3\$	
000026	012700	000001	MOV #1,R0	
000032	000207		RTS PC	
000034	005202		3\$: INC R2	4424
000036	020227	000001	CMP R2,#1	
000042	101763		BLOS 2\$	
000044	062701	000002	ADD #2,R1	4422
000050	020127	000022	CMP R1,#22	
000054	101755		BLOS 1\$	
000056	012737	000000G 000000G	MOV #COM.AREA,HEAD.AREA	4440
000064	012737	000010G 000000G	MOV #COM.AREA+10,RECEIVE.RING	4441
000072	012737	000030G 000000G	MOV #COM.AREA+30,SEND.RING	4442
000100	005000		CLR R0	4450
000102	010001		4\$: MOV R0,R1	4451
000104	063701	000000G	ADD HEAD.AREA,R1	
000110	005011		CLR (R1)	
000112	062700	000002	ADD #2,R0	4450
000116	020027	000006	CMP R0,#6	
000122	101767		BLOS 4\$	
000124	005003		CLR R3	4460
000126	010301		5\$: MOV R3,R1	4462
000130	006301		ASL R1	
000132	006301		ASL R1	
000134	010102		MOV R1,R2	
000136	063702	000000G	ADD SEND.RING,R2	
000142	010346		MOV R3,-(SP)	: I,*
000144	012746	000054	MOV #54,-(SP)	
000150	004737	000000G	JSR PC,BLSMUL	
000154	062700	000004G	ADD #SND.ENVELOPE+4,R0	
000160	010012		MOV R0,(R2)	
000162	010100		MOV R1,R0	
000164	063700	000000G	ADD SEND.RING,R0	4463

ZRQB4
REV A PATCH 00

RDRX DISK FORMATTER
GLOBAL ROUTINE DECLARATIONS

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (24)

000170	062700	000002		ADD	#2,R0				
000174	012710	040000		MOV	#40000,(R0)	:			4467
000200	022626			CMP	(SP)+,(SP)+	:			4461
000202	005203			INC	R3	:	I		4460
000204	020327	000003		CMP	R3,#3	:	I,*		
000210	101746			BLOS	5\$				
000212	005002			CLR	R2	:	I		4477
000214	010201		6\$:	MOV	R2,R1	:	I,*		4479
000216	006301			ASL	R1				
000220	006301			ASL	R1				
000222	010103			MOV	R1,R3				
000224	063703	000000G		ADD	RECEIVE.RING,R3				
000230	010200			MOV	R2,R0	:	I,*		
000232	000300			SWAB	R0				
000234	106000			RORB	R0				
000236	006000			ROR	R0				
000240	006000			ROR	R0				
000242	142700	000077		BICB	#77,R0				
000246	062700	000004G		ADD	#REC.ENVELOPE+4,R0				
000252	010013			MOV	R0,(R3)				
000254	010100			MOV	R1,R0	:			4480
000256	063700	000000G		ADD	RECEIVE.RING,R0				
000262	062700	000002		ADD	#2,R0				
000266	012710	140000		MOV	#140000,(R0)	:			4484
000272	005202			INC	R2	:	I		4477
000274	020227	000003		CMP	R2,#3	:	I,*		
000300	101745			BLOS	6\$				
000302	012737	177777	000000G	MOV	#-1,NRD.SLOT	:			4492
000310	012737	177777	000000G	MOV	#-1,NSD.SLOT	:			4493
000316	105037	000000G		CLRB	NXT.CRN	:			4494
000322	005000			CLR	R0	:	I		4501
000324	012760	000074	000000G	MOV	#74,REC.ENVELOPE(R0)	:	*,*(I)		4502
000332	062700	000100		ADD	#100,R0	:	*,I		4501
000336	020027	000300		CMP	R0,#300	:	I,*		
000342	101770			BLOS	7\$				
000344	005000			CLR	R0	:	I		4511
000346	012760	100000	000000G	MOV	#-100000,OUT\$STD.BUF(R0)	:	*,*(I)		4513
000354	005060	000002G		CLR	OUT\$STD.BUF+2(R0)	:	*(I)		4514
000360	062700	000004		ADD	#4,R0	:	*,I		4511
000364	020027	000014		CMP	R0,#14	:	I,*		
000370	101766			BLOS	8\$				
000372	005000			CLR	R0	:			4413
000374	000207			RTS	PC	:			4355

: Routine Size: 127 words, Routine Base: \$CODE\$ + 5536
: Maximum stack depth per invocation: 7 words

: 4524 global routine copy (fft, ttt, length) = !copy a string
: 4525 begin
: 4526 incru i from 0 to .length by 2 do
: 4527 .ttt + .i = .(.fff + .i)
: 4528 end;

		.SBTTL COPY GLOBAL ROUTINE DECLARATIONS		
000000	004137	000000G	COPY:: JSR R1,\$SAVE2	4524
000004	005001		CLR R1	4525
000006	000411		BR 2\$	
000010	010102	000012	1\$: MOV R1,R2	4527
000012	066602		ADD 12(SP),R2	
000016	010100		MOV R1,R0	
000020	066600	000014	ADD 14(SP),R0	
000024	011012		MOV (R0),(R2)	
000026	062701	000002	ADD #2,R1	4525
000032	020166	000010	2\$: CMP R1,10(SP)	
000036	101764		BLOS 1\$	
000040	012700	177777	MOV #-1,R0	4524
000044	000207		RTS PC	

: Routine Size: 19 words, Routine Base: \$CODE\$ + 6134
 : Maximum stack depth per invocation: 4 words

: 4529
 : 4530 end
 : 4531
 : 4532 eludom

:
 : OTS external references
 : .GLOBL \$SAVE4, \$SAVE3, \$SAVE2, BLSMUL

: PSECT SUMMARY

Psect Name	Words	Attributes
\$CODE\$	1601	RO, I, LCL, REL, CON

: LIBRARY STATISTICS

File	----- Total	Symbols Loaded	----- Percent	Blocks Read
DISK\$USER:[PRUCHA.RELEASE]ZRQBA0.L16;2	356	313	87	65

ZRQB4 RDRX DISK FORMATTER
REV A PATCH 00 GLOBAL ROUTINE DECLARATIONS

L 14

28-Jun-1983 13:04:36
27-Jun-1983 18:06:48

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB4.B16;1 (24)

SEQ 180
Page 93

COMMAND QUALIFIERS

:
:
: Size: 1601 code + 0 data words
: Run Time: 01:09.3
: Elapsed Time: 02:57.0
: Memory Used: 253 pages
: Compilation Complete

ZRQB5

RDRX DISK FORMATTER

M 14

28-Jun-1983 13:07:41
27-Jun-1983 18:07:06

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB5.B16;1 (1)

SEQ 181
Page 1

```
:  
: 0001 MODULE ZRQB5 (%TITLE 'RDRX DISK FORMATTER'  
: 0002 IDENT = 'REV A PATCH 00'  
: 0003 ADDRESSING MODE (ABSOLUTÉ),  
: 0004 ENVIRONMENT (NOEIS)) =  
: 0005 BEGIN  
: 0006  
: 0007 LIBRARY 'ZRQBA0.L16'; !Define RDRX Formatter Library  
: 0008  
: 0009 REQUIRE 'BLSMAC.REQ'; !Define Bliss macro require file  
: 1500  
: 1501 %SBTTL 'LAST ADDRESS AND SETUP SECTION'  
: 1502  
: 1503  
:
```

```

1504
1505
1506 | +
1507 | The LASTAD macro must be the final statement (except .end) in a pro-
1508 | gram. The call generates an even address reflecting the first word of
1509 | memory unused by the program.
1510 | -
1511 LASTAD
1512
1513 | +
1514 | Hardcoded P-TABLES
1515
1516 | These optional hardware P-TABLES are located (when present) between
1517 | the "LASTAD" macro and the ".END" statement. These hardware P-TABLES
1518 | are above and beyond the default hardware P-TABLE located in the main
1519 | body of the program. These P-TABLES wind up appended to the BIN file
1520 | of the diagnostic, just as though the supervisor or the "SETUP" utility
1521 | had built them there. Thus the diagnostic can be "pre-parameterized"
1522 | by the programmer.
1523
1524 | If this hardcoded P TABLE section is not wanted then define "number" in
1525 | the BGNSETUP macro to zero and omitt BGNTAB and ENDTAB macros.
1526
1527 | Coding sample is as follows:
1528
1529 | LASTAD
1530
1531 | BGNSETUP (Number) !Number of P-TABLES
1532
1533 | BGNPTAB
1534 | (DATA)
1535 | (DATA)
1536 | (DATA)
1537 | ENDPTAB
1538
1539 | BGNPTAB
1540 | (DATA)
1541 | (DATA)
1542 | (DATA)
1543 | ENDPTAB
1544
1545 | ENDSETUP
1546 | .END
1547 | -
1548
1549 | BGNSETUP (1);
P 1550 | BGNPTAB
P 1551 | %0'172150', %0'154', 4, 0
1552 | ENDPTAB
1553 | ENDSETUP

```

.TITLE ZRQB5 RDRX DISK FORMATTER

ZRQB5 RDRX DISK FORMATTER
REV A PATCH 00 LAST ADDRESS AND SETUP SECTION

28-Jun-1983 13:07:41
27-Jun-1983 18:07:06

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB5.B16;1 (2)

.IDENT /REV A /
.ENABL AMA

000000
000000 000020'
000002 000000C
000004 000000
000006 000004
000010 172150
000012 000154
000014 000004
000016 000000
000020 000000

.PSECT \$XYZ\$, RO
BLSLAS:::WORD TSFREE
.WORD <<TSFREE-<BLSLAS+4>>/2>
P.AAA: .WORD 0
.WORD 4
P.AAB: .WORD -5630
.WORD 154
.WORD 4
.WORD 0
TSFREE:::WORD 0

: Plit count word

000004'
000001
000004'
000010'

L\$LAST== BLSLAS+4
T\$PTHV== 1
S\$LAS1= P.AAA
S\$REM2= P.AAB

000000 000207

.SBTTL SEND.LINK LAST ADDRESS AND SETUP SECTION
SEND.LINK:::
RTS PC ;

1499

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

: 1554 END
: 1555 ELUDOM

PSECT SUMMARY

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

LIBRARY STATISTICS

: File ----- Symbols ----- Blocks
: Total Loaded Percent Read
: DISK\$USER:[PRUCHA.RELEASE]ZRQBA0.L16;2
: 356 53 14 13

ZRQB5 RDRX DISK FORMATTER
REV A PATCH 00 LAST ADDRESS AND SETUP SECTION

28-Jun-1983 13:07:41
27-Jun-1983 18:07:06

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQB5.B16;1 (2)

: COMMAND QUALIFIERS

: Size: 1 code + 9 data words
: Run Time: 00:07.1
: Elapsed Time: 00:32.6
: Memory Used: 107 pages
: Compilation Complete

RDRX MACROS
ASSIGN BITS

MACRO M1200 28-JUN-83 13:08 PAGE 5

```

15
16
17
18
19
20
21
22
23
24
25
26
27 000000
28 000000
29 000000 000037 000034 000037
   000006 000036 000037 000036
   000014 000037 000037 000036
   000022 000037 000036 000037
30
31 000030
32
33
34
35
36 000030
   000030 010446
   000032 010346
   000034 010246
   000036 010146
37 000040 005000
38 000042 016601 000012
39 000046 005003
40
41 000050 112102
42 000052
   000052 004767 000162
   000056 103001
   000060 000207
43 000062
   000062 004767 000214
44 000066 112102
45 000070
   000070 004767 000144
   000074 103001
   000076 000207
46 000100 060203
47 000102 020327 000014
48 000106
   000106 000000
49
50 000110 005201
51 000112 005303
52 000114 010304
53 000116 005003
54 000120 112102
55 000122
   000122 004767 000112
    
```

.LIST MEB

```

.MACRO $DIGCHK
CALL .DIGCHK
BCC .+4
RETURN
.ENDM
    
```

```

.MACRO $MUL10,?L
CALL .MUL10
.ENDM
    
```

.PSECT \$DATES

```

DAT.TBL: .WORD 31.,28.,31.,30.,31.,30.,31.,31.,30.,31.,30.,31.
    
```

DATE.CHECK::

```

:STACK PICTURE: BUFFER ADDRESS
:                RETURN ADDRESS <-SP
:
:
    
```

```

:PRM R1,R2,R3,R4
:MOV R4,-(SP)
:MOV R3,-(SP)
:MOV R2,-(SP)
:MOV R1,-(SP)
:CLR R0
:MOV 12(SP),R1
:CLR R3
:GET AND CHECK MONTH, SAVE BINARY VALUE AS INDEX TO CHECK DAY
:MOV (R1)+,R2
:SDIGCHK
:JSR PC,.DIGCHK
:BCC .+4
:RTS PC
:SMUL10
:JSR PC,.MUL10
:MOV (R1)+,R2
:SDIGCHK
:JSR PC,.DIGCHK
:BCC .+4
:RTS PC
:ADD R2,R3
:CMP R3,#12.
:IF LE
:WORD 0
:GET AND CHECK DAY
:INC R1
:DEC R3
:MOV R3,R4
:CLR R3
:MOV (R1)+,R2
:SDIGCHK
:JSR PC,.DIGCHK
:ASSUME SUCCESS
:R1 CONTAINS BUFFER POINTER
:R3 USED AS ACCUMULATOR
:R2 CONTAINS INDIVIDUAL DIGITS
:CHECK DIGIT 1ST M DIGIT
: MULTIPLY IT BY 10
: 2ND M DIGIT
:MAKE ROOM FOR BRANCH INSTRUCTION
:SKIP '-' CHAR (OR WHATEVER DELIMITER OP CHOOSES)
: SAVE MONTH AS INDEX INTO DAT.TBL
: CLEAR R3 AS ACCUMULATOR
: 1ST D DIGIT
    
```

RDRX MACROS
ASSIGN BITS

MACRO M1200 28-JUN-83 13:08 PAGE 5-1

```

000126 103001          BCC    .+4
000130 000207          RTS    PC
56 000132          JSR    PC, .MUL10
000132 004767 000144          MOVW  (R1)+,R2          ; 2ND D DIGIT
57 000136 112102          JSR    PC, .DIGCHK
58 000140          JSR    PC, .DIGCHK
000140 004767 000074          BCC    .+4
000144 103001          RTS    PC
000146 000207          ADD    R2,R3
59 000150 060203          CMP    DAT.TBL(R4),R3
60 000152 026403 000000'    IF    LE
61 000156          .WORD  0          ;MAKE ROOM FOR BRANCH INSTRUCTION
000156 000000          ;GET AND CHECK YEAR
62          INC    R1          ;SKIP '-' CHAR (OR WHATEVER DELIMITER OP CHO
63 000160 005201          MOVW  (R1)+,R2          ; 1ST YEAR DIGIT
64 000162 112102          JSR    PC, .DIGCHK
65 000164 004767 000050          BCC    .+4
000164 004767 000050          RTS    PC
000170 103001          MOVW  (R1),R2          ; 2ND YEAR DIGIT
000172 000207          JSR    PC, .DIGCHK
66 000174 111102          BCC    .+4
000174 111102          RTS    PC
67 000176          MOVW  (R1),R2          ; 2ND YEAR DIGIT
000176 004767 000036          JSR    PC, .DIGCHK
000202 103001          BCC    .+4
000204 000207          RTS    PC
68 000206          POP    R1,R2,R3,R4          ; SUCCESS
000206 012601          MOV    (SP)+,R1
000210 012602          MOV    (SP)+,R2
000212 012603          MOV    (SP)+,R3
000214 012604          MOV    (SP)+,R4
69 000216 012616          MOV    (SP)+,(SP)          ; MOVE UP RETURN ADDRESS
70 000220          RETURN
000220 000207          RTS    PC
71 000222          ENDF
000156 003021          .WORD  $$$I2+$
72 000222          ENDF
000106 003045          .WORD  $$$I1+$
73          ; FAILURE
74 000222          DATBAD:
75 000222          POP    R1,R2,R3,R4
000222 012601          MOV    (SP)+,R1
000224 012602          MOV    (SP)+,R2
000226 012603          MOV    (SP)+,R3
000230 012604          MOV    (SP)+,R4
76 000232 012616          MOV    (SP)+,(SP)          ; MOVE UP RETURN ADDRESS
77 000234 005200          INC    R0
78 000236          RETURN
000236 000207          RTS    PC
79
80 000240          .DIGCHK:
81          :
82          : CHECKS ASCII DIGIT IN R2 (MAKES SURE IT IS A DIGIT),
83          : AND STRIPS OFF ASCII MASK RETURNING BINARY VALUE IN R2
84          :
85 000240 042702 177600          BIC    #177600,R2          ;STRIP UPPER BYTE
86 000244 120227 000060          CMPB  R2,#'0
87 000250          IF    GE

```

RDRX MACROS
ASSIGN BITS

MACRO M1200 28-JUN-83 13:08 PAGE 5-2

```

      000250 000000          .WORD 0          ;MAKE ROOM FOR BRANCH INSTRUCTION
88 000252 120227 000071    CMPB R2,#'9
89 000256          .WORD 0          IF LE
90 000260 042702 000060    BIC #60,R2 ;MAKE ROOM FOR BRANCH INSTRUCTION
91 000264          .WORD 0          RETURN
92 000266 000207    RTS PC
      000264 000207    RTS PC
      000266 003003    .WORD $$$I2+$
93 000266          .WORD $$$I1+$
      000250 002406    .WORD $$$I1+$
94 000266 011626    MOV (SP),(SP)+
95 000270 012766 000222' 000002  MOV #DATBAD,2(SP)
96 000276 000261    SEC
97 000300          RETURN
      000300 000207    RTS PC
98
99 000302          .MUL10:
100          .:
101          .: MULTIPLIES BINARY NUMBER IN R2 BY 10, LEAVING RESULT IN R3
102          .: CORRUPTS R2
103          .:
104 000302 006302    ASL R2
105 000304 010203    MOV R2,R3
106 000306 006303    ASL R3
107 000310 006303    ASL R3
108 000312 060203    ADD R2,R3
109 000314          RETURN
      000314 000207    RTS PC
110
111          000001    .END

```

RDRX MACROS
SYMBOL TABLE

MACRO M1200 28-JUN-83 13:08 PAGE 5-3

BIT0 = 000001 G	BIT1 = 000002 G	BIT6 = 000100 G	\$\$\$B2 = 000256R	002 \$\$\$LE = 003000
BIT00 = 000001 G	BIT10 = 002000 G	BIT7 = 000200 G	\$\$\$CC = 103400	\$\$\$LO = 103400
BIT01 = 000002 G	BIT11 = 004000 G	BIT8 = 000400 G	\$\$\$CS = 103000	\$\$\$LOS = 101000
BIT02 = 000004 G	BIT12 = 010000 G	BIT9 = 001000 G	\$\$\$EQ = 001000	\$\$\$LT = 002000
BIT03 = 000010 G	BIT13 = 020000 G	DATBAD 000222R	002 \$\$\$GE = 002400	\$\$\$MI = 100000
BIT04 = 000020 G	BIT14 = 040000 G	DATE.C 000030RG	002 \$\$\$GT = 003400	\$\$\$NE = 001400
BIT05 = 000040 G	BIT15 = 100000 G	DAT.TB 000000R	002 \$\$\$HI = 101400	\$\$\$PL = 100400
BIT06 = 000100 G	BIT2 = 000004 G	\$ = 000006	\$\$\$HIS = 103000	\$\$\$VC = 102400
BIT07 = 000200 G	BIT3 = 000010 G	\$\$ = 000266R	002 \$\$\$IF = 000000	\$\$\$VS = 102000
BIT08 = 000400 G	BIT4 = 000020 G	\$\$\$BR = 000400	\$\$\$I1 = 002400	.DIGCH 000240R 002
BIT09 = 001000 G	BIT5 = 000040 G	\$\$\$B1 = 000250R	002 \$\$\$I2 = 003000	.MUL10 000302R 002

. ABS. 000000 000
 000000 001
 \$DATES 000316 002
 ERRORS DETECTED: 0

VIRTUAL MEMORY USED: 11096 WORDS (44 PAGES)
 DYNAMIC MEMORY: 20060 WORDS (77 PAGES)
 ELAPSED TIME: 00:00:17
 DATCHK.OBJ,DATCHK.LIS/-SP=ZRQBMC.MAC,DATCHK.MAC

```

0001 %TITLE 'RDRX FORMATTER LIBRARY MODULE'
0002
0003 %sbttl 'MACRO DEFINITIONS'
0004
0005 macro
M 0006   WRT_RDRX (O, FIELDNAM, IMAGE) =
M 0007   begin
M 0008     local
M 0009     RCSM_REG;
M 0010     RCSM_REG = .RDRX_ADDR [O, RC ALL];
M 0011     RCSM_REG <%fieldexpand (FIELDNAM)> = IMAGE;
M 0012     (.RDRX_ADDR + %upval*O) = .RCSM_REG;
M 0013     end%;
0014
0015   :
0016   Dup Protocol Macros Declarations
0017   :
M 0018   REC_BASE =
0019     HDR_SIZ, 0, 0, 16, 0%;
M 0020   SND_BASE =
0021     HDR_SIZ + REC_ALLOCATE , 0, 0, 16, 0%;
0022   :
0023   Macro to clear out the DUP send data text buffer
0024   before requesting input form operator. This by
0025   default puts a null byte at the end of the ascii
0026   input.
0027   :
M 0028   CLR_SBUF =
M 0029     incru i from 0 to SNDB_SIZE - 1 do
0030       SND_BUF [.i] = ZEROS;%;
0031   :
0032   General purpose word reference field select
0033   :
M 0034   WORD_REF =
0035     0, 16, 0%;
0036

```

0037 %sbttl 'SUPERVISOR DEFINED LITERALS'

0038

0039 literal

0040

0041 !+ BIT DIFINITIONS

0042 !-

0043 BIT15 = %o'100000',
 0044 BIT14 = %o'40000',
 0045 BIT13 = %o'20000',
 0046 BIT12 = %o'10000',
 0047 BIT11 = %o'4000',
 0048 BIT10 = %o'2000',
 0049 BIT09 = %o'1000',
 0050 BIT08 = %o'400',
 0051 BIT07 = %o'200',
 0052 BIT06 = %o'100',
 0053 BIT05 = %o'40',
 0054 BIT04 = %o'20',
 0055 BIT03 = %o'10',
 0056 BIT02 = %o'4',
 0057 BIT01 = %o'2',
 0058 BIT00 = %o'1',

0059

0060 BIT9 = BIT09,
 0061 BIT8 = BIT08,
 0062 BIT7 = BIT07,
 0063 BIT6 = BIT06,
 0064 BIT5 = BIT05,
 0065 BIT4 = BIT04,
 0066 BIT3 = BIT03,
 0067 BIT2 = BIT02,
 0068 BIT1 = BIT01,
 0069 BIT0 = BIT00,

0070 !+

0071 !+ EVENT FLAG DEFINITIONS

0072 !- EF32:EF17 RESERVED FOR SUPERVISOR TO PROGRAM COMMUNICATION

0073 !-

0074 EF_START = 32,	! START COMMAND WAS ISSUED
0075 EF_RESTART = 31,	! RESTART COMMAND WAS ISSUED
0076 EF_CONTINUE = 30,	! CONTINUE COMMAND WAS ISSUED
0077 EF_NEW = 29,	! A NEW PASS HAS BEEN STARTED
0078 EF_PWR = 28,	! A POWER-FAIL/POWER-UP OCCURRED

0079 !+

0080 !+ PRIORITY LEVEL DEFINITIONS

0081 !-

0082 PRI07 = %o'340',
 0083 PRI06 = %o'300',
 0084 PRI05 = %o'240',
 0085 PRI04 = %o'200',
 0086 PRI03 = %o'140',
 0087 PRI02 = %o'100',
 0088 PRI01 = %o'40',
 0089 PRI00 = %o'0',

RDRX FORMATTER LIBRARY MODULE
SUPERVISOR DEFINED LITERALS

28-Jun-1983 13:01:22
28-Jun-1983 12:55:24

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQBA0.R16;2 (2)

0090 !+
0091 !: OPERATOR FLAG BITS
0092 !-
0093 EVL = %o'4'
0094 LOT = %o'10'
0095 ADR = %o'20'
0096 IDU = %o'40'
0097 ISR = %o'100'
0098 UAM = %o'200'
0099 BOE = %o'400'
0100 PNT = %o'1000'
0101 PRI = %o'2000'
0102 IXE = %o'4000'
0103 IBE = %o'10000'
0104 IER = %o'20000'
0105 LOE = %o'40000'
0106 HOE = %o'100000'

0107 %sbttl 'FORMATTER DEFINED LITERALS'

0108 :
0109 : Message selection literals
0110 :

0111 MSG0 = 0.
0112 MSG1 = 1.
0113 MSG2 = 2.
0114 MSG3 = 3.
0115 MSG4 = 4.
0116 MSG5 = 5.
0117 MSG6 = 6.
0118 MSG7 = 7.
0119 MSG8 = 8.
0120 MSG9 = 9.
0121 MSG10 = 10.
0122 MSG11 = 11.
0123 MSG12 = 12.
0124 MSG13 = 13.
0125 MSG14 = 14.
0126 MSG15 = 15.
0127 MSG16 = 16.
0128 MSG17 = 17.
0129 MSG18 = 18.
0130 MSG19 = 19.
0131 MSG20 = 20.
0132 MSG21 = 21.
0133 MSG22 = 22.
0134 MSG23 = 23.
0135 MSG24 = 24.
0136 MSG25 = 25.
0137 MSG26 = 26.
0138 MSG27 = 27.
0139 MSG28 = 28.
0140 MSG29 = 29.
0141 MSG30 = 30.

!Select message 0
!Select message 1
!Select message 2
!Select message 3
!Select message 4
!Select message 5
!Select message 6
!Select message 7
!Select message 8
!Select message 9
!Select message 10
!Select message 11
!Select message 12
!Select message 13
!Select message 14
!Select message 15
!Select message 16
!Select message 17
!Select message 18
!Select message 19
!Select message 20
!Select message 21
!Select message 22
!Select message 23
!Select message 24
!Select message 25
!Select message 26
!Select message 27
!Select message 28
!Select message 29
!Select message 30

0142 :
0143 : Miscellaneous literals
0144 :

0145 FAILURE = 0.
0146 SUCCESS = 1.
0147 BLK0 = 0.
0148 BLK1 = 1.
0149 BLK2 = 2.
0150 BLK3 = 3.
0151 WRD0 = 0.
0152 WRD1 = 1.
0153 WRD2 = 2.
0154 WRD3 = 3.
0155 WRD4 = 4.
0156 WRD5 = 5.
0157 WRD6 = 6.
0158 ISRD = 0.
0159 ISWRT = 1.

!Failure return code
!Success return code
!Selects block 0 of struct
!Selects block 1 of struct
!Selects block 2 of struct
!Selects block 3 of struct
!Selects word 0 of block
!Selects word 1 of block
!Selects word 2 of block
!Selects word 3 of block
!Selects word 4 of block
!Selects word 5 of block
!Selects word 6 of block
!Select the read word (1st) in ISD_STRUCTURE
!Select the write word (2nd) in ISD_STRUCTURE

RDRX FORMATTER LIBRARY MODULE
FORMATTER DEFINED LITERALS28-Jun-1983 13:01:22
28-Jun-1983 12:55:24VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQBA0.R16;2 (3)

```

0160 STEP1 = 0,           !Initialization sequence step one
0161 STEP4 = 3,         !Initialization sequence step two
0162 TRUE = 1,          !True indicator
0163 FALSE = 0,         !False indicator
0164 SET_FLG = 1,       !Set the indicated flag to one
0165 CLR_FLG = 0,       !Clear the indicated flag to zero
0166 ONE = 1,           !Ones data type
0167 ZERO = 0,          !Zero data type
0168 ONES = %o'177777', !All ones data type
0169 ZEROS = 0,         !All zeros data type
0170 LSLIMIT = 16,     !Allowable number of units to format
0171 DM_SIZE = 8192,   !Size of DM buffer size
0172 FCT_SIZE = 4096,  !Size of fct buffer size
0173 !
0174 ! Command opcode bits 6 and 7 indicate the type of message
0175 ! (command, end or attention message). The following literals
0176 ! define these field values.
0177 !
0178 CMD$MSG = %b'00',   !Command opcode message type
0179 ATT$MSG = %b'01',   !Attention opcode msg type
0180 END$MSG = %b'10',  !End opcode message type
0181 !
0182 ! Error code literals
0183 !
0184 PAS_CODE = %o'00',  !Pass code
0185 CIE_CODE = %o'01',  !Communication area init error
0186 CTO_CODE = %o'11',  !Controller time out error
0187 PFE_CODE = %o'21',  !Port fatal error
0188 RSE_CODE = %o'31',  !response status error
0189 PVE_CODE = %o'41',  !Host/Controller out of sequence
0190 RPD_CODE = %o'51',  !Remote program died error code
0191 PSE_CODE = %o'61',  !Port/host synchronous error
0192 MLE_CODE = %o'71',  !Message length error code
0193 UEC_CODE = %o'101', !Unknown end code error
0194 APR_CODE = %o'201', !Adaptor purge request error
0195 UIN_CODE = %o'301', !Unknown interrupt error code
0196 ATN_CODE = %o'401', !Attention msg received
0197 CMD_CODE = %o'501', !Command msg received
0198 SEX_CODE = %o'601', !Serious exception error received
0199 IVC_CODE = %o'701', !Invalid command error received
0200 UMT_CODE = %o'1001', !Unknown message type
0201 OBF_CODE = %o'2001', !Out standing buffer full error
0202 OSE_CODE = %o'3001', !Out$std_buffer out of sync error
0203 UMN_CODE = %o'4001', !Unknown message number
0204 FRE_CODE = %o'5001', !File read error code from load media
0205 !
0206 ! Dup Protocol literals
0207 !
0208 ! Note:
0209 ! The values assigned to the literals
0210 ! REC_SIZ, SND_SIZ are represented in
0211 ! powers of 2 notation per DUP spec.
0212 !

```

```

0213 | By redefining these two literals the
0214 | communications area allocation and init
0215 | sequence is automatically handled and
0216 | no other parameter modification is needed.
0217 |
0218 | Further more the send and receive rings
0219 | can be of different lengths. However the
0220 | maximum value allowed is 7 (2*7 = 128 slots)
0221 | per DUP spec.
0222 |
0223 | REC_SIZ = 2,                !Define number of receive slots
0224 | SND_SIZ = 2,                !Define number of send slots
0225 |
0226 | This one is tricky: (Hdr_siz)
0227 | The communication area is defined to be a
0228 | contiguous Blockvector (two words per block)
0229 | data segment of size, Rec_allocate + Snd_allocate +
0230 | Hdr_siz. The two words per block coming from the
0231 | two words needed to represent the ring descriptors.
0232 |
0233 | Hdr_siz is then really * 2 or 4 words of storage
0234 | to represent the interrupt and purge indicators.
0235 |
0236 | HDR_SIZ = 2,                !Define com area header size 4 words
0237 |
0238 |
0239 | REC_ALLOCATE = 1*REC_SIZ,   !Define receive ring allocation
0240 | SND_ALLOCATE = 1*SND_SIZ,   !Define send ring allocation
0241 |
0242 | Ring_size equals the total number of ring descriptors
0243 | (number of 2 word blocks) allocated within the
0244 | communications area.
0245 |
0246 | RING_SIZE = REC_ALLOCATE + SND_ALLOCATE + HDR_SIZ,
0247 |
0248 | The RB_SIZ by definition, DUP spec, must be
0249 | a minimum of 60 bytes long (30 words) 64 bytes
0250 | overall. The additional 2 words for the UQ
0251 | port information is accounted for in azkel2
0252 | when the storage is allocated.
0253 |
0254 | RB_SIZE = 30,               !Number of "words" in response buffer
0255 |
0256 | The biggest command I'll ever send will
0257 | be (I hope) will be 40 bytes, 42 overall,
0258 | and again the UQ port 2 words is allowed
0259 | for in azkel2.
0260 |
0261 | SB_SIZE = 20,               !Number of "words" in send buffer
0262 |
0263 | SNDB_SIZE = 37,             !Send DUP cmd text buffer size
0264 | SNDB_COUNT = 16,           !Byte count to send to RDRX Formatter
0265 | RECB_SIZE = 120,           !Receive DUP cmd text buffer size

```

RDRX FORMATTER LIBRARY MODULE
FORMATTER DEFINED LITERALS28-Jun-1983 13:01:22
28-Jun-1983 12:55:24VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQBA0.R16;2 (3)

```

0266      PORT_OWNED = 1,
0267      HOST_OWNED = 0,
0268
0269      Delay literal values
0270
0271      An argument of one results in a 100us
0272      delay.
0273
0274      A roman numeral notation is used to
0275      denote values of delay arguments.
0276      The notation is as follows:
0277
0278      I (1), V (5), X (10), L (50), C (100),
0279      D (500), M (1000)
0280
0281      Any symbol following another of equal or greater
0282      value adds to its value, as II = 2, XI = 11.
0283
0284      Any symbol proceeding one of greater value subtracts
0285      from the second and the remainder added to the first
0286      as XIV = 14, LIX = 59.
0287
0288      ONE_SEC = 10000,
0289      C_US = 1,
0290      CC_US = 2,
0291      CCC_US = 3,
0292      XC_US = 4,
0293      D_US = 5,
0294
0295      RDRX register offsets
0296
0297      RCIP = 0,
0298      RCSA = 1,
0299
0300      Command packet opcodes
0301      (See note following)
0302
0303      OP_ABO = %o'1',
0304      OP_ACC = %o'20',
0305      OP_AVL = %o'10',
0306      OP_CCD = %o'21',
0307      OP_CMP = %o'40',
0308      OP_DAP = %o'13',
0309      OP_ERS = %o'22',
0310      OP_FLU = %o'23',
0311      OP_GCS = %o'02',
0312      OP_GUS = %o'03',
0313      OP_ONL = %o'11',
0314      OP_RD = %o'41',
0315      OP_RPL = %o'24',
0316      OP_SCC = %o'04',
0317      OP_SUC = %o'12',
0318      OP_WR = %o'42',

```

!Descriptor owned by port
!Descriptor owned by host!One second delay argument
!100 micro sec delay argument
!200 micro sec delay argument
!300 micro sec delay argument
!400 micro sec delay argument
!500 micro sec delay argument!Abort command
!Access command
!Available command
!Compare controller data command
!Compare host data command
!Determine access path
!Erase command
!Flush command
!Get command status command
!Get unit status command
!Online command
!Read command
!Replace command
!Set controller characteristics command
!Set unit characteristics command
!Write command

```

0319 | OP_MRD = %o'30',           !Maintenance read command
0320 | OP_MWR = %o'31',           !Maintenance write command
0321 |
0322 | End message and serious exception encodes
0323 | (see note following)
0324 |
0325 | OP_END = %o'200',         !End packet flag
0326 | OP_SEX = %o'7',          !Serious exception end packet
0327 |
0328 | MSCP Attention message encodes
0329 |
0330 | OP_AVA = %o'100',        !Available attention message
0331 | OP_DUP = %o'101',        !Duplicate unit number attention message
0332 | OP_ACP = %o'102',        !Access path attention message
0333 | OP_RLC = %o'103',        !Reset command limit attention message
0334 |
0335 | The following are the dup op-code commands
0336 |
0337 | OP_GDS = %o'1',          !Get dust status
0338 | OP_ESP = %o'2',          !Execute supplied program
0339 | OP_ELP = %o'3',          !Execute local program
0340 | OP_SED = %o'4',          !Send data
0341 | OP_RED = %o'5',          !Receive data
0342 | OP_ABT = %o'6',          !Abort program
0343 |
0344 | NOTE:
0345 | -----
0346 |
0347 | End message opcodes (also called encodes) are formed by adding the end
0348 | message flag to the command opcode. For example, a READ commands end
0349 | message contains the value OP.RED + OP.END in its opcode field. The Invalid
0350 | command end message contains just the end message flag (i.e., OP.END) in
0351 | its opcode field. The serious exception opcode shown above (i.e. OP.SEX +
0352 | OP.END) in its opcode field.
0353 |
0354 | Commands opcode bits 6 and 7 indicate the type of message (command, end or
0355 | attention message. Command opcodes bits 3 through 5 indicate the command
0356 | category (immediate, sequential or no-sequential) and whether or not the
0357 | command includes a buffer descriptor.
0358 |
0359 | See MSCP document appendix 'A-1 NOTE:Ⓜ for more information on this topic.
0360 |
0361 | DUP endcode message types
0362 |
0363 |
0364 | EOP_GDS = OP_GDS + OP_END, !Get dust status
0365 | EOP_ESP = OP_ESP + OP_END, !Execute supplied program
0366 | EOP_ELP = OP_ELP + OP_END, !Execute local program
0367 | EOP_RED = OP_RED + OP_END, !Receive data
0368 | EOP_SED = OP_SED + OP_END, !Send data
0369 | EOP_ABT = OP_ABT + OP_END, !Abort program
0370 |
0371 | MSCP endcode messag types

```

RDRX FORMATTER LIBRARY MODULE
FORMATTER DEFINED LITERALS28-Jun-1983 13:01:22
28-Jun-1983 12:55:24VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQBA0.R16;2 (3)SEQ 197
Page 9

```

0372      !
0373      ! EOP_ABO = OP_ABO + OP_END,      !Abort command
0374      ! EOP_ACC = OP_ACC + OP_END,      !Access command
0375      ! EOP_AVL = OP_AVL + OP_END,      !Available command
0376      ! EOP_CCD = OP_CCD + OP_END,      !Compare controller data command
0377      ! EOP_CMP = OP_CMP + OP_END,      !Compare host data command
0378      ! EOP_ERS = OP_ERS + OP_END,      !Erase command
0379      ! EOP_FLU = OP_FLU + OP_END,      !Flush command
0380      ! EOP_GCS = OP_GCS + OP_END,      !Get command status command
0381      ! EOP_GUS = OP_GUS + OP_END,      !Get unit status command
0382      ! EOP_ONL = OP_ONL + OP_END,      !Online command
0383      ! EOP_RD = OP_RD + OP_END,        !Read command
0384      ! EOP_RPL = OP_RPL + OP_END,      !Replace command
0385      ! EOP_SCC = OP_SCC + OP_END,      !Set controller characteristics command
0386      ! EOP_SUC = OP_SUC + OP_END,      !Set unit characteristics command
0387      ! EOP_WR = OP_WR + OP_END,        !Write command
0388      ! EOP_MRD = OP_MRD + OP_END,      !Maintenance read command
0389      ! EOP_MWR = OP_MWR + OP_END,      !Maintenance write command
0390      ! EOP_SEX = OP_SEX + OP_END,      !Serious exception
0391
0392      !+
0393      ! Dup Command message envelope
0394      ! byte sizes beginning at text+0
0395      ! of the command envelope.
0396      !-
0397      ! SZ_GDS = 12,                      !Get dust status size
0398      ! SZ_ESP = 40,                      !Execute supplied program size
0399      ! SZ_ELP = 48,                      !Execute local program size
0400      ! SZ_RED = 28,                      !Receive data size
0401      ! SZ_SED = 28,                      !Send data size
0402      ! SZ_ABT = 12,                      !Abort program size
0403      !+
0404      ! MSCP Command message envelope
0405      ! byte sizes beginning at text+0
0406      ! of the command envelope.
0407      !-
0408      ! SZ_SCC = 32,                      !Set Controller characteristics
0409      ! SZ_ONL = 36,                      !On-line command
0410
0411      !+
0412      ! The following are the expected number of bytes
0413      ! in a commands end packet transmitted by the
0414      ! communications mechanism to the host.
0415      !-
0416      !
0417      ! DUP command end message sizes
0418      !
0419      ! ESZ_GDS = %DECIMAL '22',          !Get dust status end packet size
0420      ! ESZ_ESP = %DECIMAL '12',          !Execute supplied prog end packet size
0421      ! ESZ_ELP = %DECIMAL '16',          !Execute local prog end packet size
0422      ! ESZ_RED = %DECIMAL '16',          !Receive data end packet size
0423      ! ESZ_SED = %DECIMAL '16',          !Send data end packet size
0424      ! ESZ_ABT = %DECIMAL '12',          !Abort program end packet size

```

RDRX FORMATTER LIBRARY MODULE
FORMATTER DEFINED LITERALS

```

0425 |
0426 | MSCP command end message sizes
0427 |
0428 | ESZ_SCC = %DECIMAL '28',
0429 | ESZ_ONL = %DECIMAL '44',
0430 |
0431 | DUP STANDALONE FLAG MODIFIER
0432 |
0433 | DUP_STND = 1,
0434 |
0435 | JDT Trap Vector
0436 |
0437 | O_TVEC = %0'14';

```

```

!Set controller characteristics
!On line command

```


0438 %sbttl 'FIELD DECLARATIONS'

0439

0440 field

0441

0442

0443

0444

0445

0446

0447

0448

0449

0450

0451

0452

0453

0454

0455

0456

0457

0458

0459

0460

0461

0462

0463

0464

0465

0466

0467

0468

0469

0470

0471

0472

0473

0474

0475

0476

0477

0478

0479

0480

0481

0482

0483

0484

0485

0486

0487

0488

0489

0490

+
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

: Miscellaneous status register field
: reference 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 'pwr of 2'

S1W_RRING = [8, 3, 0].

!Number of R-ring slots 'pwr of 2'

```

0491      S1W_IE = [7, 1, 0],           !Init Sequence interrupt request
0492      S1W_VADR = [0, 7, 0],        !Interrupt vector address
0493      |
0494      | Step two read SA register field reference
0495      |
0496      S2R_STEP = [12, 1, 0],        !Step two step bit
0497      S2R_PTyp = [8, 3, 0],        !Port type number
0498      S2R_BIT7 = [7, 1, 0],        !Echoed IE bit from step one write
0499      S2R_WR = [6, 1, 0],          !Echoed bit 14 from step one write
0500      S2R_CRING = [3, 3, 0],       !Echoed bits 3-5 from step one write
0501      S2R_RRING = [0, 3, 0],       !Echoed bits 0-2 from step one write
0502      |
0503      | Step two write SA register field reference
0504      |
0505      S2W_LRBASE = [0, 16, 0],      !Ring base lower address
0506      S2W_PI = [0, 1, 0],          !Adapter purge interrupt request
0507      |
0508      | Step three read SA register field reference
0509      |
0510      S3R_STEP = [13, 1, 0],        !Step three step bit
0511      S3R_RSVD = [8, 3, 0],        !Reserved
0512      S3R_IE = [7, 1, 0],          !Echoed IE bit from step one write
0513      S3R_VADR = [0, 7, 0],        !Echoed VADR from step one write
0514      |
0515      | Step three write SA register field reference
0516      |
0517      S3W_PP = [15, 1, 0],          !Purge & Poll test request
0518      S3W_HRBASE = [0, 15, 0],     !Ring base high address
0519      |
0520      | Step four read SA register field reference
0521      |
0522      S4R_STEP = [14, 1, 0],        !Step four step bit
0523      S4R_RSVD = [8, 3, 0],        !Reserved
0524      S4R_UVER = [0, 8, 0],        !Controller u-CODE version
0525      |
0526      | Step four write SA register field reference
0527      |
0528      S4W_RSVD = [8, 8, 0],         !Reserved
0529      S4W_BURST = [2, 6, 0],       !Max number longwords per NPR xfer
0530      S4W_LF = [1, 1, 0],          !Last fail request
0531      S4W_GO = [0, 1, 0],          !Go bit
0532      tes,

```

RDRX FORMATTER LIBRARY MODULE
FIELD DECLARATIONS

28-Jun-1983 13:01:22
28-Jun-1983 12:55:24

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQBA0.R16;2 (5)

```

0533      |
0534      | +
0535      | Field declaration to define the interrupt
0536      | indicator and purge words of the communications
0537      | area.
0538      | -
0539      |
0540      | HDR_FIELD =
0541      | set
0542      | Header Word Ringbase -4
0543      |
0544      | ! RESERVED = [0, 0, 16, 0],
0545      |
0546      | Header Word Ringbase -3
0547      |
0548      | ! RSVD = [1, 0, 8, 0],
0549      | ADP_CH = [1, 8, 8, 0],
0550      |
0551      | Header Word Ringbase -2
0552      |
0553      | CMD_INT = [2, 0, 16, 0],
0554      |
0555      | Header Word Ringbase -1
0556      |
0557      | RSP_INT = [3, 0, 16, 0]
0558      | tes,
0559      |

```

```
0560      |  
0561      | + Field declaration to define the fields within  
0562      | | the send and receive ring descriptors.  
0563      | -  
0564      |  
0565      | DSC_FIELD =  
0566      |   set  
0567      |   |  
0568      |   | Low order envelope address  
0569      |   |  
0570      |   | LO_EN$AD = [0, 0, 16, 0],  
0571      |   |  
0572      |   | High order 18 bit unibus or Qbus address  
0573      |   |  
0574      |   | HI_EN$AD = [1, 0, 2, 0],  
0575      |   |  
0576      |   | Q_Bus extention address  
0577      |   |  
0578      |   | QB_EXT = [1, 2, 4, 0],  
0579      |   |  
0580      |   | Reserver field  
0581      |   |  
0582      |   | D$RSVD = [1, 6, 8, 0],  
0583      |   |  
0584      |   | Flag bit  
0585      |   |  
0586      |   | FLAG_BIT = [1, 14, 1, 0],  
0587      |   |  
0588      |   | Ownership bit  
0589      |   |  
0590      |   | OWN_BIT = [1, 15, 1, 0]  
0591      | tes,
```

```

0592      |++
0593      | Field declaration to define the fields within
0594      | message envelope buffers.
0595      |--
0596
0597      ENV_FIELD =
0598      set
0599
0600      | UQ Port envelope header field declaration
0601
0602      MSG_LENGTH = [0, 0, 16, 0],      !Message length
0603      CREDITS = [1, 0, 4, 0],          !Credits
0604      MSG_TYPE = [1, 4, 4, 0],        !Message type
0605      CONN_ID = [1, 8, 8, 0],         !Connection ID
0606
0607      | DUP/MSCP command and response envelope header
0608      | field declaration
0609
0610      CMD_LREF = [2, 0, 16, 0],        !Command Ref number low word
0611      CMD_HREF = [3, 0, 16, 0],        !Command Ref number high word
0612      UNIT_NUM = [4, 0, 16, 0],        !Unit selection field
0613      UN_LOSED = [4, 0, 16, 0],        !Unused low word
0614      UN_HUSED = [5, 0, 16, 0],        !Unused high word
0615      TYP$MSG = [6, 6, 2, 0],          !End code message type
0616      OPCODE = [6, 0, 8, 0],           !Opcode
0617      ENDCODE = [6, 0, 8, 0],          !Opcode
0618      ESFLAG = [6, 8, 8, 0],          !End message flag field
0619      RSVD = [6, 8, 8, 0],             !Reserved
0620      STATUS = [7, 0, 16, 0],          !Status
0621      MODIFIER = [7, 0, 16, 0],        !Modifier
0622
0623      |++
0624      | DUP command and response envelope parameter
0625      | field declarations
0626      |--
0627
0628      |++
0629      | ABORT command and response envelope parameter
0630      | field declaration
0631      |--
0632
0633      | No parameters declared
0634
0635      | response FIELD
0636      | No parameters declared
0637
0638      |++
0639      | GET DUST STATUS command and response envelope
0640      | parameter field declaration
0641      |--
0642
0643      | COMMAND FIELD
0644      | No parameters declared

```

RDRX FORMATTER LIBRARY MODULE
FIELD DECLARATIONS28-Jun-1983 13:01:22
28-Jun-1983 12:55:24VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZROBA0.R16;2 (7)

```

0645      |      response FIELD
0646      |
0647      | PLO_EXT = [8, 0, 16, 0],      |Program Extension low word
0648      | PHI_EXT = [9, 0, 8, 0],      |Program Extension high word
0649      | F$LAGS = [9, 8, 8, 0],      |Flags
0650      | FLG_B0 = [9, 8, 1, 0],      |Flag field bit 0
0651      | FLG_B1 = [9, 9, 1, 0],      |Flag field bit 1
0652      | FLG_B2 = [9, 10, 1, 0],     |Flag field bit 2
0653      | FLG_B3 = [9, 11, 1, 0],    |Flag field bit 3
0654      | PLO_IND = [10, 0, 16, 0],  |Progress indicator low word
0655      | PHI_IND = [11, 0, 16, 0],  |Progress indicator high word
0656      | TIM_OUT = [12, 0, 16, 0],  |Time out
0657      |
0658      |
0659      | + EXECUTE SUPPLIED PROGRAM command and response
0660      | envelope parameter field declartion
0661      | -
0662      |
0663      |      COMMAND FIELD
0664      |
0665      | BLO_CNT = [8, 0, 16, 0],      |Byte count low word
0666      | BHI_CNT = [9, 0, 16, 0],      |Byte count high word
0667      | BPA_LO = [10, 0, 16, 0],     |Buffer physical adrs bits <0-15>
0668      | BPA_HI = [11, 0, 2, 0],     |Buffer physical adrs bits <16-17>
0669      | QBUS_EXT = [11, 2, 4, 0],   |Q bus extention
0670      | RSV = [11, 6, 2, 0],        |Reserved field
0671      | UBA_CHAN = [11, 8, 8, 0],   |Unibus adaptor channel number
0672      | RSV0 = [ 12, 0, 16, 0],    |These next four words are not
0673      | RSV1 = [ 13, 0, 16, 0],    |in the UQ port implementation.
0674      | RSV2 = [ 14, 0, 16, 0],
0675      | RSV3 = [ 15, 0, 16, 0],
0676      |
0677      | These next field definitions are the same
0678      | as above except they are for the overlay
0679      | buffer descriptors. To make life easy for
0680      | me I'll use the same names and just prefix
0681      | the names with a $ for uniqueness.
0682      |
0683      | $BPA_LO = [16, 0, 16, 0],     |Buffer physical adrs bits <0-15>
0684      | $BPA_HI = [17, 0, 2, 0],     |Buffer physical adrs bits <16-17>
0685      | $QBUS_EXT = [17, 2, 4, 0],   |Q bus extention
0686      | $RSV = [17, 6, 2, 0],         |Reserved field
0687      | $UBA_CHAN = [17, 8, 8, 0],   |Unibus adaptor channel number
0688      | $RSV0 = [ 18, 0, 16, 0],    |These next four words are not
0689      | $RSV1 = [ 19, 0, 16, 0],    |in the UQ port implementation.
0690      | $RSV2 = [ 20, 0, 16, 0],
0691      | $RSV3 = [ 21, 0, 16, 0],
0692      |
0693      |      response FIELD
0694      | No parameters declared
0695      |
0696      | + EXECUTE LOCAL PROGRAM command and response
0697      |

```

RDRX FORMATTER LIBRARY MODULE
FIELD DECLARATIONS

28-Jun-1983 13:01:22
28-Jun-1983 12:55:24

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQBA0.R16:2 (7)

0698
0699
0700
0701
0702
0703
0704
0705
0706
0707
0708
0709
0710
0711
0712
0713
0714
0715
0716
0717
0718
0719
0720
0721
0722
0723
0724
0725
0726
0727
0728
0729
0730
0731
0732
0733
0734
0735
0736
0737
0738
0739
0740
0741
0742
0743
0744
0745
0746
0747
0748
0749
0750

```

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
RSSVD = [ 11, 0, 16, 0],    !Reserved
TSD_0 = [ 12, 0, 16, 0],    !Time and Date word 0
TSD_1 = [ 13, 0, 16, 0],    !Time and Date word 1
TSD_2 = [ 14, 0, 16, 0],    !Time and Date word 2
TSD_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

```

RDRX FORMATTER LIBRARY MODULE
FIELD DECLARATIONS

```

0751 !MSCP VER = [ 8, 0, 16, 0], !Same as cmd field
0752 !CTL_FLAGS = [ 9, 0, 16, 0], !Same as cmd field
0753 CTL_TOV = [ 10, 0, 16, 0], !Cntlr time out value
0754 CSVRSN = [ 11, 0, 8, 0], !Cntlr S/W, F/W, u-code rev num
0755 CHVRSN = [ 11, 8, 8, 0], !Cntlr H/W rev num
0756 CID_0 = [ 12, 0, 16, 0], !Cntlr identifier wrd 0
0757 CID_1 = [ 13, 0, 16, 0], !Cntlr identifier wrd 1
0758 CID_2 = [ 14, 0, 16, 0], !Cntlr identifier wrd 2
0759 CID_3 = [ 15, 0, 16, 0], !Cntlr identifier wrd 3
0760
0761 !+
0762 ! ONLINE COMMAND command and response
0763 ! parameter field declaration
0764 !-
0765
0766 ! COMMAND FIELD
0767
0768 RSVSD= [ 8, 0, 16, 0], !Reserved
0769 UNT_FLAGS = [ 9, 0, 16, 0], !Unit flag field
0770 RSVSD$0 = [ 10, 0, 16, 0], !Reserved field
0771 RSVSD$1 = [ 11, 0, 16, 0], !Reserved field
0772 RSVSD$2 = [ 12, 0, 16, 0], !Reserved field
0773 RSVSD$3 = [ 13, 0, 16, 0], !Reserved field
0774 RSVSD$4 = [ 14, 0, 16, 0], !Reserved field
0775 RSVSD$5 = [ 15, 0, 16, 0], !Reserved field
0776 DDP_LO = [ 16, 0, 16, 0], !Device dependent parameter
0777 DDP_HI = [ 17, 0, 16, 0], !Device dependent parameter
0778 SHADOW UNIT = [ 18, 0, 16, 0], !Shadow unit
0779 COPY_SPEED = [ 19, 0, 16, 0], !Copy speed
0780
0781 ! RESPONSE FIELD
0782
0783 MUNT CODE = [ 8, 0, 16, 0], !Multi-unit code
0784 ! UNT_FLAGS = [ 9, 0, 16, 0], !Same as cmd field
0785 ! RSVSD$0 = [ 10, 0, 16, 0], !Same as cmd field
0786 ! RSVSD$1 = [ 11, 0, 16, 0], !Same as cmd field
0787 ! UID_0 = [ 12, 0, 16, 0], !Unit ident word 0
0788 ! UID_1 = [ 13, 0, 16, 0], !Unit ident word 1
0789 ! UID_2 = [ 14, 0, 16, 0], !Unit ident word 2
0790 ! UID_3 = [ 15, 0, 16, 0], !Unit ident word 3
0791 ! MTID_LO = [ 16, 0, 16, 0], !Media type ident word 0
0792 ! MTID_HI = [ 17, 0, 16, 0], !Media type ident word
0793 ! SHADOW UNIT = [ 18, 0, 16, 0], !Same as cmd
0794 ! SHA_STATE = [ 19, 0, 16, 0], !Shadow state
0795 ! USZ_LO = [ 20, 0, 16, 0], !Unit size lo word
0796 ! USZ_HI = [ 21, 0, 16, 0], !Unit size hi word
0797 ! VSN_LO = [ 22, 0, 16, 0], !Volume serial num lo word
0798 ! VSN_HI = [ 23, 0, 16, 0], !Volume serial num hi word
0799 tes,
0800

```


RDRX FORMATTER LIBRARY MODULE
FIELD DECLARATIONS

28-Jun-1983 13:01:22
28-Jun-1983 12:55:24

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQBA0.R16;2 (8)

```

0801      !+
0802      ! Send Receive command buffer field definition
0803      !-
0804      RECB_FIELD =
0805          set
0806          MSG_NUM = [0, 0, 12, 0],
0807          MSG_TYP = [0, 12, 4, 0],
0808          MSG_TXT = [1, 0, 16, 0]
0809      tes;
0810
0811      !+
0812      ! Outstanding command buffer field declarations
0813      !-
0814      OUT$FIELD =
0815          set
0816          CMD_WRD = [0, 0, 16, 0],           !Command word ref "word 0 of slot"
0817          REC_FLG = [0, 15, 1, 0],         !Command received indicator flag
0818          CMD_REF = [0, 0, 8, 0],          !Command reference field
0819          ENV_ADR = [1, 0, 16, 0]         !Envelope adrs field
0820      tes;
0821

```

RDRX FORMATTER LIBRARY MODULE
LINKAGE DELCARATIONS

28-Jun-1983 13:01:22
28-Jun-1983 12:55:24

VAX-11 Bliss-16 V3-555
DISK\$USER:[PRUCHA.RELEASE]ZRQBA0.R16;2 (9)

SEQ 208
Page 20

0822 %sbttl 'LINKAGE DELCARATIONS'

0823

0824 Linkage

0825

0826 Call_lnk\$typ

0827

0828 This specifies that the PDP-11 JSR and RTS instructions
0829 are used by the compiled code, and that the parameters
0830 with standard parameters locations are passed using
0831 register 5 (R5) as the argument pointer with the register
0832 usage as follows:

0833

0834 Register

Usage

0835

0836 0 Value return register, non-preserved

0837

0838 1-4 Preserved

0839

0840 5 Argument pointer

0841

0842 6 Stack pointer

0843

0844 7 Program counter

0845

0846 Int_lnk\$typ

0847

0848 Specifies that a routine will be called only by a PDP-11
0849 hardware or software interrupt and will be returned via
0850 the RTI instruction. Register usage is as follows:

0851

0852 Register

Default usage

0853

0854 0-5 Preserved

0855

0856 6 Stack pointer

0857

0858 7 Program pointer

0859

INT_LNK\$TYP = interrupt (standard);

COMMAND QUALIFIERS

: Run Time: 00:06.0
: Elapsed Time: 00:08.7
: Memory Used: 46 pages
: Library Precompilation Complete

ZRQBA0.EXE Memory allocation map TKB M40.02 Page 1
 28-JUN-83 13:08

Partition name : DUMMY
 Identification : REV A
 Task UIC : [202,10]
 Task attributes: -HD
 Total address windows: 1.
 Task image size : 4544. words
 Task address limits: 002000 023547
 R-W disk blk limits: 000002 000023 000022 00018.

*** Root segment: ZRQB1

R/W mem limits: 002000 023547 021550 09064.
 Disk blk limits: 000002 000023 000022 00018.

Memory allocation synopsis:

Section	Title	Ident	File
. BLK.:(RW,I,LCL,REL,CON)	002000	000000	00000.
\$CODE\$(RO,I,LCL,REL,CON)	002000	011266	04790.
	002000	000170	00120. ZRQB1 REV A ZRQB1.OBJ;2
	002170	000716	00462. ZRQB2 REV A ZRQB2.OBJ;1
	003106	001332	00730. ZRQB3 REV A ZRQB3.OBJ;1
	004440	006202	03202. ZRQB4 REV A ZRQB4.OBJ;1
	012642	000316	00206. B16MUL 2.8 ZRQB.OLB;1
	013160	000106	00070. B16SAV 2.4 ZRQB.OLB;1
\$DATES:(RW,I,LCL,REL,CON)	013266	000316	00206.
	013266	000316	00206. RDRX 001 DATCHK.OBJ;1
\$GLOB\$(RO,D,GBL,REL,CON)	013604	001532	00858.
	013604	001532	00858. ZRQB1 REV A ZRQB1.OBJ;2
\$SPLIT\$(RO,D,GBL,REL,CON)	015336	006164	03188.
	015336	006164	03188. ZRQB1 REV A ZRQB1.OBJ;2
\$XYZ\$(RO,I,LCL,REL,CON)	023522	000024	00020.
	023522	000024	00020. ZRQB5 REV A ZRQB5.OBJ;1

Global symbols:

ABORT	006724-R	BIT10	002000	BL\$DIV	013066-R	DEF.DA	016154-R	FMT.ER	021556-R	GPS5	002324-R	LOAD.O	004514-R
ABO.MS	020230-R	BIT11	004000	BL\$LAS	023522-R	DEF.SE	016166-R	FMT1	015336-R	HEAD.A	013654-R	LUN	015262-R
BIT0	000001	BIT12	010000	BL\$MOD	013100-R	DFLT\$.	016006-R	FMT2	015342-R	HW.BR.	002144-R	LSACP	002110-R
BIT00	000001	BIT13	020000	BL\$MUL	012642-R	DFPTBL	002140-R	FMT3	015430-R	HW.IP.	002140-R	LSAPT	002036-R
BIT01	000002	BIT14	040000	BL\$SHF	013112-R	DOWN\$.	015702-R	FMT4	015524-R	HW.Q1.	020664-R	LSAU	003076-R
BIT02	000004	BIT15	100000	BOOT.F	020460-R	DUP\$I.	005612-R	GET.CM	004606-R	HW.Q2.	020714-R	LSAUT	002070-R
BIT03	000010	BIT2	000004	BOOT.R	011736-R	D\$PCNT	002122-R	GET.DU	007222-R	HW.Q3.	020746-R	LSAUTO	003036-R
BIT04	000020	BIT3	000010	BR.LEV	015300-R	EMSG.S	023446-R	GET.NR	004466-R	HW.UNI	002146-R	LSCCP	002106-R
BIT05	000040	BIT4	000020	COM.AR	013604-R	ERRBLK	002134-R	GET.NS	004440-R	HW.VEC	002142-R	LSCLEA	003052-R
BIT06	000100	BIT5	000040	COPY	012574-R	ERRMSG	002132-R	GOOD.N	020366-R	INACC\$	015726-R	LSCO	002032-R
BIT07	000200	BIT6	000100	CTO.WA	006534-R	ERRNBR	002130-R	GPS1	002254-R	INIT.C	012176-R	LSDEPO	002011-R
BIT08	000400	BIT7	000200	DATES.	016116-R	ERRTYP	002126-R	GPS2	002264-R	INT\$I.	011476-R	LSDESC	002170-R
BIT09	001000	BIT8	000400	DATE.C	013316-R	EXIST\$	015636-R	GPS3	002274-R	ISD.ST	015316-R	LSDESP	002076-R
BIT1	000002	BIT9	001000	DECODE	004672-R	EX.LOC	007520-R	GPS4	002312-R	IS.TIM	011500-R	LSDEVP	002060-R

ZRQBA0.EXE Memory allocation map TKB M40.02 Page 2
 ZRQB1 28-JUN-83 13:08

L\$DISP 002124-R	L\$HARD 002254-R	L\$NAME 002000-R	L\$SPCP 002020-R	PID.SA 015264-R	SEND.D 010042-R	T1 004424-R
L\$DLY 002116-R	L\$HIME 002120-R	L\$NDHR 002306-R	L\$SPTP 002024-R	PORT.I 020610-R	SEND.R 013660-R	UC.VER 015266-R
L\$DTP 002040-R	L\$HPCP 002016-R	L\$NDHW 002150-R	L\$STA 002030-R	PROTO. 020542-R	SERIAL 016056-R	UNIT\$. 015602-R
L\$DTYP 002034-R	L\$HPTP 002022-R	L\$NDSF 002336-R	L\$SW 002154-R	PTBL.P 015304-R	SET.CN 011132-R	UNIT.N 015302-R
L\$DU 003064-R	L\$HRDL 002252-R	L\$NDSW 002160-R	L\$SWLE 002152-R	PWR.MS 020146-R	SFPTBL 002154-R	VEC.AD 015276-R
L\$DUT 002072-R	L\$HW 002140-R	L\$PRIO 002042-R	L\$TEST 002114-R	RDRX.A 015274-R	SND.BU 015122-R	\$SEND.L 023544-R
L\$DVTY 002216-R	L\$HWLE 002136-R	L\$PROT 002162-R	L\$TIML 002014-R	RECEIV 013656-R	SND.EN 014262-R	\$SAVE2 013160-R
L\$EF 002052-R	L\$ICP 002104-R	L\$PRT 002112-R	L\$UNIT 002012-R	REC.BU 014542-R	SW.MOD 002156-R	\$SAVE3 013174-R
L\$ENVI 002044-R	L\$INIT 003024-R	L\$REPP 002062-R	MSGADR 014544-R	REC.DA 010476-R	SW.Q1. 020776-R	\$SAVE4 013212-R
L\$ERRT 002126-R	L\$LADP 002026-R	L\$REV 002010-R	NRD.SL 015272-R	REC.EN 013662-R	SW.Q2. 021072-R	\$SAVE5 013232-R
L\$ETP 002102-R	L\$LAST 023526-R	L\$RPT 002342-R	NSD.SL 015270-R	RET.EN 015254-R	SW.UNI 002154-R	
L\$EXP1 002046-R	L\$LOAD 002100-R	L\$SFTL 002310-R	NXT.CR 015256-R	RET.ST 015260-R	TO.MAN 020310-R	
L\$EXP4 002064-R	L\$LUN 002074-R	L\$SOFT 002312-R	OUT\$ST 015234-R	RINGBA 013614-R	T\$FREE 023542-R	
L\$EXP5 002066-R	L\$MREV 002050-R	L\$SPC 002056-R	PFE.ST 020072-R	RSVD.S 015306-R	T\$PTHV 000001	

*** Task builder statistics:

Total work file references: 13352.
 Work file reads: 0.
 Work file writes: 0.
 Size of core pool: 5486. words (21. pages)
 Size of work file: 2560. words (10. pages)

Elapsed time:00:00:09