MODULE ZRQB1 (\TITLE \ 'ADRX DISK FORMATTER'
IDENT = \ 'REV C PATCH 0'
ADDRESSING MODE (ABSOLUTE),
ENVIRONMENT (MDEIS)

BEGIN

IDENTIFICATION

PRODUCT CODE: AC 1566C MC
PRODUCT NAME: CZRQBC0 RDX1 DISK FORMATTER
PRODUCT DATE: 9 APRIL 1984
MAINTAINER: DIAGNOSTIC ENGINEERING
AUTHOR: Doug Neale

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

COPYRIGHT (C) 1983/84 BY DIGITAL EQUIPMENT CORPORATION
THE FOLLOWING ARE TRADEMARKS OF DIGITAL EQUIPMENT CORPORATION:

DIGITAL PDP UNIBUS MASSBUS
DEC DECUS DECTAPE
February 1983  Jay Navin
  Adapted existing AZTEC formatter for use with RODX RDS1
March 1983  Russell Young
  Corrected minor incompatibilities in expectations between controller and host
February 1984  Russell Young
  Updated the main routine to work on either version 8 or 9 of the RODX microcode.
  In version 8 the ASCII strings are stored in the host code because of space
  limitations in the controller. Version 9, for increased generality, has the
  strings held in the controller and transmitted to the host. This still contains
  the original code section to maintain compatibility with version 8.
March 1984  Russell Young
  Removed checks on incoming error and informational messages to allow controller
  modification without reissuing them every time. Also, the routine SETCPU was
  added to allow this to run on an ORION system.
TABLE OF CONTENTS

1.0 GENERAL INFORMATION

1.1 PROGRAM ABSTRACT
  1.1.1 MOST RESIDENT PROGRAM
  1.1.2 CONTROLLER RESIDENT PROGRAM

1.2 SYSTEM REQUIREMENTS
  1.2.1 HARDWARE REQUIREMENTS
  1.2.2 SOFTWARE REQUIREMENTS

1.3 RELATED DOCUMENTS AND STANDARDS

2.0 OPERATING INSTRUCTIONS
  2.1 HARDWARE QUESTIONS
  2.2 SOFTWARE QUESTIONS
  2.3 FORMATTER QUESTIONS

3.0 RUNNING THE FORMATTER
  3.1 ERRORS
  3.2 SUCCESS
1.0 GENERAL INFORMATION

1.1 PROGRAM ABSTRACT

1.1.1 MOST RESIDENT PROGRAM

This program is the front end which invokes the formatter
for the RD51/52 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. The last two steps depend on
the version of the controller present. The host resident program
is designed to run both version 8 and version 9 of the RQDX
controller.

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 RDQX1 CONTROLLER board and attached
RD51/52 WINCHESTER drive(s).

As of rev C of this program, it will also run on an Or'ion
J-11 processor.

1.2.2 SOFTWARE REQUIREMENTS

THIS DIAGNOSTIC IS DESIGNED TO RUN WITH THE DIAGNOSTIC SUPERVISOR
AS DESCRIBE. IN PARAGRAPHS 2.0.
1.3 RELATED DOCUMENTS AND STANDARDS

XXOP, SUPERVISOR/USERS MANUAL, CHQUS
DSDFV12 DEC DISK FORMATTING STANDARDS
2.0 OPERATING INSTRUCTIONS

This is a rev C supervisor diagnostic: for operating instructions, please see chapter 5 of the 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 (1) ?

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 (0) 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 (0) 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.
4. BR LEVEL (D) 4 ?

Answer with the bus request interrupt level used by the above RIDX1 levels 4 through 7 are acceptable. 4 is the default.
2.2 SOFTWARE QUESTIONS

Software Parameter Questions

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

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

2. SOFTWARE QUESTIONS ONLY APPLY UNDER APT. ENTER UNIT NUMBER (0) ?
   This will accept any answer in the range of 0 to 3. It will default to 0 if a CR is struck.

3. ENTER MODE [1 = REFORMAT, 2 = RESTORE, 3 = RECONSTRUCT] (0) ?
   Answering this question with one of the above numbers will cause the formatter to try one of the three formatting modes as explained in 2a, 2b or 2c below.

2.3 FORMATTER QUESTIONS

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

2.3.1 VERSION 8 QUESTIONS

After these DRS set up questions, the version 8 formatter will ask the following manual intervention questions needed to proceed. These questions will not be asked under APT. The default answers contained in the formatter will be used. These answers interact directly with the A2D2X1 Controller.
1. Enter unit number to format (0)

The answer should be in the range of 0 to 3. The default answer is 0.
2. The next three questions select the type of format which will be done. The three modes are explained in the Disk Adapter Functional Specifications: 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 DOS 0301 system, any eight characters will suffice, and on a two DOS/52 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.
2.3.2 VERSION 9 QUESTIONS

1. Enter date <MM-DD-YYYY>
   Date should be entered in the form requested. For example,
   1 29 1958, or
   4-25-1980

2. Enter unit number to format <0>:
   Unit number is entered in decimal, 0 <= x <= 16. Default is 0

3. Use existing bad block information <N>:
   If yes, Execute a REFORMAT mode formatting. That is, try to
   use the RCT which already exists on the disk.

4. Use down-line load <N>:
   A Y here will cause question 5 to be skipped, and
   the controller to prompt for the head, cylinder, and
   byte offset of the bad blocks, as reported by the
   manufacturer.

5. Continue if bad block information is inaccessible <N>:
   If the formatter tries to read the Factory Control Table and
   fails, it will abort the format if this is not answered yes.

6. Enter non-zero serial number:
   Enter one to ten digits. The controller requires an even number,
   so if an odd number are typed a "." is appended to the end.

3.0 RUNNING

---

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

3.1.1 Version 8 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 RDS1/52. 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 INACCESSIBLE
   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.

3.1.2 Version 9 error messages
1. GET STATUS failure
2. Q-PORT send error
3. Unsuccessful command
4. Q-PORT receive error
5. Q-Bus I/O error
6. Formatter initialization error
7. Nonexistent unit number
8. DBN/XBN format error (drive FORMAT command failed)
9. FCT does not have enough good copies of each block
10. SEEK error
11. RCT does not have enough good copies of each block
12. LBN format error (drive FORMAT command failed)
13. FCT write error
14. RCT read error
15. RCT write error
16. RCT full
17. FCT read error
18. FCT nonexistent
19. FCT Down line-load error
20. Drive init timeout
21. Illegal response to start-up question
22. WARNING - possible head addressing problem - run diagnostics
23. INPUT Error
24. Media degraded

3.2 SUCCESS
If all goes well, in about 11 minutes the format will be complete.

3.2.1 VERSION 8 SUCCESS

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.

3.2.2 VERSION 9 SUCCESS

The rev 9 controller will provide additional information and statistics at the end of each run.

1. Format complete
   Signals the end of the formatting process

2. FCT used successfully
   or
   FCT was not used
   Reports if the Factory Control Table was accessible

3. xxx Re vectored LBN's
   The total number of bad blocks found

4. xxx Primary re vectored LBN's
   The number of primary re vectored LBN's. Currently the controller does not support primary re vectoring.

5. xxx Secondary/tertiary re vectored LBN's
   This number should be equal to the total LBNs reported in message 3.

6. xxx Bad blocks in the RCT area due to data errors
   bad blocks found in the Replacement and Caching Table area

7. xxx Bad blocks in the DBN area due to data errors
   Bad blocks found in the Diagnostic Block Number area

8. xxx Bad blocks in the XBN area due to data errors
   Bad blocks in the Extended Block Number area

9. xxx Bad RBN's
   Bad blocks found in the Replacement Block Number area

10. xxx Blocks retried on the check pass
   The number of re vectorings
The structure of a diagnostic program may contain any or all of the ten optional sections. But five of these optional sections require a pointer that is derived by and for the supervisor, and is located in the header block. Therefore, in relation to the effective use of these five pointers, the optional sections call must be coded to reflect usage (i.e., any, all, or none).

The following coding possibilities exist:

POINTER (BGNRPT, BGNSW, BGNSTF, BGNAU, BGNODU, ERRATBL, BGNSETUP)
(Or any subset of the args)

POINTER (ALL): All provides pointers for all five sections.

POINTER (NONE): None indicates to supervisor that no pointers are required.

This is the default.

No pointers are optional using bliss. Make sure the following sections of code are in place (in the correct slots), even if the sections are blank.

ARGUMENT FUNCTION
---------- ----------
RPT REPORT CODE
SW SOFTWARE TABLE
SFT SOFTWARE TABLE QUESTIONS
AU ADD CODE
DU DROP CODE
TBL ERROR TABLE
SETUP ASSEMBLED P TABLES

POINTER (ALL):
The program header section contains general information which describes the major characteristics of the diagnostic program. This includes the program name, and revision and patch-order levels. The header also provides space for an event flag register, and for the storage of pointers, through which the supervisor may find access to other key sections of the program (e.g., dispatch table, initialize and clean-up code, etc.). An argument on the header gives the device type. If this is an XDP-bootable device, this enables the supervisor to provide load medium protection when necessary.

HEADER (ASCII:ZROB, ASCII:C, ASCII:O, 1200, 0, PRI00);
RDX DISK FORMATTER

(dispatch table)

The dispatch table section contains address pointers to the various tests contained within the diagnostic program. This section requires the coding of only the dispatch macro.

`#def dispatch (DSNBR OF TESTS);`

`!Defining Supervisor Error table storage`
The default hardware P Table contains default values of the test-device parameters. The structure of this table is identical to the structure of the hardware P-Tables, and is used as a "template" for building the P-Tables.

```
2068 1 \*\* DEFAULT HARDWARE P TABLE \*\*
2069 1
2070 1 The default hardware P Table contains default values of
2071 1 the test-device parameters. The structure of this table
2072 1 is identical to the structure of the hardware P-Tables,
2073 1 and is used as a "template" for building the P-Tables.
2074 1:
2075 1:
2076 1 global
2077 1
2078 1 MW_IP_ADDRS : word initial (\$'172150'), !Define RDRX Controller IP reg
2079 1 MW VECTOR : word initial (\$'154'), !Define RDRX interrupt vector addr
2080 1 MW_BR_LEVEL : word initial (4), !Define RDRX bus request level
2081 1 MW UNIT_NO : word initial (0), !Define RDRX unit no. to format
2082 1
2083 1 ENDMW;
```
2Q81    RDRX DISK FORMATTER
REV C PATCH 0    SOFTWARE P-TABLE

6 Mar 1984 14:16:44    VAX-11 Bliss 16 V4.0 579
6 Mar 1984 14:16:36    DISKUSER2:[YOUNG_FMT]2Q81.B16.4

; 2084 1  #sbt1 SOFTWARE P-TABLE
; 2085 1  !
; 2086 1  ! The software table contains various data used by the
; 2087 1  ! program as operational parameters. These parameters are
; 2088 1  ! set up at assembly time and may be varied by the operator
; 2089 1  ! at run time.
; 2090 1  !
; 2091 1  !
; 2092 1  !
; 2093 1  ! All software parameter coding is done within the RDRX
; 2094 1  ! DM code. This is per DUR functional spec compliance.
; 2095 1  !
; 2096 1  !
; 2097 1  global
; 2098 1  SW_UNIT_NO : word initial (0), !default unit to 0
; 2099 1  SW_MODE : word initial (3); !default mode to reconstruct
; 2099 1  ENDSW;
2100 1 \texttt{\#protection table}
2101 1!
2102 1 ! This table is used by the runtime
2103 1 ! services to protect the load media.
2104 1
2105 1 ! 1st arg = Offset into P-Table for csr address
2106 1 ! 2nd arg = Offset into P-Table for massbus address
2107 1 ! 3rd arg = Offset into P-Table for drive number
2108 1
2109 1 \texttt{BGNPROT(1,-1,1);} 
2110 1 \texttt{ENDPROT;}
*sbt1 'MODULE DECLARATIONS'

; 2111 1
; 2112 1
; 2113 1
; 2114 1
; 2115 1
; 2116 1
; 2117 1
; 2118 1
; 2119 1
; 2120 1
; 2121 1
; 2122 1
; 2123 1
; 2124 1
; 2125 1
; 2126 1
; 2127 1
; 2128 1
; 2129 1
; 2130 1
; 2131 1
; 2132 1
; 2133 1
; 2134 1
; 2135 1
; 2136 1
; 2137 1
; 2138 1
; 2139 1
; 2140 1
; 2141 1
; 2142 1
; 2143 1
; 2144 1
; 2145 1
; 2146 1
; 2147 1
; 2148 2
; 2149 2
; 2150 2
; 2151 2
; 2152 2
; 2153 2
; 2154 2
; 2155 1
; 2156 1

\* Within BLSMAC.REQ the psect names, plit global and own, 
\* are redefined to be as code. This is done to force the 
\* tlb linker to link the header information starting at 
\* at absolute address 2000. Redefine these psect names 
\* back to their original names for house keeping purposes. 
\* Also change the attributes for the psect "global" so that 
\* global data will not be linked starting at absolute address 
\* 2000.

\* psect
     \* plit = $plitt( global),
     \* global = $glob(norwite, nexecute, global, concatenate),
     \* own = tawni;

\* Structure declarations used within this 
\* module:

structure

\* RDRX register accessing structure. This 
\* structure allows RDRX register accessing 
\* to be transportable between the PDP-11 and 
\* VAX Diagnostic Supervisors.

\* This also defines an access algorithm for 
\* VAX to allow field reference to PBA address 
\* space without generating machine checks.

\* RDRX [0, P, S, E] =
       \begin{verbatim}
       \* begin
       local
       RC15_REG;
       RC15_REG = (.RDRX + #upval+0)<<0, #bval, 0*;
       end
       \* P, S, E;
       \end{verbatim}
The global data section contains data that are used in more than one test or module.

global

: Communication area Declarations

COM_AREA : blockvector [REC_ALLOCATE, SND_ALLOCATE, HDR_SZ, 2, word],
HEAD_AREA : ref block [4, word] field (HDR_FIELD),
RECEIVE_RING : ref blockvector [REC_ALLOCATE, 2, word] field (DSC_FIELD),
SEND_RING : ref blockvector [SND_ALLOCATE, 2, word] field (DSC_FIELD),
RECV_ENVELOPE : blockvector [REC_ALLOCATE, RB_SIZE, 2, word] field (ENV_FIELD),
SEND_ENVELOPE : blockvector [SND_ALLOCATE, SB_SIZE, 2, word] field (ENV_FIELD),
RECV_BUF : block [RECB_SIZE, word] field (RECB_FIELD),
SEND_BUF : vector [SNDB_SIZE, word],
OUTSTD_BUF : BLOCKVECTOR [REC_ALLOCATE, 2, WORD] FIELD (OUTFIELD),
RET_ENIAD : ref block [RB_SIZE, 2, word] field (ENV_FIELD);

global bind

: Diagnostic supervisor printing ascii format strings.

FMT1 = uplit (\asciz'\\M1'),
FMT2 = uplit (\asciz'\\M2\asm\FORMATTING PHYSICAL UNIT \m03'),
FMT3 = uplit (\asciz'\\M3\asm\ALI\ophysical UNIT \m02\asm\FORMAT ABORTED'),
FMT4 = uplit (\asciz'\\M4\asm\FORMAT COMPLETED, \m02\asm\REV\ectored \m05'),
FMT5 = uplit (\asciz'\\M5\asm\FORMAT ABORTED, ERROR NUMBER \m02'),
CRFL = uplit (\asciz'\n'),

: Formatter messages (so formatter can conserve space).

UNIT1_MSG = uplit (\asciz'ENTER UNIT TO BE FORMATTED'),
EXIST1_MSG = uplit (\asciz'USE EXISTING BAD BLOCK INFORMATION'),
DOWN1_MSG = uplit (\asciz'USE DOWN LOAD'),
INACC1_MSG = uplit (\asciz'CONTINUE IF BAD BLOCK INFORMATION IS \INACCURATE'),
DFT1_MSG = uplit (\asciz'\M4\EXISTING BAD BLOCK INFORMATION USED'),
SERIAL1_MSG = uplit (\asciz'ENTER 8 CHARACTER SERIAL NUMBER'),
DATE1_MSG = uplit (\asciz'ENTER DATE IN MM-DD-YY FORMAT'),
DATMSG = uplit (\asciz'ENTER DATE <MM-DD-YYYY'>'),
default strings

DEF_DATE = uplit (\asciz'01-29-58'),
DEF_SERIAL = uplit (\asciz'04502179'),
RING_BASE = COM_AREA [REC_BASE],

MSGADR = RECB_BUF [MSG_TXT];
global

! Miscellaneous data declarations

! Overlay section starting adrs
! Stores next cmd ref number
! Saves various return status codes
! Stores logical unit number being formatted
! Saves process indicator word
! Stores ucode version number
! Next send Descriptor slot
! Next receive Descriptor slot

! Hardware P_Table storage declarations

! Controller register access structure
! Interrupt vector address storage
! Bus request level storage
! Unit number to format storage
! Stores P_Table base address

! Dup Protocol data structures

! Reserved field mask structure declaration

! Reserved SA reg fields definitions
! Step one rsvd field
! Step two rsvd field
! Step three rsvd field
! Step four rsvd & ucode field

! Init Sequence Data Structure declaration

! ISD_STRUCT : blockvector [ 4, 2, word ] field ( ISD_FIELD ) preset ()

! Step one read SA register field declaration

! Error bit
! All step bit fields
! No host inter vec settable adrs
! 22-bit addressing support
! Enhanced diag implementation
! Reserved field

! Step one write SA register field declaration

! Error bit
! Diag wrap around
! Number of Send ring slots pwr's of 2
! Number of Receive ring slots pwr's of 3
2263 1 (BLKO, WRDO, SW IE)= 0,
2264 1 (BLKO, WRDO, SW VADR)= '00000000',
2265 1 !Init Sequence interrupt request
2266 1 !Interrupt vector address
2267 1 !
2268 1 !Step two read SA register field declaration
2269 1 !BLK1, WRDO, ERR_BIT) = 0,
2270 1 !BLK1, WRDO, S1P_FIEL) = '00000000',
2271 1 !BLK1, WRDO, S2P_TYP) = 0,
2272 1 !BLK1, WRDO, S2P_BIT7) = 0,
2273 1 !BLK1, WRDO, S2P_SR) = '00000000',
2274 1 !BLK1, WRDO, S2P_CRIC) = '00000000',
2275 1 !BLK1, WRDO, S2P_SIZ) = '00000000',
2276 1 !BLK1, WRDO, S2P_SRING) = '00000000',
2277 1 !BLK1, WRDO, S2P_SIZ) = '00000000',
2278 1 !BLK1, WRDO, SW_LBASE) = '00000000',
2279 1 !Ring base lower address
2280 1 !
2281 1 !NOTE:
2282 1 !The adapter purge interrupt is loaded within
2283 1 !the bginit code due to the inability to field
2284 1 !select bits <1, 15, 0> from the ringbase adrs.
2285 1 !BLK1, WRDO, S3W_PI) = 0,
2286 1 !Adapter purge interrupt request
2287 1 !
2288 1 !Step three read SA register field declaration
2289 1 !BLK2, WRDO, ERR_BIT) = 0,
2290 1 !BLK2, WRDO, S1P_FIEL) = '00000000',
2291 1 !BLK2, WRDO, S2P_TYP) = 0,
2292 1 !BLK2, WRDO, S2P_BIT7) = 0,
2293 1 !BLK2, WRDO, S2P_SR) = '00000000',
2294 1 !BLK2, WRDO, S2P_CRIC) = '00000000',
2295 1 !BLK2, WRDO, S2P_SIZ) = '00000000',
2296 1 !BLK2, WRDO, S2P_SRING) = '00000000',
2297 1 !BLK2, WRDO, S2P_SIZ) = '00000000',
2298 1 !BLK2, WRDO, S2P_PP) = 0,
2299 1 !Purge & Poll test request
2300 1 !BLK2, WRDO, S3W_RBASE) = 0,
2301 1 !Ring base high address
2302 1 !
2303 1 !Step four read SA register field declaration
2304 1 !BLK3, WRDO, ERR_BIT) = 0,
2305 1 !BLK3, WRDO, S1P_FIEL) = '00000000',
2306 1 !BLK3, WRDO, S2P_TYP) = 0,
2307 1 !BLK3, WRDO, S2P_SR) = '00000000',
2308 1 !BLK3, WRDO, S2P_CRIC) = '00000000',
2309 1 !BLK3, WRDO, S2P_SIZ) = '00000000',
2310 1 !BLK3, WRDO, S2P_SRING) = '00000000',
2311 1 !BLK3, WRDO, S2P_PP) = 0,
2312 1 !Last fail request
2313 1 !BLK3, WRDO, S4W_GO) = 0);
global bind

Self-detected fatal port/controller errors

PFE_STRUCT * uplit( 
  uplit (asciz'UNMAINTLERR- UNRECOGNIZABLE ERROR CODE'), 
  uplit (asciz'UNMAINTLERR- ENVELOPE/PACKET READ (PARITY OR TIMEOUT)'), 
  uplit (asciz'UNMAINTLERR- ENVELOPE/PACKET WRITE (PARITY OR TIMEOUT)'), 
  uplit (asciz'UNMAINTLERR- CONTROLLER ROM AND RAM PARITY'), 
  uplit (asciz'UNMAINTLERR- CONTROLLER RAM PARITY'), 
  uplit (asciz'UNMAINTLERR- CONTROLLER ROM PARITY'), 
  uplit (asciz'UNMAINTLERR- RING READ (PARITY OR TIMEOUT)'), 
  uplit (asciz'UNMAINTLERR- RING WRITE (PARITY OR TIMEOUT)'), 
  uplit (asciz'UNMAINTLERR- INTERRUPT MASTER'), 
  uplit (asciz'UNMAINTLERR- MOST ACCESS TIMEOUT'), 
  uplit (asciz'UNMAINTLERR- CREDIT LIMIT EXCEEDED'), 
  uplit (asciz'UNMAINTLERR- UNIBUS MASTER ERROR'), 
  uplit (asciz'UNMAINTLERR- DIAGNOSTIC CONTROLLER FATAL ERROR'), 
  uplit (asciz'UNMAINTLERR- INSTRUCTION LOOP TIMEOUT'), 
  uplit (asciz'UNMAINTLERR- INVALID CONNECTION IDENTIFIER'), 
  uplit (asciz'UNMAINTLERR- INTERRUPT WRITE'), 
  uplit (asciz'UNMAINTLERR- MAINTENANCE READ/WRITE INVALID REGION IDENTIFIER'), 
  uplit (asciz'UNMAINTLERR- MAINTENANCE WRITE LOAD TO NON-LOADABLE CONTROLLER'), 
  uplit (asciz'UNMAINTLERR- CONTROLLER RAM ERROR (NON-PARITY)'), 
  uplit (asciz'UNMAINTLERR- INIT SEQUENCE ERROR'), 
  uplit (asciz'UNMAINTLERR- HIGH-LEVEL PROTOCOL INCOMPATIBILITY ERROR'), 
  uplit (asciz'UNMAINTLERR- PURGE/POLL HARDWARE FAILURE') : vector [22], 

Init code error and informational messages

PWR_MSG = uplit (asciz'PWR_FAIL- INIT CODE RE-ENTERED DUE TO PWR FAIL'), 
ABO_MSG = uplit (asciz'ABORT- ABORTING HOST AND REMOTE PROGRAMS'), 
TO_MANY_UNITS = uplit (asciz'ILLEGAL NUMBER OF UNITS SELECTED'), 
GOOD_NUM_UNITS = uplit (asciz'LIMIT OF SIXTEEN UNITS PER FORMATING SESSION'), 
BOOT_FAILURE = uplit (asciz'RDRX CONTROLLER INITIALIZATION ERROR'), 
PROTO_VIOLATION = uplit (asciz'PROTOCOL VIOLATION ERROR'), 
PORT_INIT_ERR = uplit (asciz'COMMUNICATION AREA INIT ERROR'), 

Local load media DM module file name.ext

DM_FNAME = UPLIT (ASCIZ'AFMTR.SAV'), 

Hardware parameter coding questions

HW_Q1 IP = uplit (asciz'IP REGISTER ADDRESS'),
HW_Q2 VECTOR = uplit ('\ascii:INTERRUPT VECTOR ADDRESS').
HW_Q3 BR = uplit ('\ascii:BUS REQUEST LEVEL').
SW_Q1_UNIT = uplit ('\ascii:SOFTWARE QUESTIONS ONLY APPLY UNDER API, ENTER UNIT NUMBER'),
SW_Q2_MODE = uplit ('\ascii:ENTER MODE [1 = REFORMAT, 2 = RESTORE, 3 = RECONSTRUCT]').

Program flow ascii string messages

! Formater error returns
FMT_ERR = uplit ( 
uplit ('\ascii:FAILURE CLOSING FCTS').
uplit ('\ascii:NAWD WRITE FAILUR').
uplit ('\ascii:NAWEVECTOR LIMIT EXCEEDED').
uplit ('\ascii:NAWSEEK FAILUR DURING ACTUAL FORMATTING').
uplit ('\ascii:NAFACTOR BAD BLOCK INFORMATION IS INACCESSABLE').
uplit ('\ascii:NAFACTOR FAILURE ACCESSING FCT').
uplit ('\ascii:NAUNIT IS NOT WINCHESTER OR CAN NOT BE SELECTED'): vector [7].

! Error message structure
EMSG_STRUCT = uplit ( 
uplit ('\ascii:FAILERR RESPONSE STATUS ERROR').
uplit ('\ascii:FAILERR HOST/CONTROLLER OUT OF SEQ').
uplit ('\ascii:FAILERR REMOTE PROG NOT RUNNING').
uplit ('\ascii:FAILERR UNKNOWN RETURN STATUS CODE').
uplit ('\ascii:FAILERR CON AREA INIT ERROR').
uplit ('\ascii:FAILERR PORT/MOST SYNC ERROR').
uplit ('\ascii:FAILERR MESSAGE LENGTH ERROR').
uplit ('\ascii:FAILERR UNKNOWN ENCODE RECEIVED').
uplit ('\ascii:FAILERR ADAPTOR PURGE ERROR').
uplit ('\ascii:FAILERR UNKNOWN INTERRUPT').
uplit ('\ascii:FAILERR INIT SEQ TIME OUT').
uplit ('\ascii:FAILERR INIT SEQ COMPARE ERROR').
uplit ('\ascii:FAILERR UNEXPECTED ATTENTION END MESSAGE RECEIVED').
uplit ('\ascii:FAILERR UNEXPECTED COMMANDPCODE IN END MESSAGE RECEIVED').
uplit ('\ascii:FAILERR UNEXPECTED EXCEPTION END MESSAGE RECEIVED').
uplit ('\ascii:FAILERR INVALID COMMAND END MESSAGE RECEIVED').
uplit ('\ascii:FAILERR UNKNOWN MESSAGE RECEIVED').
uplit ('\ascii:FAILERR OUTSTANDING COMMAND BUFFER FULL').,
.TITLE ZRQB1 RDRX DISK FORMATTER
.IDENT "REV C/
.ENABLE AEA

000000 132 122 121}
000000 102
000004 000
000005 000
000006 000
000007 000
000010 000
000010 103
000011 060
000012 000000G
000014 002260
000016 000000G
000020 000000G
000022 000142
000024 000156
000026 000000G
000030 000000
000032 000000
000034 000000
000036 000000
000040 000126
000042 000000
000044 000000
000046 000000
000050 003
000051 003
000052 000000
000054 000000
000056 000000
000060 000000G
000062 000000G
000064 000000
000066 000000
000070 000000G
000072 000000G
000074 000000

L1NAME:.PSEC1 CODE1, RO
.L1UNIT:.WORD TIPTHV
.L1TIME:.WORD 2260
.L1UPC:.WORD L1HARD
.L1SPC:.WORD L1SOFT
.L1HPT:.WORD L1HW
.L1SPT:.WORD L1SW
.L1LAD:.WORD L1LAST
.L1STA:.WORD 0
.L1CO:.WORD 0
.L1DTP:.WORD 0
.L1APT:.WORD 0
.L1DTP:.WORD L1DISPATCH
.L1PRIO:.WORD 0
.L1ENVI:.WORD 0
.L1EXP1:.WORD 0
.L1HREV:.WORD 0
.BYTE 3
.BYTE 3
.L1EF:.WORD 0
.BYTE 0
.L1SPC:.WORD 0
.L1DEV:.WORD L1DVTYP
.L1REPP:.WORD L1RPT
.L1EXP:.WORD 0
.L1EXP:.WORD 0
.L1AUT:.WORD L1AU
.L1DUT:.WORD L1DU
.L1LUN:.WORD 0
<table>
<thead>
<tr>
<th>Offset</th>
<th>Character</th>
<th>Type</th>
<th>Offset</th>
<th>Character</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>000034</td>
<td>125</td>
<td>116</td>
<td>111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000037</td>
<td>124</td>
<td>040</td>
<td>045</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000042</td>
<td>104</td>
<td>063</td>
<td>000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000045</td>
<td>000</td>
<td>000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000046</td>
<td>045</td>
<td>116</td>
<td>045</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000047</td>
<td>P, AAC:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000051</td>
<td>045</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000054</td>
<td>107</td>
<td>111</td>
<td>103</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000057</td>
<td>101</td>
<td>114</td>
<td>040</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000062</td>
<td>125</td>
<td>116</td>
<td>111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000065</td>
<td>124</td>
<td>040</td>
<td>045</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000070</td>
<td>104</td>
<td>062</td>
<td>045</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000073</td>
<td>123</td>
<td>045</td>
<td>101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000076</td>
<td>120</td>
<td>110</td>
<td>131</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000101</td>
<td>123</td>
<td>111</td>
<td>103</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000104</td>
<td>101</td>
<td>114</td>
<td>040</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000107</td>
<td>125</td>
<td>116</td>
<td>111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000112</td>
<td>124</td>
<td>040</td>
<td>045</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000115</td>
<td>104</td>
<td>062</td>
<td>045</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000120</td>
<td>123</td>
<td>045</td>
<td>101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000123</td>
<td>106</td>
<td>117</td>
<td>122</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000126</td>
<td>115</td>
<td>101</td>
<td>124</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000131</td>
<td>040</td>
<td>101</td>
<td>102</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000134</td>
<td>117</td>
<td>122</td>
<td>124</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000137</td>
<td>105</td>
<td>104</td>
<td>000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000142</td>
<td>045</td>
<td>116</td>
<td>045</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000145</td>
<td>101</td>
<td>106</td>
<td>117</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000150</td>
<td>122</td>
<td>115</td>
<td>101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000153</td>
<td>124</td>
<td>040</td>
<td>103</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000156</td>
<td>117</td>
<td>115</td>
<td>120</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000161</td>
<td>114</td>
<td>105</td>
<td>124</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000164</td>
<td>105</td>
<td>104</td>
<td>054</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000167</td>
<td>040</td>
<td>045</td>
<td>104</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000172</td>
<td>062</td>
<td>045</td>
<td>123</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000175</td>
<td>045</td>
<td>101</td>
<td>040</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000200</td>
<td>122</td>
<td>105</td>
<td>126</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000203</td>
<td>105</td>
<td>103</td>
<td>124</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000206</td>
<td>117</td>
<td>122</td>
<td>105</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000211</td>
<td>104</td>
<td>040</td>
<td>114</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000214</td>
<td>102</td>
<td>116</td>
<td>123</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000217</td>
<td>000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000220</td>
<td>045</td>
<td>116</td>
<td>000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000223</td>
<td>000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000224</td>
<td>105</td>
<td>116</td>
<td>124</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000227</td>
<td>105</td>
<td>122</td>
<td>040</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000230</td>
<td>125</td>
<td>116</td>
<td>111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000235</td>
<td>124</td>
<td>040</td>
<td>124</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000240</td>
<td>117</td>
<td>040</td>
<td>102</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000243</td>
<td>105</td>
<td>040</td>
<td>106</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000246</td>
<td>117</td>
<td>122</td>
<td>115</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000251</td>
<td>101</td>
<td>124</td>
<td>124</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000254</td>
<td>105</td>
<td>104</td>
<td>000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000257</td>
<td>000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000260</td>
<td>125</td>
<td>123</td>
<td>105</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000263</td>
<td>000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- P.AAC: ASCII / UNI/
- P.AAD: ASCII / UNH/
- P.AAE: ASCII / UNX/<00>
- P.AAF: ASCII / ENI/
- P.AAG: ASCII / USE/
<table>
<thead>
<tr>
<th>ZRQB1 REV C PATCH 0</th>
<th>RDRx DISK FORMATTER GLOBAL TEXT SECTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>000263 040 103 150</td>
<td>.ASCII / EX/</td>
</tr>
<tr>
<td>000266 111 123 124</td>
<td>.ASCII / IST/</td>
</tr>
<tr>
<td>000271 111 116 107</td>
<td>.ASCII / ING/</td>
</tr>
<tr>
<td>000274 040 102 101</td>
<td>.ASCII / BA/</td>
</tr>
<tr>
<td>000277 104 040 102</td>
<td>.ASCII / D B/</td>
</tr>
<tr>
<td>000302 114 117 103</td>
<td>.ASCII / LOC/</td>
</tr>
<tr>
<td>000305 113 040 111</td>
<td>.ASCII / K I/</td>
</tr>
<tr>
<td>000310 116 106 101</td>
<td>.ASCII / INFO/</td>
</tr>
<tr>
<td>000313 122 115 101</td>
<td>.ASCII / RAM/</td>
</tr>
<tr>
<td>000316 124 111 117</td>
<td>.ASCII / TIO/</td>
</tr>
<tr>
<td>000321 116 000 000</td>
<td>.ASCII / N/&lt;&lt;00&lt;&lt;00&gt;</td>
</tr>
<tr>
<td>000324 125 123 105</td>
<td>.ASCII / USE/</td>
</tr>
<tr>
<td>000327 040 104 117</td>
<td>.ASCII / DO/</td>
</tr>
<tr>
<td>000332 127 116 040</td>
<td>.ASCII / WN/</td>
</tr>
<tr>
<td>000335 114 111 116</td>
<td>.ASCII / LIN/</td>
</tr>
<tr>
<td>000340 105 040 114</td>
<td>.ASCII / E L/</td>
</tr>
<tr>
<td>000343 117 101 104</td>
<td>.ASCII / OAD/</td>
</tr>
<tr>
<td>000346 000 000 000</td>
<td>.ASCII / &lt;&lt;00&lt;&lt;00&gt;</td>
</tr>
<tr>
<td>000350 103 117 116</td>
<td>.ASCII / CON/</td>
</tr>
<tr>
<td>000353 124 111 116</td>
<td>.ASCII / TIN/</td>
</tr>
<tr>
<td>000356 125 105 040</td>
<td>.ASCII / U/</td>
</tr>
<tr>
<td>000361 111 106 040</td>
<td>.ASCII / IF/</td>
</tr>
<tr>
<td>000364 102 101 040</td>
<td>.ASCII / BAD/</td>
</tr>
<tr>
<td>000367 040 102 114</td>
<td>.ASCII / BL/</td>
</tr>
<tr>
<td>000372 117 103 113</td>
<td>.ASCII / OCK/</td>
</tr>
<tr>
<td>000375 040 111 116</td>
<td>.ASCII / IN/</td>
</tr>
<tr>
<td>000400 106 117 122</td>
<td>.ASCII / FDR/</td>
</tr>
<tr>
<td>000403 115 101 124</td>
<td>.ASCII / MAT/</td>
</tr>
<tr>
<td>000406 111 117 116</td>
<td>.ASCII / IDN/</td>
</tr>
<tr>
<td>000411 040 111 123</td>
<td>.ASCII / IS/</td>
</tr>
<tr>
<td>000414 040 111 116</td>
<td>.ASCII / IN/</td>
</tr>
<tr>
<td>000417 101 103 103</td>
<td>.ASCII / ACC/</td>
</tr>
<tr>
<td>000422 125 122 101</td>
<td>.ASCII / URA/</td>
</tr>
<tr>
<td>000425 124 105 000</td>
<td>.ASCII / TE/&lt;&lt;00&gt;</td>
</tr>
<tr>
<td>000430 045 116 045</td>
<td>.ASCII / &lt;&lt;00/</td>
</tr>
<tr>
<td>000433 101 105 130</td>
<td>.ASCII / AEX/</td>
</tr>
<tr>
<td>000436 111 123 124</td>
<td>.ASCII / IST/</td>
</tr>
<tr>
<td>000441 111 116 107</td>
<td>.ASCII / ING/</td>
</tr>
<tr>
<td>000444 040 102 101</td>
<td>.ASCII / BA/</td>
</tr>
<tr>
<td>000447 104 040 102</td>
<td>.ASCII / D B/</td>
</tr>
<tr>
<td>000452 114 117 103</td>
<td>.ASCII / LOC/</td>
</tr>
<tr>
<td>000455 113 040 111</td>
<td>.ASCII / K I/</td>
</tr>
<tr>
<td>000460 115 106 117</td>
<td>.ASCII / INFO/</td>
</tr>
<tr>
<td>000463 122 115 101</td>
<td>.ASCII / RAM/</td>
</tr>
<tr>
<td>000466 124 111 117</td>
<td>.ASCII / TIO/</td>
</tr>
<tr>
<td>000471 116 040 125</td>
<td>.ASCII / N U/</td>
</tr>
<tr>
<td>000474 123 105 104</td>
<td>.ASCII / SED/</td>
</tr>
<tr>
<td>000477 000 000 000</td>
<td>.ASCII / &lt;&lt;00&gt;</td>
</tr>
<tr>
<td>000500 105 116 124</td>
<td>.ASCII / EN/</td>
</tr>
<tr>
<td>000503 105 122 040</td>
<td>.ASCII / ER/</td>
</tr>
<tr>
<td>000506 070 040 103</td>
<td>.ASCII / B C/</td>
</tr>
<tr>
<td>000511 110 101 122</td>
<td>.ASCII / MAR/</td>
</tr>
<tr>
<td>000514 101 103 124</td>
<td>.ASCII / ACT/</td>
</tr>
<tr>
<td>ZRQB1</td>
<td>RDRX DISK FORMATTER</td>
</tr>
<tr>
<td>-------</td>
<td>---------------------</td>
</tr>
<tr>
<td>001203</td>
<td>105</td>
</tr>
<tr>
<td>001206</td>
<td>122</td>
</tr>
<tr>
<td>001211</td>
<td>040</td>
</tr>
<tr>
<td>001214</td>
<td>122</td>
</tr>
<tr>
<td>001217</td>
<td>131</td>
</tr>
<tr>
<td>001222</td>
<td>045</td>
</tr>
<tr>
<td>001225</td>
<td>101</td>
</tr>
<tr>
<td>001230</td>
<td>124</td>
</tr>
<tr>
<td>001233</td>
<td>122</td>
</tr>
<tr>
<td>001236</td>
<td>040</td>
</tr>
<tr>
<td>001241</td>
<td>116</td>
</tr>
<tr>
<td>001244</td>
<td>117</td>
</tr>
<tr>
<td>001247</td>
<td>105</td>
</tr>
<tr>
<td>001252</td>
<td>122</td>
</tr>
<tr>
<td>001255</td>
<td>040</td>
</tr>
<tr>
<td>001260</td>
<td>122</td>
</tr>
<tr>
<td>001263</td>
<td>131</td>
</tr>
<tr>
<td>001266</td>
<td>045</td>
</tr>
<tr>
<td>001271</td>
<td>101</td>
</tr>
<tr>
<td>001274</td>
<td>124</td>
</tr>
<tr>
<td>001277</td>
<td>122</td>
</tr>
<tr>
<td>001302</td>
<td>040</td>
</tr>
<tr>
<td>001305</td>
<td>116</td>
</tr>
<tr>
<td>001310</td>
<td>122</td>
</tr>
<tr>
<td>001313</td>
<td>104</td>
</tr>
<tr>
<td>001316</td>
<td>120</td>
</tr>
<tr>
<td>001321</td>
<td>111</td>
</tr>
<tr>
<td>001324</td>
<td>040</td>
</tr>
<tr>
<td>001327</td>
<td>040</td>
</tr>
<tr>
<td>001332</td>
<td>115</td>
</tr>
<tr>
<td>001335</td>
<td>125</td>
</tr>
<tr>
<td>001340</td>
<td>000</td>
</tr>
<tr>
<td>001342</td>
<td>045</td>
</tr>
<tr>
<td>001345</td>
<td>101</td>
</tr>
<tr>
<td>001350</td>
<td>124</td>
</tr>
<tr>
<td>001353</td>
<td>122</td>
</tr>
<tr>
<td>001356</td>
<td>040</td>
</tr>
<tr>
<td>001361</td>
<td>116</td>
</tr>
<tr>
<td>001364</td>
<td>127</td>
</tr>
<tr>
<td>001367</td>
<td>124</td>
</tr>
<tr>
<td>001372</td>
<td>050</td>
</tr>
<tr>
<td>001375</td>
<td>122</td>
</tr>
<tr>
<td>001380</td>
<td>131</td>
</tr>
<tr>
<td>001383</td>
<td>122</td>
</tr>
<tr>
<td>001396</td>
<td>117</td>
</tr>
<tr>
<td>001411</td>
<td>117</td>
</tr>
<tr>
<td>001414</td>
<td>051</td>
</tr>
<tr>
<td>001416</td>
<td>045</td>
</tr>
<tr>
<td>001421</td>
<td>101</td>
</tr>
<tr>
<td>001424</td>
<td>124</td>
</tr>
<tr>
<td>001427</td>
<td>122</td>
</tr>
<tr>
<td>001432</td>
<td>040</td>
</tr>
<tr>
<td>001435</td>
<td>124</td>
</tr>
<tr>
<td>ZRQB1</td>
<td>REV  C PATCH 0</td>
</tr>
<tr>
<td>-------</td>
<td>----------------</td>
</tr>
</tbody>
</table>

<p>| 001440 | 122 | 125 | 120 | ASCII /RUP/ |
| 001443 | 124 | 040 | 115 | ASCII /T M/ |
| 001446 | 101 | 123 | 124 | ASCII /AST/ |
| 001451 | 105 | 122 | 000 | ASCII /ER/&lt;00&gt; |
| 001454 | 045 | 116 | 045 | ASCII /SNM/ |
| 001457 | 101 | 044 | 106 | ASCII /AIF/ |
| 001462 | 124 | 114 | 105 | ASCII /MLE/ |
| 001465 | 124 | 122 | 055 | ASCII /RR/ |
| 001470 | 040 | 110 | 117 | ASCII /HO/ |
| 001473 | 123 | 124 | 040 | ASCII /ST/ |
| 001476 | 101 | 103 | 103 | ASCII /ACC/ |
| 001501 | 105 | 123 | 123 | ASCII /ESS/ |
| 001504 | 040 | 124 | 111 | ASCII /TI/ |
| 001507 | 115 | 105 | 117 | ASCII /ME0/ |
| 001512 | 125 | 124 | 000 | ASCII /UT/&lt;00&gt; |
| 001515 | 000 |     |     | ASCII /&lt;0&gt;/ |
| 001516 | 045 | 116 | 045 | ASCII /MN/ |
| 001521 | 101 | 044 | 106 | ASCII /AIF/ |
| 001524 | 124 | 114 | 105 | ASCII /MLE/ |
| 001527 | 122 | 122 | 055 | ASCII /RR/ |
| 001532 | 040 | 103 | 122 | ASCII /CR/ |
| 001535 | 105 | 104 | 111 | ASCII /EDI/ |
| 001540 | 124 | 040 | 114 | ASCII /TL/ |
| 001543 | 111 | 115 | 111 | ASCII /SM/ |
| 001546 | 124 | 040 | 105 | ASCII /TE/ |
| 001551 | 130 | 103 | 105 | ASCII /XCE/ |
| 001554 | 105 | 104 | 105 | ASCII /EDE/ |
| 001557 | 104 | 000 | 000 | ASCII /&lt;0&gt;&lt;0&gt;&lt;0&gt; |
| 001562 | 045 | 116 | 045 | ASCII /MN/ |
| 001565 | 101 | 044 | 106 | ASCII /AIF/ |
| 001570 | 124 | 114 | 105 | ASCII /MLE/ |
| 001573 | 122 | 122 | 055 | ASCII /RR/ |
| 001576 | 040 | 125 | 116 | ASCII /LM/ |
| 001601 | 111 | 102 | 125 | ASCII /BU/ |
| 001604 | 123 | 040 | 115 | ASCII /SM/ |
| 001607 | 101 | 123 | 124 | ASCII /AST/ |
| 001612 | 105 | 122 | 040 | ASCII /ER/ |
| 001615 | 105 | 122 | 122 | ASCII /ERR/ |
| 001620 | 117 | 122 | 000 | ASCII /OR/&lt;0&gt; |
| 001623 | 000 |     |     | ASCII /&lt;0&gt;/ |
| 001624 | 045 | 116 | 045 | ASCII /MN/ |
| 001627 | 101 | 044 | 106 | ASCII /AIF/ |
| 001632 | 124 | 114 | 105 | ASCII /MLE/ |
| 001635 | 122 | 122 | 055 | ASCII /RR/ |
| 001640 | 040 | 104 | 111 | ASCII /DF/ |
| 001643 | 101 | 107 | 116 | ASCII /AGN/ |
| 001646 | 117 | 123 | 124 | ASCII /DST/ |
| 001651 | 111 | 103 | 040 | ASCII /IC/ |
| 001654 | 103 | 117 | 116 | ASCII /CON/ |
| 001657 | 124 | 122 | 117 | ASCII /TR/ |
| 001662 | 114 | 114 | 105 | ASCII /LLE/ |
| 001665 | 122 | 040 | 106 | ASCII /RF/ |
| 001670 | 101 | 124 | 101 | ASCII /ATA/ |</p>
<table>
<thead>
<tr>
<th>Offset</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001673</td>
<td>114 040 105</td>
</tr>
<tr>
<td>0001676</td>
<td>122 122 117</td>
</tr>
<tr>
<td>0001701</td>
<td>122 000 000</td>
</tr>
<tr>
<td>0001704</td>
<td>045 116 045</td>
</tr>
<tr>
<td>0001707</td>
<td>161 044 106</td>
</tr>
<tr>
<td>0001712</td>
<td>124 114 105</td>
</tr>
<tr>
<td>0001715</td>
<td>122 122 055</td>
</tr>
<tr>
<td>0001720</td>
<td>040 111 116</td>
</tr>
<tr>
<td>0001723</td>
<td>123 124 122</td>
</tr>
<tr>
<td>0001726</td>
<td>125 103 124</td>
</tr>
<tr>
<td>0001731</td>
<td>111 117 116</td>
</tr>
<tr>
<td>0001734</td>
<td>040 114 117</td>
</tr>
<tr>
<td>0001737</td>
<td>117 120 040</td>
</tr>
<tr>
<td>0001742</td>
<td>124 111 115</td>
</tr>
<tr>
<td>0001745</td>
<td>105 117 125</td>
</tr>
<tr>
<td>0001750</td>
<td>124 000 000</td>
</tr>
<tr>
<td>0001752</td>
<td>045 116 045</td>
</tr>
<tr>
<td>0001755</td>
<td>101 044 106</td>
</tr>
<tr>
<td>0001760</td>
<td>124 114 105</td>
</tr>
<tr>
<td>0001763</td>
<td>122 122 055</td>
</tr>
<tr>
<td>0001766</td>
<td>040 111 116</td>
</tr>
<tr>
<td>0001771</td>
<td>126 101 114</td>
</tr>
<tr>
<td>0001774</td>
<td>111 104 040</td>
</tr>
<tr>
<td>0001777</td>
<td>103 117 116</td>
</tr>
<tr>
<td>0002002</td>
<td>116 105 103</td>
</tr>
<tr>
<td>0002005</td>
<td>124 111 117</td>
</tr>
<tr>
<td>0002010</td>
<td>116 040 111</td>
</tr>
<tr>
<td>0002013</td>
<td>104 105 116</td>
</tr>
<tr>
<td>0002016</td>
<td>124 111 106</td>
</tr>
<tr>
<td>0002021</td>
<td>111 105 122</td>
</tr>
<tr>
<td>0002024</td>
<td>040 000 000</td>
</tr>
<tr>
<td>0002026</td>
<td>045 116 045</td>
</tr>
<tr>
<td>0002031</td>
<td>101 044 106</td>
</tr>
<tr>
<td>0002034</td>
<td>124 114 105</td>
</tr>
<tr>
<td>0002037</td>
<td>122 122 055</td>
</tr>
<tr>
<td>0002042</td>
<td>040 111 116</td>
</tr>
<tr>
<td>0002045</td>
<td>124 105 122</td>
</tr>
<tr>
<td>0002050</td>
<td>122 125 120</td>
</tr>
<tr>
<td>0002053</td>
<td>124 040 127</td>
</tr>
<tr>
<td>0002056</td>
<td>122 111 124</td>
</tr>
<tr>
<td>0002061</td>
<td>105 000 000</td>
</tr>
<tr>
<td>0002064</td>
<td>045 116 045</td>
</tr>
<tr>
<td>0002067</td>
<td>101 044 106</td>
</tr>
<tr>
<td>0002072</td>
<td>124 114 105</td>
</tr>
<tr>
<td>0002075</td>
<td>122 122 055</td>
</tr>
<tr>
<td>0002100</td>
<td>040 115 101</td>
</tr>
<tr>
<td>0002103</td>
<td>111 116 124</td>
</tr>
<tr>
<td>0002106</td>
<td>105 116 101</td>
</tr>
<tr>
<td>0002111</td>
<td>116 103 105</td>
</tr>
<tr>
<td>0002114</td>
<td>040 122 105</td>
</tr>
<tr>
<td>0002117</td>
<td>101 104 057</td>
</tr>
<tr>
<td>0002122</td>
<td>127 122 111</td>
</tr>
<tr>
<td>0002125</td>
<td>124 105 040</td>
</tr>
<tr>
<td>Address</td>
<td>Value</td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>002364</td>
<td>123</td>
</tr>
<tr>
<td>002367</td>
<td>125</td>
</tr>
<tr>
<td>002372</td>
<td>103</td>
</tr>
<tr>
<td>002375</td>
<td>105</td>
</tr>
<tr>
<td>002400</td>
<td>117</td>
</tr>
<tr>
<td>002403</td>
<td>000</td>
</tr>
<tr>
<td>002404</td>
<td>045</td>
</tr>
<tr>
<td>002407</td>
<td>101</td>
</tr>
<tr>
<td>002412</td>
<td>124</td>
</tr>
<tr>
<td>002415</td>
<td>122</td>
</tr>
<tr>
<td>002420</td>
<td>110</td>
</tr>
<tr>
<td>002423</td>
<td>110</td>
</tr>
<tr>
<td>002426</td>
<td>114</td>
</tr>
<tr>
<td>002431</td>
<td>114</td>
</tr>
<tr>
<td>002434</td>
<td>120</td>
</tr>
<tr>
<td>002437</td>
<td>124</td>
</tr>
<tr>
<td>002442</td>
<td>117</td>
</tr>
<tr>
<td>002445</td>
<td>116</td>
</tr>
<tr>
<td>002450</td>
<td>115</td>
</tr>
<tr>
<td>002453</td>
<td>101</td>
</tr>
<tr>
<td>002456</td>
<td>111</td>
</tr>
<tr>
<td>002461</td>
<td>111</td>
</tr>
<tr>
<td>002464</td>
<td>040</td>
</tr>
<tr>
<td>002467</td>
<td>122</td>
</tr>
<tr>
<td>002472</td>
<td>000</td>
</tr>
<tr>
<td>002474</td>
<td>045</td>
</tr>
<tr>
<td>002477</td>
<td>101</td>
</tr>
<tr>
<td>002502</td>
<td>124</td>
</tr>
<tr>
<td>002505</td>
<td>122</td>
</tr>
<tr>
<td>002510</td>
<td>040</td>
</tr>
<tr>
<td>002513</td>
<td>122</td>
</tr>
<tr>
<td>002516</td>
<td>057</td>
</tr>
<tr>
<td>002521</td>
<td>114</td>
</tr>
<tr>
<td>002524</td>
<td>110</td>
</tr>
<tr>
<td>002527</td>
<td>104</td>
</tr>
<tr>
<td>002532</td>
<td>122</td>
</tr>
<tr>
<td>002535</td>
<td>106</td>
</tr>
<tr>
<td>002540</td>
<td>114</td>
</tr>
<tr>
<td>002543</td>
<td>105</td>
</tr>
<tr>
<td>002546</td>
<td>1000564</td>
</tr>
<tr>
<td>002550</td>
<td>000724</td>
</tr>
<tr>
<td>002552</td>
<td>001012</td>
</tr>
<tr>
<td>002554</td>
<td>001102</td>
</tr>
<tr>
<td>002556</td>
<td>001156</td>
</tr>
<tr>
<td>002560</td>
<td>001222</td>
</tr>
<tr>
<td>002562</td>
<td>001266</td>
</tr>
<tr>
<td>002564</td>
<td>001342</td>
</tr>
<tr>
<td>002566</td>
<td>001416</td>
</tr>
<tr>
<td>002570</td>
<td>001454</td>
</tr>
<tr>
<td>002572</td>
<td>001516</td>
</tr>
<tr>
<td>002574</td>
<td>001562</td>
</tr>
<tr>
<td>002576</td>
<td>001624</td>
</tr>
<tr>
<td>002600</td>
<td>001704</td>
</tr>
<tr>
<td>ZRQ81</td>
<td>RDRX DISK FORMATTER</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------</td>
</tr>
<tr>
<td>003030</td>
<td>040</td>
</tr>
<tr>
<td>003033</td>
<td>114</td>
</tr>
<tr>
<td>003036</td>
<td>124</td>
</tr>
<tr>
<td>003041</td>
<td>000</td>
</tr>
<tr>
<td>003042</td>
<td>045</td>
</tr>
<tr>
<td>003045</td>
<td>101</td>
</tr>
<tr>
<td>003050</td>
<td>124</td>
</tr>
<tr>
<td>003053</td>
<td>122</td>
</tr>
<tr>
<td>003056</td>
<td>040</td>
</tr>
<tr>
<td>003061</td>
<td>115</td>
</tr>
<tr>
<td>003064</td>
<td>040</td>
</tr>
<tr>
<td>003067</td>
<td>040</td>
</tr>
<tr>
<td>003072</td>
<td>130</td>
</tr>
<tr>
<td>003075</td>
<td>105</td>
</tr>
<tr>
<td>003100</td>
<td>125</td>
</tr>
<tr>
<td>003103</td>
<td>124</td>
</tr>
<tr>
<td>003106</td>
<td>120</td>
</tr>
<tr>
<td>003111</td>
<td>040</td>
</tr>
<tr>
<td>003114</td>
<td>122</td>
</tr>
<tr>
<td>003117</td>
<td>124</td>
</tr>
<tr>
<td>003122</td>
<td>107</td>
</tr>
<tr>
<td>003125</td>
<td>105</td>
</tr>
<tr>
<td>003130</td>
<td>111</td>
</tr>
<tr>
<td>003133</td>
<td>000</td>
</tr>
<tr>
<td>003134</td>
<td>045</td>
</tr>
<tr>
<td>003137</td>
<td>101</td>
</tr>
<tr>
<td>003142</td>
<td>124</td>
</tr>
<tr>
<td>003145</td>
<td>122</td>
</tr>
<tr>
<td>003150</td>
<td>040</td>
</tr>
<tr>
<td>003153</td>
<td>122</td>
</tr>
<tr>
<td>003156</td>
<td>103</td>
</tr>
<tr>
<td>003161</td>
<td>124</td>
</tr>
<tr>
<td>003164</td>
<td>114</td>
</tr>
<tr>
<td>003167</td>
<td>122</td>
</tr>
<tr>
<td>003172</td>
<td>116</td>
</tr>
<tr>
<td>003175</td>
<td>111</td>
</tr>
<tr>
<td>003200</td>
<td>111</td>
</tr>
<tr>
<td>003203</td>
<td>124</td>
</tr>
<tr>
<td>003206</td>
<td>116</td>
</tr>
<tr>
<td>003211</td>
<td>122</td>
</tr>
<tr>
<td>003214</td>
<td>122</td>
</tr>
<tr>
<td>003216</td>
<td>045</td>
</tr>
<tr>
<td>003221</td>
<td>101</td>
</tr>
<tr>
<td>003224</td>
<td>124</td>
</tr>
<tr>
<td>003227</td>
<td>122</td>
</tr>
<tr>
<td>003232</td>
<td>040</td>
</tr>
<tr>
<td>003235</td>
<td>117</td>
</tr>
<tr>
<td>003240</td>
<td>103</td>
</tr>
<tr>
<td>003243</td>
<td>040</td>
</tr>
<tr>
<td>003246</td>
<td>117</td>
</tr>
<tr>
<td>003251</td>
<td>124</td>
</tr>
<tr>
<td>003254</td>
<td>116</td>
</tr>
<tr>
<td>003257</td>
<td>122</td>
</tr>
<tr>
<td>ZRQ81</td>
<td>RDPX DISK FORMATTER</td>
</tr>
<tr>
<td>--------</td>
<td>---------------------</td>
</tr>
<tr>
<td>003262</td>
<td>122 000</td>
</tr>
<tr>
<td>003264</td>
<td>045 116</td>
</tr>
<tr>
<td>003267</td>
<td>101 044 106</td>
</tr>
<tr>
<td>003272</td>
<td>124 114 105</td>
</tr>
<tr>
<td>003275</td>
<td>122 055</td>
</tr>
<tr>
<td>003300</td>
<td>103 017</td>
</tr>
<tr>
<td>003303</td>
<td>115 125</td>
</tr>
<tr>
<td>003306</td>
<td>116 103</td>
</tr>
<tr>
<td>003311</td>
<td>101 124 111</td>
</tr>
<tr>
<td>003314</td>
<td>117 040</td>
</tr>
<tr>
<td>003317</td>
<td>101 122 105</td>
</tr>
<tr>
<td>003322</td>
<td>124 040 111</td>
</tr>
<tr>
<td>003325</td>
<td>116 111 124</td>
</tr>
<tr>
<td>003330</td>
<td>040 105 122</td>
</tr>
<tr>
<td>003333</td>
<td>122 117 122</td>
</tr>
<tr>
<td>003336</td>
<td>000 000</td>
</tr>
<tr>
<td>003340</td>
<td>111 120 040</td>
</tr>
<tr>
<td>003343</td>
<td>122 105 107</td>
</tr>
<tr>
<td>003346</td>
<td>111 123 124</td>
</tr>
<tr>
<td>003351</td>
<td>105 122 040</td>
</tr>
<tr>
<td>003352</td>
<td>101 104 104</td>
</tr>
<tr>
<td>003357</td>
<td>122 105 123</td>
</tr>
<tr>
<td>003362</td>
<td>123 000</td>
</tr>
<tr>
<td>003364</td>
<td>111 116 124</td>
</tr>
<tr>
<td>003367</td>
<td>105 122 122</td>
</tr>
<tr>
<td>003372</td>
<td>125 120 124</td>
</tr>
<tr>
<td>003375</td>
<td>040 126 105</td>
</tr>
<tr>
<td>003400</td>
<td>105 124 117</td>
</tr>
<tr>
<td>003403</td>
<td>122 040 101</td>
</tr>
<tr>
<td>003406</td>
<td>104 104 122</td>
</tr>
<tr>
<td>003411</td>
<td>105 123 123</td>
</tr>
<tr>
<td>003414</td>
<td>000 000</td>
</tr>
<tr>
<td>003416</td>
<td>102 125 123</td>
</tr>
<tr>
<td>003421</td>
<td>040 122 105</td>
</tr>
<tr>
<td>003424</td>
<td>121 125 105</td>
</tr>
<tr>
<td>003427</td>
<td>123 040 124</td>
</tr>
<tr>
<td>003432</td>
<td>114 105 126</td>
</tr>
<tr>
<td>003435</td>
<td>105 114 000</td>
</tr>
<tr>
<td>003440</td>
<td>123 117 106</td>
</tr>
<tr>
<td>003443</td>
<td>124 127 101</td>
</tr>
<tr>
<td>003446</td>
<td>122 105 040</td>
</tr>
<tr>
<td>003451</td>
<td>121 125 105</td>
</tr>
<tr>
<td>003454</td>
<td>123 124 111</td>
</tr>
<tr>
<td>003457</td>
<td>117 116 123</td>
</tr>
<tr>
<td>003462</td>
<td>040 117 116</td>
</tr>
<tr>
<td>003465</td>
<td>114 131 040</td>
</tr>
<tr>
<td>003470</td>
<td>101 120 120</td>
</tr>
<tr>
<td>003473</td>
<td>114 131 040</td>
</tr>
<tr>
<td>003476</td>
<td>125 116 104</td>
</tr>
<tr>
<td>003501</td>
<td>105 122 040</td>
</tr>
<tr>
<td>003504</td>
<td>101 120 124</td>
</tr>
<tr>
<td>003507</td>
<td>056 040 105</td>
</tr>
<tr>
<td>003512</td>
<td>115 124 105</td>
</tr>
<tr>
<td>ZROB1</td>
<td>ADRX DISK FORMATTER</td>
</tr>
<tr>
<td>-------</td>
<td>---------------------</td>
</tr>
<tr>
<td>003746</td>
<td>105</td>
</tr>
<tr>
<td>003751</td>
<td>106</td>
</tr>
<tr>
<td>003754</td>
<td>104</td>
</tr>
<tr>
<td>003757</td>
<td>105</td>
</tr>
<tr>
<td>003762</td>
<td>125</td>
</tr>
<tr>
<td>003765</td>
<td>116</td>
</tr>
<tr>
<td>003770</td>
<td>101</td>
</tr>
<tr>
<td>003773</td>
<td>125</td>
</tr>
<tr>
<td>003776</td>
<td>040</td>
</tr>
<tr>
<td>004001</td>
<td>122</td>
</tr>
<tr>
<td>004004</td>
<td>124</td>
</tr>
<tr>
<td>004007</td>
<td>116</td>
</tr>
<tr>
<td>074012</td>
<td>045</td>
</tr>
<tr>
<td>074015</td>
<td>101</td>
</tr>
<tr>
<td>074020</td>
<td>103</td>
</tr>
<tr>
<td>074025</td>
<td>122</td>
</tr>
<tr>
<td>074026</td>
<td>102</td>
</tr>
<tr>
<td>074031</td>
<td>040</td>
</tr>
<tr>
<td>074034</td>
<td>117</td>
</tr>
<tr>
<td>074037</td>
<td>040</td>
</tr>
<tr>
<td>074042</td>
<td>106</td>
</tr>
<tr>
<td>074045</td>
<td>115</td>
</tr>
<tr>
<td>074050</td>
<td>111</td>
</tr>
<tr>
<td>074053</td>
<td>040</td>
</tr>
<tr>
<td>074056</td>
<td>040</td>
</tr>
<tr>
<td>074061</td>
<td>101</td>
</tr>
<tr>
<td>074064</td>
<td>105</td>
</tr>
<tr>
<td>074067</td>
<td>101</td>
</tr>
<tr>
<td>074072</td>
<td>045</td>
</tr>
<tr>
<td>074077</td>
<td>101</td>
</tr>
<tr>
<td>074082</td>
<td>101</td>
</tr>
<tr>
<td>074105</td>
<td>101</td>
</tr>
<tr>
<td>074110</td>
<td>106</td>
</tr>
<tr>
<td>074113</td>
<td>114</td>
</tr>
<tr>
<td>074116</td>
<td>103</td>
</tr>
<tr>
<td>074121</td>
<td>103</td>
</tr>
<tr>
<td>074124</td>
<td>123</td>
</tr>
<tr>
<td>074127</td>
<td>116</td>
</tr>
<tr>
<td>074132</td>
<td>106</td>
</tr>
<tr>
<td>074135</td>
<td>000</td>
</tr>
<tr>
<td>074136</td>
<td>045</td>
</tr>
<tr>
<td>074141</td>
<td>101</td>
</tr>
<tr>
<td>074144</td>
<td>111</td>
</tr>
<tr>
<td>074147</td>
<td>111</td>
</tr>
<tr>
<td>074152</td>
<td>116</td>
</tr>
<tr>
<td>074155</td>
<td>040</td>
</tr>
<tr>
<td>074160</td>
<td>116</td>
</tr>
<tr>
<td>074163</td>
<td>105</td>
</tr>
<tr>
<td>074166</td>
<td>105</td>
</tr>
<tr>
<td>074171</td>
<td>117</td>
</tr>
<tr>
<td>074174</td>
<td>105</td>
</tr>
<tr>
<td>074177</td>
<td>040</td>
</tr>
<tr>
<td>Command</td>
<td>Address</td>
</tr>
<tr>
<td>-----------</td>
<td>---------</td>
</tr>
<tr>
<td>P.ABY</td>
<td>004220</td>
</tr>
<tr>
<td>P.ACB</td>
<td>004224</td>
</tr>
<tr>
<td>P.ACD</td>
<td>004226</td>
</tr>
<tr>
<td>P.ACE</td>
<td>004230</td>
</tr>
<tr>
<td>P.ACF</td>
<td>004232</td>
</tr>
<tr>
<td>P.ACH</td>
<td>004236</td>
</tr>
<tr>
<td>P.ACI</td>
<td>004302</td>
</tr>
<tr>
<td>P.ACJ</td>
<td>004351</td>
</tr>
<tr>
<td>P.AK</td>
<td>004355</td>
</tr>
<tr>
<td>P.AK</td>
<td>004363</td>
</tr>
<tr>
<td>P.AK</td>
<td>004371</td>
</tr>
<tr>
<td>P.AK</td>
<td>004377</td>
</tr>
<tr>
<td>P.AK</td>
<td>004405</td>
</tr>
<tr>
<td>P.AK</td>
<td>004413</td>
</tr>
<tr>
<td>P.AK</td>
<td>004420</td>
</tr>
</tbody>
</table>

**Notes:**
- ASCII: Each line represents a memory address with associated values.
- The values are likely byte addresses or memory locations in a program.
- The context suggests this is a part of a memory dump or a listing from a computer program.
<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>004426</td>
<td>124</td>
<td>114 105</td>
</tr>
<tr>
<td>004431</td>
<td>122</td>
<td>122 055</td>
</tr>
<tr>
<td>004434</td>
<td>040</td>
<td>125 116</td>
</tr>
<tr>
<td>004437</td>
<td>113</td>
<td>116 117</td>
</tr>
<tr>
<td>004442</td>
<td>127</td>
<td>116 040</td>
</tr>
<tr>
<td>004448</td>
<td>122</td>
<td>105 124</td>
</tr>
<tr>
<td>004450</td>
<td>125</td>
<td>122 116</td>
</tr>
<tr>
<td>004453</td>
<td>123</td>
<td>124 124</td>
</tr>
<tr>
<td>004456</td>
<td>101</td>
<td>124 125</td>
</tr>
<tr>
<td>004461</td>
<td>123</td>
<td>040 103</td>
</tr>
<tr>
<td>004464</td>
<td>117</td>
<td>104 105</td>
</tr>
<tr>
<td>004467</td>
<td>000</td>
<td></td>
</tr>
<tr>
<td>004470</td>
<td>045</td>
<td>116 045</td>
</tr>
<tr>
<td>004472</td>
<td>101</td>
<td>044 106</td>
</tr>
<tr>
<td>004474</td>
<td>124</td>
<td>114 105</td>
</tr>
<tr>
<td>004481</td>
<td>122</td>
<td>122 055</td>
</tr>
<tr>
<td>004483</td>
<td>040</td>
<td>120 117</td>
</tr>
<tr>
<td>004486</td>
<td>122</td>
<td>124 057</td>
</tr>
<tr>
<td>004490</td>
<td>110</td>
<td>117 123</td>
</tr>
<tr>
<td>004493</td>
<td>124</td>
<td>124 123</td>
</tr>
<tr>
<td>004495</td>
<td>131</td>
<td>116 103</td>
</tr>
<tr>
<td>004497</td>
<td>040</td>
<td>105 122</td>
</tr>
<tr>
<td>004500</td>
<td>122</td>
<td>117 122</td>
</tr>
<tr>
<td>004503</td>
<td>000</td>
<td></td>
</tr>
<tr>
<td>004504</td>
<td>045</td>
<td>116 045</td>
</tr>
<tr>
<td>004507</td>
<td>101</td>
<td>044 106</td>
</tr>
<tr>
<td>004510</td>
<td>124</td>
<td>114 105</td>
</tr>
<tr>
<td>004513</td>
<td>122</td>
<td>122 055</td>
</tr>
<tr>
<td>004516</td>
<td>040</td>
<td>115 105</td>
</tr>
<tr>
<td>004519</td>
<td>123</td>
<td>123 101</td>
</tr>
<tr>
<td>004522</td>
<td>107</td>
<td>105 116</td>
</tr>
<tr>
<td>004524</td>
<td>124</td>
<td>124 110</td>
</tr>
<tr>
<td>004526</td>
<td>040</td>
<td>105 122</td>
</tr>
<tr>
<td>004528</td>
<td>122</td>
<td>117 122</td>
</tr>
<tr>
<td>004530</td>
<td>000</td>
<td></td>
</tr>
</tbody>
</table>

**P.ACL:**
- ASCII /TLE/

**P.ACM:**
- ASCII /MM/

**P.ACN:**
- ASCII /MM/

**P.ACO:**
- ASCII /MM/
<table>
<thead>
<tr>
<th>ZR081 REV C PATCH O</th>
<th>RDRX DISK FORMATTER GLOBAL TEXT SECTION</th>
<th>6 Mar 1984 14:16:44</th>
<th>VAX-11 B1:ie 16 V4.0.579</th>
</tr>
</thead>
<tbody>
<tr>
<td>004655</td>
<td>113 116 117</td>
<td>ASCII /KNO/</td>
<td>6 Mar 1984 14:16:36</td>
</tr>
<tr>
<td>004663</td>
<td>105 116 104</td>
<td>ASCII /END/</td>
<td>Page 44</td>
</tr>
<tr>
<td>004666</td>
<td>103 117 104</td>
<td>ASCII /COO/</td>
<td>(17)</td>
</tr>
<tr>
<td>004671</td>
<td>105 040 102</td>
<td>ASCII /E/R/</td>
<td></td>
</tr>
<tr>
<td>004674</td>
<td>105 103 105</td>
<td>ASCII /EC/E/</td>
<td></td>
</tr>
<tr>
<td>004677</td>
<td>111 126 105</td>
<td>ASCII /IVE/</td>
<td></td>
</tr>
<tr>
<td>004702</td>
<td>104 000</td>
<td>ASCII /D/&lt;00&gt;</td>
<td></td>
</tr>
<tr>
<td>004704</td>
<td>045 116 045</td>
<td>P.ACP: ASCII /&amp;MM/</td>
<td></td>
</tr>
<tr>
<td>004707</td>
<td>101 044 106</td>
<td>ASCII /AIF/</td>
<td></td>
</tr>
<tr>
<td>004712</td>
<td>124 114 105</td>
<td>ASCII /TEL/</td>
<td></td>
</tr>
<tr>
<td>004715</td>
<td>122 122 055</td>
<td>ASCII /RR/</td>
<td></td>
</tr>
<tr>
<td>004720</td>
<td>120 101 104</td>
<td>ASCII /AD/</td>
<td></td>
</tr>
<tr>
<td>004723</td>
<td>101 120 124</td>
<td>ASCII /APT/</td>
<td></td>
</tr>
<tr>
<td>004726</td>
<td>117 122 040</td>
<td>ASCII /OR/</td>
<td></td>
</tr>
<tr>
<td>004731</td>
<td>120 125 122</td>
<td>ASCII /PUR/</td>
<td></td>
</tr>
<tr>
<td>004734</td>
<td>107 105 040</td>
<td>ASCII /GE/</td>
<td></td>
</tr>
<tr>
<td>004737</td>
<td>105 122 122</td>
<td>ASCII /ERR/</td>
<td></td>
</tr>
<tr>
<td>004742</td>
<td>117 122 000</td>
<td>ASCII /OR/&lt;00&gt;</td>
<td></td>
</tr>
<tr>
<td>004745</td>
<td>000</td>
<td>ASCII /&lt;00&gt;</td>
<td></td>
</tr>
<tr>
<td>004746</td>
<td>045 116 045</td>
<td>P.ACQ: ASCII /&amp;MM/</td>
<td></td>
</tr>
<tr>
<td>004751</td>
<td>101 044 106</td>
<td>ASCII /AIF/</td>
<td></td>
</tr>
<tr>
<td>004754</td>
<td>124 114 105</td>
<td>ASCII /TEL/</td>
<td></td>
</tr>
<tr>
<td>004757</td>
<td>122 122 055</td>
<td>ASCII /RR/</td>
<td></td>
</tr>
<tr>
<td>004762</td>
<td>040 125 116</td>
<td>ASCII /UN/</td>
<td></td>
</tr>
<tr>
<td>004765</td>
<td>113 116 117</td>
<td>ASCII /KNO/</td>
<td></td>
</tr>
<tr>
<td>004770</td>
<td>127 116 040</td>
<td>ASCII /MN/</td>
<td></td>
</tr>
<tr>
<td>004773</td>
<td>111 116 124</td>
<td>ASCII /INT/</td>
<td></td>
</tr>
<tr>
<td>004776</td>
<td>105 122 122</td>
<td>ASCII /ERR/</td>
<td></td>
</tr>
<tr>
<td>005001</td>
<td>125 120 124</td>
<td>ASCII /JPT/</td>
<td></td>
</tr>
<tr>
<td>005004</td>
<td>000 000</td>
<td>ASCII /&lt;00&gt;&lt;00&gt;</td>
<td></td>
</tr>
<tr>
<td>005006</td>
<td>045 116 045</td>
<td>P.ACR: ASCII /&amp;MM/</td>
<td></td>
</tr>
<tr>
<td>005011</td>
<td>101 044 106</td>
<td>ASCII /AIF/</td>
<td></td>
</tr>
<tr>
<td>005014</td>
<td>124 114 105</td>
<td>ASCII /TEL/</td>
<td></td>
</tr>
<tr>
<td>005017</td>
<td>122 122 055</td>
<td>ASCII /RR/</td>
<td></td>
</tr>
<tr>
<td>005022</td>
<td>040 111 116</td>
<td>ASCII /IN/</td>
<td></td>
</tr>
<tr>
<td>005025</td>
<td>111 124 040</td>
<td>ASCII /IT/</td>
<td></td>
</tr>
<tr>
<td>005030</td>
<td>123 121</td>
<td>ASCII /SEQ/</td>
<td></td>
</tr>
<tr>
<td>005033</td>
<td>040 123 124</td>
<td>ASCII /ST/</td>
<td></td>
</tr>
<tr>
<td>005036</td>
<td>105 120 040</td>
<td>ASCII /EP/</td>
<td></td>
</tr>
<tr>
<td>005041</td>
<td>124 111 115</td>
<td>ASCII /TIM/</td>
<td></td>
</tr>
<tr>
<td>005044</td>
<td>105 104 040</td>
<td>ASCII /ED/</td>
<td></td>
</tr>
<tr>
<td>005047</td>
<td>117 125 124</td>
<td>ASCII /OUT/</td>
<td></td>
</tr>
<tr>
<td>005052</td>
<td>000 000</td>
<td>ASCII /&lt;00&gt;&lt;00&gt;</td>
<td></td>
</tr>
<tr>
<td>005054</td>
<td>045 116 045</td>
<td>P.ACS: ASCII /&amp;MM/</td>
<td></td>
</tr>
<tr>
<td>005057</td>
<td>101 044 106</td>
<td>ASCII /AIF/</td>
<td></td>
</tr>
<tr>
<td>005062</td>
<td>124 114 105</td>
<td>ASCII /TEL/</td>
<td></td>
</tr>
<tr>
<td>005065</td>
<td>122 122 055</td>
<td>ASCII /RR/</td>
<td></td>
</tr>
<tr>
<td>005070</td>
<td>040 111 116</td>
<td>ASCII /IN/</td>
<td></td>
</tr>
<tr>
<td>005073</td>
<td>111 124 040</td>
<td>ASCII /IT/</td>
<td></td>
</tr>
<tr>
<td>005076</td>
<td>123 105 121</td>
<td>ASCII /SEQ/</td>
<td></td>
</tr>
<tr>
<td>005101</td>
<td>040 103 117</td>
<td>ASCII /CO/</td>
<td></td>
</tr>
<tr>
<td>005104</td>
<td>115 120 101</td>
<td>ASCII /MPA/</td>
<td></td>
</tr>
<tr>
<td>Start Address</td>
<td>Data</td>
<td>Field Name</td>
<td>Value</td>
</tr>
<tr>
<td>--------------</td>
<td>------</td>
<td>------------</td>
<td>-------</td>
</tr>
<tr>
<td>005107</td>
<td>122</td>
<td></td>
<td>105</td>
</tr>
<tr>
<td>005112</td>
<td>105</td>
<td></td>
<td>122</td>
</tr>
<tr>
<td>005115</td>
<td>117</td>
<td></td>
<td>122</td>
</tr>
<tr>
<td>005120</td>
<td>045</td>
<td></td>
<td>116</td>
</tr>
<tr>
<td>005123</td>
<td>101</td>
<td></td>
<td>044</td>
</tr>
<tr>
<td>005126</td>
<td>124</td>
<td></td>
<td>114</td>
</tr>
<tr>
<td>005131</td>
<td>122</td>
<td></td>
<td>122</td>
</tr>
<tr>
<td>005134</td>
<td>040</td>
<td></td>
<td>125</td>
</tr>
<tr>
<td>005137</td>
<td>105</td>
<td></td>
<td>130</td>
</tr>
<tr>
<td>005142</td>
<td>105</td>
<td></td>
<td>103</td>
</tr>
<tr>
<td>005145</td>
<td>105</td>
<td></td>
<td>104</td>
</tr>
<tr>
<td>005150</td>
<td>101</td>
<td></td>
<td>124</td>
</tr>
<tr>
<td>005153</td>
<td>105</td>
<td></td>
<td>116</td>
</tr>
<tr>
<td>005156</td>
<td>111</td>
<td></td>
<td>117</td>
</tr>
<tr>
<td>005161</td>
<td>040</td>
<td></td>
<td>105</td>
</tr>
<tr>
<td>005164</td>
<td>104</td>
<td></td>
<td>040</td>
</tr>
<tr>
<td>005167</td>
<td>105</td>
<td></td>
<td>123</td>
</tr>
<tr>
<td>005172</td>
<td>101</td>
<td></td>
<td>107</td>
</tr>
<tr>
<td>005175</td>
<td>040</td>
<td></td>
<td>122</td>
</tr>
<tr>
<td>005200</td>
<td>103</td>
<td></td>
<td>105</td>
</tr>
<tr>
<td>005203</td>
<td>126</td>
<td></td>
<td>104</td>
</tr>
<tr>
<td>005206</td>
<td>000</td>
<td></td>
<td>000</td>
</tr>
<tr>
<td>005210</td>
<td>045</td>
<td></td>
<td>116</td>
</tr>
<tr>
<td>005213</td>
<td>101</td>
<td></td>
<td>044</td>
</tr>
<tr>
<td>005216</td>
<td>124</td>
<td></td>
<td>114</td>
</tr>
<tr>
<td>005221</td>
<td>122</td>
<td></td>
<td>122</td>
</tr>
<tr>
<td>005224</td>
<td>040</td>
<td></td>
<td>125</td>
</tr>
<tr>
<td>005227</td>
<td>105</td>
<td></td>
<td>130</td>
</tr>
<tr>
<td>005232</td>
<td>105</td>
<td></td>
<td>103</td>
</tr>
<tr>
<td>005235</td>
<td>105</td>
<td></td>
<td>104</td>
</tr>
<tr>
<td>005240</td>
<td>103</td>
<td></td>
<td>117</td>
</tr>
<tr>
<td>005243</td>
<td>115</td>
<td></td>
<td>101</td>
</tr>
<tr>
<td>005246</td>
<td>104</td>
<td></td>
<td>040</td>
</tr>
<tr>
<td>005251</td>
<td>126</td>
<td></td>
<td>103</td>
</tr>
<tr>
<td>005254</td>
<td>104</td>
<td></td>
<td>105</td>
</tr>
<tr>
<td>005257</td>
<td>111</td>
<td></td>
<td>116</td>
</tr>
<tr>
<td>005262</td>
<td>105</td>
<td></td>
<td>116</td>
</tr>
<tr>
<td>005265</td>
<td>040</td>
<td></td>
<td>115</td>
</tr>
<tr>
<td>005270</td>
<td>123</td>
<td></td>
<td>123</td>
</tr>
<tr>
<td>005273</td>
<td>107</td>
<td></td>
<td>105</td>
</tr>
<tr>
<td>005276</td>
<td>122</td>
<td></td>
<td>105</td>
</tr>
<tr>
<td>005301</td>
<td>105</td>
<td></td>
<td>111</td>
</tr>
<tr>
<td>005304</td>
<td>105</td>
<td></td>
<td>104</td>
</tr>
<tr>
<td>005307</td>
<td>000</td>
<td></td>
<td>045</td>
</tr>
<tr>
<td>005310</td>
<td>045</td>
<td></td>
<td>116</td>
</tr>
<tr>
<td>005313</td>
<td>101</td>
<td></td>
<td>044</td>
</tr>
<tr>
<td>005316</td>
<td>124</td>
<td></td>
<td>114</td>
</tr>
<tr>
<td>005321</td>
<td>122</td>
<td></td>
<td>122</td>
</tr>
<tr>
<td>005324</td>
<td>040</td>
<td></td>
<td>125</td>
</tr>
<tr>
<td>005327</td>
<td>105</td>
<td></td>
<td>130</td>
</tr>
<tr>
<td>005332</td>
<td>105</td>
<td></td>
<td>103</td>
</tr>
<tr>
<td>005335</td>
<td>105</td>
<td></td>
<td>104</td>
</tr>
<tr>
<td>005340</td>
<td>123</td>
<td></td>
<td>105</td>
</tr>
</tbody>
</table>

**P.ACT:**
- 005107: ASCII /RE/
- 005112: ASCII /ERR/
- 005115: ASCII /OR/00
- 005120: ASCII /NW/
- 005123: ASCII /AIF/
- 005126: ASCII /TEL/
- 005131: ASCII /RR/
- 005134: ASCII /UN/
- 005137: ASCII /EXP/
- 005142: ASCII /ECT/
- 005145: ASCII /ED/
- 005150: ASCII /ATT/
- 005153: ASCII /ENT/
- 005156: ASCII /ION/
- 005161: ASCII /EN/
- 005164: ASCII /D N/
- 005167: ASCII /ESS/
- 005172: ASCII /AGE/
- 005175: ASCII /RE/
- 005200: ASCII /CE1/
- 005203: ASCII /VED/
- 005206: ASCII /OE/OO

**P.ACU:**
- 005210: ASCII /NW/
- 005213: ASCII /AIF/
- 005216: ASCII /TEL/
- 005221: ASCII /RR/
- 005224: ASCII /UN/
- 005227: ASCII /EXP/
- 005232: ASCII /ECT/
- 005235: ASCII /ED/
- 005240: ASCII /CON/
- 005243: ASCII /MAN/
- 005246: ASCII /D O/
- 005251: ASCII /PCD/
- 005254: ASCII /DE/
- 005257: ASCII /EN/
- 005262: ASCII /END/
- 005265: ASCII /HE/
- 005270: ASCII /SSA/
- 005273: ASCII /GE/
- 005276: ASCII /REC/
- 005301: ASCII /EIV/
- 005304: ASCII /ED/OO

**P.ACV:**
- 005310: ASCII /NW/
- 005313: ASCII /AIF/
- 005316: ASCII /TEL/
- 005321: ASCII /RR/
- 005324: ASCII /UN/
- 005327: ASCII /EXP/
- 005332: ASCII /ECT/
- 005335: ASCII /ED/
- 005340: ASCII /SER/
<table>
<thead>
<tr>
<th>ZRQ81 REV C PATCH O</th>
<th>RDX DISK FORMATTER GLOBAL TEXT SECTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII /IOU/</td>
<td>111 117 125</td>
</tr>
<tr>
<td>ASCII /S E/</td>
<td>123 040 105</td>
</tr>
<tr>
<td>ASCII /XCE/</td>
<td>130 103 105</td>
</tr>
<tr>
<td>ASCII /PT/</td>
<td>120 124 111</td>
</tr>
<tr>
<td>ASCII /ON/</td>
<td>117 116 040</td>
</tr>
<tr>
<td>ASCII /END/</td>
<td>105 116 104</td>
</tr>
<tr>
<td>ASCII /ME/</td>
<td>040 113 105</td>
</tr>
<tr>
<td>ASCII /SSA/</td>
<td>123 123 101</td>
</tr>
<tr>
<td>ASCII /GE/</td>
<td>058 105 040</td>
</tr>
<tr>
<td>ASCII /REC/</td>
<td>122 105 103</td>
</tr>
<tr>
<td>ASCII /EIV/</td>
<td>105 116 126</td>
</tr>
<tr>
<td>ASCII /ED/&lt;00&gt;</td>
<td>105 104 000</td>
</tr>
<tr>
<td>ASCII 00/</td>
<td>05404 000</td>
</tr>
<tr>
<td>P.ACW: ASCII /XNN/</td>
<td>045 116 045</td>
</tr>
<tr>
<td>P.BIZ: ASCII /XNN/</td>
<td>101 044 106</td>
</tr>
<tr>
<td>P.BIZ: ASCII /XNF/</td>
<td>116 114 050</td>
</tr>
<tr>
<td>P.BIZ: ASCII /XIF/</td>
<td>120 114 055</td>
</tr>
<tr>
<td>P.BIZ: ASCII /XIN/</td>
<td>104 101 114</td>
</tr>
<tr>
<td>P.BIZ: ASCII /XID/</td>
<td>101 040 114</td>
</tr>
<tr>
<td>P.BIZ: ASCII /XIN/</td>
<td>105 105 017</td>
</tr>
<tr>
<td>P.BIZ: ASCII /XIN/</td>
<td>045 116 045</td>
</tr>
<tr>
<td>P.BIZ: ASCII /XIN/</td>
<td>045 116 045</td>
</tr>
<tr>
<td>P.BIZ: ASCII /XIN/</td>
<td>045 116 045</td>
</tr>
<tr>
<td>P.BIZ: ASCII /XIN/</td>
<td>045 116 045</td>
</tr>
<tr>
<td>P.BIZ: ASCII /XIN/</td>
<td>045 116 045</td>
</tr>
</tbody>
</table>
000130'    L:ERRTBL**  ERRTIP
000156'    L:SW**    L:SWLEN+2
000162'    L:MW**    L:MWLEN+2
0000142'   L:DEPO**  L:REV+1
0000142'   DFPTBL**  L:MWLEN+2
0000156'   SFPTBL**  L:SWLEN+2
0000000'   FMT1**    P:AAB
0000004'   FMT2**    P:AAC
0000046'   FMT3**    P:AAC
0000142'   FMT4**    P:AAB
0000220'   CPLF**    P:AAB
0000224'   UNITL.MSG**    P:AAB
0000260'   EXISTL.MSG**    P:AAB
0000324'   DONN. MSG**    P:AAB
0000350'   INACC.MSG**    P:AAB
0000430'   DFLT.MSG**    P:AAB
0000500'   SERIAL.MSG**    P:AAB
0000540'   DATE.MSG**    P:AAB
0000576'   DATMSG**    P:AAB
0000650'   DEF.DATE**    P:AAB
0000642'   DEF.SERIAL**    P:AAB
000010'    RINGBASE**    CM AREA+10
0000740'   MSGADR**    REC BUF+2
000546'    PFE_STRUCT**   P:AAP
002622'    PMR.MSG**    P:AAB
002704'    ABO.MSG**    P:ABN
002764'    TO.MANY.UNIT**    P:ABN
003042'    GOOD.NUM.UNIT**    P:ABN
003134'    BOOT.FAILURE**    P:ABQ
003216'    PROTO.VIOLATION**    P:ABN
003264'    PORT.INIT.ERR**    P:ABN
003340'    HW.Q1.IPT**    P:ABQ
003364'    HW.Q2.VECTOR**    P:ABU
003416'    HW.Q3.BR**    P:ABV
003440'    SW.Q1.UNIT**    P:ABW
003534'    SW.Q2.MODE**    P:ABX
004220'    FMT.ERR**    P:ABY
006110'    EMSG.STRUCT**    P:ACG

PSECT SUMMARY

<table>
<thead>
<tr>
<th>Psect Name</th>
<th>Words</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CODE$</td>
<td>61</td>
<td>RO, I, LCI, REL, CON</td>
</tr>
<tr>
<td>$GLOB$</td>
<td>429</td>
<td>RO, D, GBL, REL, CON</td>
</tr>
</tbody>
</table>
Library Statistics

<table>
<thead>
<tr>
<th>File</th>
<th>Total</th>
<th>Loaded</th>
<th>Percent</th>
<th>Pages</th>
<th>Processing Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISK$USER2:[YOUNG.FMT]ZRQB80.L16:4</td>
<td>358</td>
<td>223</td>
<td>62</td>
<td>18</td>
<td>00:00.1</td>
</tr>
</tbody>
</table>

COMMAND QUALIFIERS


| Size: 0 code + 2084 data words |
| Run time: 00:20:9 |
| Elapsed Time: 00:49.1 |
| Lines/CPU Min: 6954 |
| Lexemes/CPU-Min: 38982 |
| Memory Used: 207 pages |
| Compilation Complete |
PROCEDURE RDRX DISK FORMATTER

IDENT 'RDRX DISK FORMATTER'
ADDRESSING MODE (ABSOLUTE)
ENVIRONMENT (NOEIS)

BEGIN

! CONTROLLER FUNCTIONS

library 'ZRQ800.L16';
require 'BLSMAC.REQ';

! Define RDRX Library module
! Define Bliss Macro Library

#define 'MODULE DECLARATIONS'
structure

: ! Structure declarations used within this module.

: !

: !

: ! ADRX register accessing structure. This
: structure allows ADRX register accessing
: to be transportable between the PDP-11 and
: VAX Diagnostic Supervisors.

: !

: ! This also defines an access algorithm for
: VAX to allow field reference to MBA address
: space without generating machine checks.

: !

: ADRX [Q, P, S, E] =

: begin

: local

: RC15_REG;

: RC15_REG = .((ADRX * &upval+0)<0, &ppval, 0>;

: RC15_REG

: end

: &P, S, E;

: 1577 1
The spec named "code or scode" is redefined here to be called "scode". This is done to force the linker to place the programs header information starting at absolute address 2000. Then for consistency "scode" is used in place of "code or fcode" across all modules.

```
1578 1 /* The spec named "code or scode" is redefined here 
1579 1 to be called "scode". This is done to force the linker 
1580 1 to place the programs header information starting 
1581 1 at absolute address 2000. Then for consistency "scode" 
1582 1 is used in place of "code or fcode" across all modules. 
1583 1 */
1584 1
1585 1 /*
1586 1   code = scode;
1587 1 */
1588 1
1589 1 /* External Routine declared outside this module. */
1590 1
1591 1 /* external routine */
1592 1 /* DECODE : NOVALUE, */
1593 1 /* LOAD_FILE; */
```
```
1594 1
1595 1 ! External Declaration of datums
1596 1 ! declared outside of this module.
1597 1
1598 1
1599 1 external
1600 1 ! Hardware question ascii string messages
1601 1
1602 1
1603 1  ! H/W question 1 for IP reg address
1604 1  ! H/W question 2 for interrupt vector address
1605 1  ! H/W question 3 for bus reg level
1606 1
1607 1  ! software question 1 for unit
1608 1  ! software question 2 for unit
1609 1
1610 1 ! Formatting print string
1611 1
1612 1  ! Notifies unit being formatted
1613 1  ! Notifies format abort
1614 1
1615 1 ! Init code error and informational messages
1616 1
1617 1 PWR_MSG,
1618 1 ABO_MSG,
1619 1 TC_MANY_UNITS,
1620 1 GOOD_NUM_UNITS,
1621 1
1622 1 ! Miscellaneous external data declarations
1623 1
1624 1 LUN : byte,
1625 1 PTBL_PTR : ref vector [4, word],
1626 1
1627 1 ! Supervisor defined data declarations
1628 1
1629 1 LUNIT,
1630 1
1631 1 ! Hardware P_Table storage declarations
1632 1
1633 1 RDRX_ADDR : ref RDRX field (ISD_FIELD),
1634 1 VEC_ADDR : word,
1635 1 BR_LEVEL : word,
1636 1 UNIT_NO : word,
1637 1
1638 1 ! Formatter data structures
1639 1
1640 1 ISD_STRUCT : blockvector [4, 2, word] field (ISD_FIELD);
```
#bt1 'TYPE AND DESCRIPTION

Two lines of text will be printed to the operator (in addition to the
program name). The first will come from the "DESCRIPT" macro at start
up time and will identify the diagnostics. The second will come from
the "DEVTYPE" macro at hardware dialogue time and will identify the
device under test. The arguments of both macros are 72 character
strings enclosed in parentheses:

DESCRIP ("asc:RD51/52 DISK FORMATTER");
DEVYP ("asc:RDQX1 DISK DRIVE SUBSYSTEM");
The hardware parameter coding section contains macros that are used by the supervisor to build P tables. The macros are not executed as machine instructions but are interpreted by the supervisor as data structures. The macros allow the supervisor to establish communications with the operator.

BGNHARD:

GOPMA (HW_Q1.IP, "0", 0, "0:16000", "0:17777", YES, 1);  !Get RDRX Controller IP register

GOPMA (HW_Q2_VECTOR, "0", 0, "0:774", YES, 1);  !Get RDRX Interrupt Vector address

GOPMA (HW_Q3_BR, "0", 0, "0:17777", 0, 7, YES, 1);  !Get RDRX Bus Request Priority

ENDHARD;
**SOFTWARE PARAMETER CODING SECTION**

The software parameter coding section contains macros that are used by the supervisor to build P-Tables. The macros are not executed as machine instructions but are interpreted by the supervisor as data structures. The macros allow the supervisor to establish communications with the operator.

Software parameter coding is accomplished via DM micro code. This is to conform to DUP protocol standards.

- `GPRMD (SW_Q1_UNIT, '0', 0, '177777', 0, 3, YES, 0);`  
  !get unit number

- `GPRMD (SW_Q2_MODE, '0', 0, '177777', 1, 3, YES, 0);`  
  !Get mode

- `ENDSFT;`
The statistical report coding section contains the PRINTS macros that will be used to generate statistical reports. The `BGNRPT` and `ENDRPT` macros are used as beginning and ending directives for the coding contained in the section. However, an externally located DORPT call, or a print command from the operator, may be used to request the execution of the report coding.

BGNRPT:
return;

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

ENDRPT;

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

.LDESC:: ASCII /RD5/}
.LDESC:: ASCII /1<57>/5/
.LDESC:: ASCII /2 D/
.LDESC:: ASCII /ISK/
.LDESC:: ASCII /TTE/
.LDESC:: ASCII /R=<00><00>
.LDESC:: ASCII /ROD/
.LDESC:: ASCII /X1 /
.LDESC:: ASCII /DIS/
.LDESC:: ASCII /K D /
.LDESC:: ASCII /RIV/
.LDESC:: ASCII /E S /
.LDESC:: ASCII /UBS/
.LDESC:: ASCII /EM<00>
.LDESC:: ASCII <00>

.LHDRDLN::

.GP1::
.WORD <<LHNRDRD.LHRLDN/2> 1>
.WORD 31
.WORD MW.Q1.IP
.WORD 16000
Routine Size: 4 words,  
Routine Base: $CODE$ + 0154
Maximum stack depth per invocation: 2 words
The initialization code is executed at the beginning of every sub-pass and is primarily used for requesting P_Tables. Any other set-up type functions may also be performed in the init code.

The initialize code is executed under five conditions. There are supervisor event flags that are used to let the diagnostic know under which condition the execution is taking place. The event flags are read using the "READE" macro.

The conditions under which the init code is executed and the corresponding event flags are:

- START COMMAND
- CONTINUE COMMAND
- POWERDOWN/POWERUP
- NEW PASS
- EF.NEW

Example of event flag use:

if READE(START) then
   START_FLAG = 1;

First read the event flag EF_PWR to see if this init code is being performed due to a system power fail. If it is then report the incident to the operator and abort the DM machine and further execution of this program.

if READE(EF_PWR) then
   begin
   PRINTF (PWR_MSG);
   PRINTF (ABD_MSG);
   WRT_RDRX (RCIP, RC_ALL, ZEROS);
   DOCLN;
   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

See if the DRS commands START or RESTART were used to start this formatting session. If either of them were used then start the formatting session at logical unit zero.
if (READEF (EF_START)) or (READEF (EF_RESTART)) then
    begin
        ! Check the operator for trying to format
        ! more than the defined limit. If they
        ! are then report the error and die.
        if .LSUNIT gtstu LLLIMIT then
            ! Is the formatting limit exceeded
            begin
                PRINTF (TO_MANY_UNITS);
                PRINTF (GOOD_NUM_UNITS);
                DOLCN;
                end;
        ! Everything looks good so far so lets
        ! move on and get the hardware question
        ! responses and save them.
        LUN * -1;
        ! Start formatting at logical unit 0
        do begin
            LUN * LUN + 1;
            if .LUN gequ .LSUNIT then DOLCN;
        end
        until (GPHARD (.LUN, PTBL_PTR)) nequ ZERO;
        RDRX_ADDR = .PTBL_PTR [wrd0];
        ! Load up the controllers base address
        VEC_ADDR = .PTBL_PTR [wrd1];
        ! Load up the controllers vector address
        BR_LEVEL = .PTBL_PTR [wrd2];
        ! Load up the controllers bus request
        UNIT_NO = .PTBL_PTR [wrd3];
        ! Load up the unit number to format
        ! Before leaving the init code section we must
        ! First do some house keeping left behind from
        ! the ISD_STRUCTURE preset declaration. The adapter
        ! purge interrupt bit must be defined for the
        ! type machine this formatter is running under.
        if bliss (bliss16) then
            ! Define compiler
            ISD_STRUCTURE [BLK1, WRD1, S2W.PI] = ZERO
            ! No purging for PDP 11
            else
                ISD_STRUCTURE [BLK1, WRD1, S2W.PI] = ONE;
                ! Purging for VAX 11
            !
1812 ! Now load in the RDRX formatter DM code from
1813 the local boot device into the
1814 blank buffer allocated in azkel. Repor-
1815 t load error if an error code is returned.
1816
1817 ! The DM code will only be read in during start
1818 or restarts of the host code. Block zero of the
1819 ! the DM .sav file will thrown out.
1820
1821
1822 ! IF LOAD_FILE (AZFMTR, DM_FNAME, DM_SIZE) then DECODE ( );
1823
1824
1825 ! Now calculate the overlay sections starting address
1826 and store the result in global location 'ovsa' for
1827 future ref.
1828
1829 ! This takes the DM buffer starting adrs and adds to it
1830 ! the number of bytes in the (initial load + remote prog
1831 ! header size) resulting in the first adrs of the overlay
1832 ! section.
1833
1834
1835 ! OVSA = AZFMTR + (AZFMTR [WRDO]); !Calculate overlay start adrs
1836
1837 end
1838 else
1839 begin
1840
1841 if READEF (EF_CONTINUE) then
1842
1843 begin
1844
1845 ! End this Host code if this .lun is the
1846 ! the last logical unit to format.
1847
1848 if (.LUN + 1) geq .LUNIT then DOCLN;
1849
1850 PRINTF (FMT3, .LUN, .UNIT_NO);
1851 !Report this luns format was aborted
1852
1853 end;
1854
1855
1856 if READEF (EF_NEW) then GPHARD (0, 0); !New pass means all units formatted
1857 do
1858
1859 begin
1860 LUN = .LUN + 1;
1861
1862 if .LUN geq .LUNIT then DOCLN;
1863 end
1864 until (GPHARD ( .LUN, PTBL_PTR)) neq ZERO;
RDRX_DISK_FORMATTER

INITIALIZE SECTION

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

000000 005746 LIMIT: SBTTL LIMIT INITIALIZE SECTION
000002 012700 00034
000006 104447 000000
000010 103025 000000
000012 012746 000000
000016 012746 000001
000022 010600 000000
000024 104417 000000
000026 012716 000000
000032 012746 000001
000036 010600 000000
000040 104417 000000
000042 017766 000000 000006
000050 005000 000000
000052 005077 000000
000056 104444 000000
000060 062706 000006 1f:
000064 012700 000040 1f:
000070 104447 000000
000072 103404 000000
000074 012700 000037
000078 104444 000000
000082 103065 000000
000084 023727 000000 00.020 2f:
000086 104147 000000
000088 012746 000000
00008c 012746 000001
000092 010600 000000
000094 104417 000000
000096 012716 000000
000098 012746 000001
00009a 012700 000037
00009e 012700 000037
000100 104444 000000
000104 112737 000377 000000 3f:
00010a 105237 000000 4f:
000114 005000 000000
000116 013400 000000
000118 013400 000000
00011a 013400 000000
00011c 013400 000000
000120 013400 000000
000122 013400 000000
000124 013400 000000
000126 013400 000000
000128 013400 000000
00012a 013400 000000
00012c 013400 000000
00012e 013400 000000
000130 013400 000000
000132 013400 000000
000134 013400 000000
000136 013400 000000
000138 013400 000000
00013a 013400 000000
00013c 013400 000000
00013e 013400 000000
000140 013400 000000
000142 013400 000000
000144 013400 000000
000146 062706 000006
000148 112737 000377 000000
00014a 105237 000000
00014c 005000 000000
00014e 013400 000000
000150 013400 000000
000152 112737 000377 000000
000154 105237 000000
000156 005000 000000
000158 013400 000000
00015a 013400 000000
00015c 013400 000000
00015e 013400 000000
000160 013400 000000
000162 013400 000000
000164 013400 000000
000166 013400 000000
000168 013400 000000
00016a 013400 000000
00016c 013400 000000
00016e 013400 000000
000170 013400 000000
000172 002003 000000
000174 013400 000000
000176 103401 000000
<table>
<thead>
<tr>
<th>Line</th>
<th>Address</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000202</td>
<td>005000</td>
<td>0000GG</td>
<td>Slim</td>
</tr>
<tr>
<td>000204</td>
<td>013760</td>
<td>000000</td>
<td>BSB</td>
</tr>
<tr>
<td>000210</td>
<td>010442</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000212</td>
<td>010037</td>
<td>000000</td>
<td>MEV</td>
</tr>
<tr>
<td>000216</td>
<td>001760</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000220</td>
<td>011037</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000224</td>
<td>016037</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000232</td>
<td>016037</td>
<td>000004</td>
<td>MOV</td>
</tr>
<tr>
<td>000240</td>
<td>016037</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000246</td>
<td>142737</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000254</td>
<td>000474</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000256</td>
<td>012700</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000262</td>
<td>104447</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000264</td>
<td>103025</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000266</td>
<td>005000</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000270</td>
<td>013760</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000274</td>
<td>009200</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000276</td>
<td>020000</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000302</td>
<td>013401</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000304</td>
<td>104444</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000306</td>
<td>013746</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000312</td>
<td>005046</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000314</td>
<td>113716</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000320</td>
<td>012746</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000324</td>
<td>012746</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000330</td>
<td>010600</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000332</td>
<td>104417</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000334</td>
<td>062706</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000340</td>
<td>012700</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000344</td>
<td>104447</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000346</td>
<td>103004</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000350</td>
<td>005000</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000352</td>
<td>104442</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000354</td>
<td>010037</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000360</td>
<td>015237</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000364</td>
<td>005000</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000366</td>
<td>113700</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000372</td>
<td>020007</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000376</td>
<td>103401</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000400</td>
<td>104444</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000402</td>
<td>050000</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000404</td>
<td>153700</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000410</td>
<td>104444</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000412</td>
<td>010037</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000416</td>
<td>001760</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000420</td>
<td>011037</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000424</td>
<td>016037</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000432</td>
<td>016037</td>
<td>000004</td>
<td>MOV</td>
</tr>
<tr>
<td>000440</td>
<td>016037</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000446</td>
<td>005726</td>
<td>000000</td>
<td>MOV</td>
</tr>
<tr>
<td>000450</td>
<td>000207</td>
<td>000000</td>
<td>MOV</td>
</tr>
</tbody>
</table>

Routine Size: 149 words
Routine Base: $CODE+0164
Maximum stack depth per invocation: 7 words

Routine Size: 4 words, Routine Base: $CODE$ + 0636
Maximum stack depth per invocation: 2 words
This code is executed immediately after the initialize code if
the "ADR" flag was set. The unit(s) under test are checked to
see if they will respond. Those that don't are immediately
dropped from testing.

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

Routine Size: 4 words.  Routine Base: $CODE1$ + 0650
Maximum stack depth per invocation: 2 words
Cleanup coding is assembled with the diagnostic program, utilizing initiating (BGNCLN) and ending (ENDCLN) directives. The coding can be used by either the diagnostic program or the supervisor to affect the return of a test device to a static state.

The clean-up code is invoked in three different ways:

A. At end of every sub pass
B. Issuance of DOCLN macro
C. Operator "C"

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

Routine Size: 4 words, Routine Base: $CODES + 0664
Maximum stack depth per invocation: 2 words
The drop code is invoked by a DODU macro or a drop command, and contains any code that needs to be executed in conjunction with the dropping of a unit from the test cycle. No coding is required in this section.

The effect of a DODU is the same whether executed in the init code or in a hardware test. It invokes the drop unit coding and causes subsequent GPHARD'S for that logical unit to be returned - 'NOT COMPLETE'. This effect lasts only for the duration of the current command.

Routine Size: 1 word, Routine Base: $CODE1 * 0674
Maximum stack depth per invocation: 0 words
The add code is invoked by the ADD command, and contains any code that needs to be executed in conjunction with adding a unit back to the test cycle. No coding is required in this section.

Units may be added to the test sequence only through the use of operator ADD command. Each unit must have a P-TABLE in memory due to an earlier hardware dialogue (i.e., the unit was previously dropped).

The ADD code must be delimited by BGNAU, ENDAU. There is no particular lar coding required in the add code to cause the add to be effective:

The section is just for programmer housekeeping.

```
000000 000207
     LAU: RTS PC

000000 004737 000706
       L$AU: JSR PC,LAU
000004 104452
       TRAP 52
000006 000207
     RTS PC

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

```
end

PSECT SUMMARY

Psect Name       Words Attributes
$CODE#          232   RO, I, LCL, REL, CON
```

Library Statistics
<table>
<thead>
<tr>
<th>File</th>
<th>Total</th>
<th>Symbols</th>
<th>Percent</th>
<th>Pages</th>
<th>Mapped</th>
<th>Processing Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISK$USER2:([YOUNG.FMT]ZROB2.B16:4</td>
<td>358</td>
<td>104</td>
<td>29</td>
<td>18</td>
<td></td>
<td>00:00.1</td>
</tr>
</tbody>
</table>

**COMMAND QUALIFIERS**

BLISS/PDP11/LIST ZROB2.B16/EN;NOEIS/SOURCE=PAGE:53

- **Size:** 179 code + 53 data words
- **Run Time:** 00:15:4
- **Elapsed Time:** 02:12:3
- **Lines/CPU Min:** 7535
- **Lines/CPU-Min:** 47202
- **Memory Used:** 194 pages
- **Compilat' on Complete**
MODULE ZRQB3 (#TITLE 'RDRX DISK FORMATTER'
IDENT = 'REV C PATCH O',
ADDRESSING MODE (ABSOLUTE),
ENVIRONMENT (NOEIS)
)
BEGIN
!

MAIN CODE FOR FORMATTER

!Define RDRX formatter library
library 'ZRQB30.L16';

!Define Bliss Macro require file
require 'BLSMAC.REQ';

!module declaration
#include 'MODULE DECLARATIONS';
forward routine

btod,

REV8 : novalue,
REV9 : novalue;

! The psect named "code or $code" is redefined here
! to be called "abicode". This is done to organize the
! formatters test sections into a separate psect.
!
!
! psect
! code = abicode;
!
!
! Structure declarations used within this module.
!
!
structure

! RDX register accessing structure. This
! structure allows RDX register accessing
! to be transportable between the PDP 11 and
! VAX Diagnostic Supervisors.
!
! This also defines an access algorithm for
! VAX to allow field reference to MBA address
! space without generating machine checks.
!
RDX [O, P, S, E] =
begin
local
RC+5_REG;

RC+5_REG = (RDX + $bpval+0)<0, $bpval, 0>;

end

<P, S, E>;
! External Routines declared outside this module.

external routine

setcpu,

COPY,

ABORT,

GET_DUST_STATUS,

EX_LOC_PROG,

REC_DATA,

SEND_DATA,

SET_CNTL_CHAR,

DUPI_SERVICE : INT_LNK_TYP novalue,

INTI_SERVICE : INT_LNK_TYP novalue,

INIT_COM_AREA,

BOOT_RDRX,

DECODE : novalue;
EXTERNAL Declaration of datums declared outside of this module.

external

! Miscellaneous external data declarations

REC_ENVELOPE : blockvector [REC_ALLOCATE, RB_SIZE + 2, word] field (ENV_FIELD),

NXT_CRN : byte, ! Next sea command ref number

RET_ENHND : ref block (RB_SIZE + 2, word) field (ENV_FIELD),

SND_BUF : vector [SNDB_SIZE, word], ! DUP send cmd text buffer

REC_BUF : block [RECB_SIZE, word] field (RECB_FIELD), ! DUP receive cmd text buffer

MSGADR, ! Pointer to DM sent msg: text

NSD_SLOT : word, ! Stores next send ring slot to load cmd into

NRD_SLOT : word, ! Stores next receive slot to expect response in

VECADDR : word, ! Stores controllers vector address

RET_STATUS : word, ! Stores return status of called routines

PID_SAVE : word, ! Saves process indicator word

RDRX_ADDR : ref RDRX field (ISD_FIELD), ! RDRX reference structure

PTBL_PTR : ref vector [4, word], ! Table pointer for fetching unit number

SW_UNIT_NO : word, ! Software table unit number

SW_MODE : word, ! Software table mode

UC_VER : byte, ! Controller version number

UNIT_NO : word, ! Unit number to format

! Init sequence code error and informational messages

BOOT_FAILURE,

PROTO_VIOLATION,

PORT_INIT_ERR,

! Default values

DEF_SERIAL,

DEF_DATE,

! Printing format strings

FMT1,

FMT2,

FMT4,

FMT_ERR : vector [7],

DATMSG,

CRLF,

! Special questions so the formatter can save code.

UNIT# MSG,

EXIST# MSG,

DOWN# MSG,
<table>
<thead>
<tr>
<th></th>
<th>1662</th>
<th>1663</th>
<th>1664</th>
<th>1665</th>
<th>1666</th>
</tr>
</thead>
<tbody>
<tr>
<td>INACC1_MSG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DFLOG_MSG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SERIAL3_MSG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATE8_MSG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
FUNCTIONAL DESCRIPTION:
Implicit Inputs:
Implicit Outputs:
Completion Codes:
Side Effects:

LABEL TLOOP;

own RETRIES,

loopc;

The next thing to be done is to boot the RDRX controller. We will allow a few retries if not successful after the first boot before we considered the Controller dead.

But before we do the boot sequence the processors priority and interrupt vector address must first be loaded. During the init sequence the Init sequence interrupt service routine will just flag any interrupts and ignore them since interrupts are disabled during init sequence. Later the DUP interrupt service routine will be load and do the DUP communications protocol.

The following priorities will be assigned:
1. Processor will run at priority zero.
2. The RDRX runs at priority 5 by default.
3. The DUP interrupt routine will run at priority 7.

Set the processors priority
!Clear out the vector before setting
!Set the interrupt service priority

!Retry the RDRX booting until the
return is true or the retry limit
is reached.

!Set delay constant for timing loops

!Reset the retry counter

begin
RET_STATUS = BOOT_RDRX();
RETRIES = RETRIES + 1;
end
until (.RET, STATUS) or (.RETRIES equal ONE);

!Repeat the boot until done

!Did the Controller boot

!Report a boot error

!Report some statistical data

!Abort the program

! Now that the RDX controller is
! booted, check to make sure that the
! controller has done its part of the
! DUP protocol by clearing out the
! port communications area.
! While we're there set up the communication
! area for the up and coming communications
! between the remote and host program.

if INIT_COM_AREA ()

then

begin

PRINTB (PROTO_VIOLATION);
PRINTB (PORT_INIT_ERR);
DOCLN;
end;

!Was the com area cleared out as expected

!Com area not init'd so error and abort

Before writing the go to the IP register
load the DUP interrupt service addr into
the RDX vector address for the up and
coming communications between the host
and controller.

CLRVEC (.VEC_ADDR);
SETVEC (.VEC_ADDR, DUP services, P107);

!Clear out the vector before starting

!Set the interrupt service priority

! All systems are go. Start the controllers
! functional micro code off and running. Write
! to the SA Register with the go bit set and start
! things rolling.

WRIT_RDX (RCSA, RC_ALL, ONE);

!Let the controller go
FORMATER SECTION 2

Functional Description:

Implicit Inputs:

Implicit Outputs:

Completion Codes:

Side Effects:

Set the controller characteristics. Default values will be taken in all cases except for the host time out value which will be changed to 'wait for ever'.

IF SET_CNTLR_CHAR() THEN DECODE();

Call decode 'if not successful

Do a unit on line cmd which will spin up the device and load the heads. The unit to be placed on line will be the last unit number received from the 'gphard' macro in the init code.
`FORMATTER SECTION 3

1799  3  ! Call Decode if connection error
1800  3  ! See if the Dup server in the RDRX Controller is in an
1801  3  ! idle state. To do this first get the dust status and
1802  3  ! then look at the flag field bit 3 for a :
1803  3  ! 0 = idle
1804  3  ! 1 = active
1805  3  !
1806  3  !
1807  3  IF GET_DUST_STATUS () THEN DECODE ();
1808  3  ! Call Decode if connection error
1809  3  !Look in the flag field bit 3 to see if the server is active.
1810  3  !
1811  3  !
1812  3  IF _RET_ENVAD [FLG_B3]
1813  3  ! If the server is active exit and reboot
1814  3  then
1815  4  begin
1816  5  !-----------------------
1817  5  ! Do some stat recording
1818  5  !-----------------------
1819  5  DOCLN;
1820  5  ! Exit this passes execution
1821  5  end;
1822  3  !
1823  3  ! The server is not active so down line load the formatter
1824  3  and start its execution by issuing a "Execute local program".
1825  3  Call the decode routine if a connection error is detected.
1826  3  !
1827  3  !
1828  3  IF EX_LOC_PROG () THEN DECODE ();
1829  3  ! Call decode if connection error
1830  3  ! Get the dust status to see if the server is in an active
1831  3  ! state. An active state is what we want so error 'if the
1832  3  ! server is in an idle state.
1833  3  !
1834  3  IF _GET_DUST_STATUS () THEN DECODE ();
1835  3  ! Call decode if connection error
1836  3  ! Look at the flag field bit 3 to see if the server is active.
1837  3  !
1838  3  !
1839  3  if not (_RET_ENVAD [FLG_B3])
1840  3  ! Reboot if server is idle
1841  3  then
1842  4  begin
1843  5  !-----------------------
1844  5  ! Do some stat recording
1845  5  !-----------------------
1846  4  !`
DOCLN;
end

else

PID_SAVE = .RETNAD [PLO_IND];

!Save the progress indicator

!
! The Dux server is in the active state running the formatter
! program. This DO LOOP will loop on the DUX sub protocol
! doing the "send and receive" data commands. These commands
! establish the communications between this host program and
! the remote formatter program running in the RDUX controller.
!
loopc = 0;

while .loopc eq 0 do

begin

!
! Do a 'Receive_data' command which pulls the remote program
! for a message. The returned message can either be a:
!
1. Question
! Where the ascii text is a prompt for information.
!
2. Default question
! Where the default question message is identical
! to the question message except that a null (zero
! length) send data is taken to be a default answer
! to the question.
!
3. Information
! Where the ascii text is an informative message.
!
4. Termination
! Where the ascii text is a normal termination message.
!
5. Fatal Error
! Where the ascii text is a fatal error message.
!
6. Special
! This type is used when only a host program could
! respond.
!
CLR_RBUF;
if REC_DATA () then DECODE ();

!Clear out the receive buffer area
!Call decode if connection error

!
! From the first word in the send/receive data buffer, look
! to see what type message the remote program has sent to
! the Host program. Use this message type number to index
! into the select expression to perform the requested action
! by the remote program.
! CLR.SBUF;
! CMTR (MSGADR, REC.ENVELOPE [.NSD.SLOT, BLK_CNT] - 2) x 0, 8 + 0;
if .UC_VER leq 8 then REV8 (loopc)
! be compatible with version 8 and 9
else REV9 (RETRIES, loopc);

end:

if :loopc equ 1 then begin
begin
EXIT.TST;

! normal termination
end

else begin
DDCLN;

! error abort
end;

ENDTST;

.TITLE ZRQB3 RDX DISK FORMATTER
.IDENT /REV C /
.ENABLE AMA

.PSEC 10W$, D
000000 RETRIES: BLKW 1
000002 LOOPC: BLKW 1

.GLOBL SETCPU, COPY, ABORT, GET.DUST.STATUS
.GLOBL EX.LOC.PROG, REC.DATA, SEND.DATA
.GLOBL SET.CNTLR.CHAR, DUPF$.SERVICE
.GLOBL INTI$.SERVICE, INIT.COM.AREA, BOOT.RDX
.GLOBL DECODE, REC.ENVELOPE, NXT.CRN
.GLOBL RET.ENIAD, SND.BUF, REC.BUF, MSGADR
.GLOBL NSD.SLOT, NRD.SLOT, VEC.ADDR, RET.STATUS
.GLOBL PID.SAVE, RDX.ADDR, P1BL.PTR
.GLOBL SW_UNIT.NO, SW.MODE, UC.VER, UNIT.NO
.GLOBL BOO.T.FAILURE, PROT.VIOLATION
.GLOBL PORT.INIT.ERR, DEF.SERIAL, DEF.DATE
.GLOBL FMT1, FMT2, FMT4, FMT.ERR, DATHMSG
.GLOBL CRFL, UNIT$.MSG, EXIST$.MSG, DOWN$.MSG
.GLOBL INACC$.MSG, DFLT1$.MSG, SERIAL$.MSG
.GLOBL DATE$.MSG

.SBTTL $T1 FORMATTER SECTION 1
.PSEC 4COF$, RO

000000 010146 $T1: MOV R1, (SP) ;
0000002 162706 000006 ;
0000006 005000 CLR R0 ;
0000010 104441 TRAP 41 ;

1665
1°02
000012 013700 000000G
000016 104436
000020 012746 000040
000024 012746 000000G
000030 012746 000000G
000034 012746
000040 010437
000042 004737 000000G
000046 012747 177777 000000
000054 004737 000000G
14: MOV 01, RETRIES
JSR PC,BOOT,RO
MOV RO,RET STATUS
INC RETRIE
BIT #1,RO
*:*,RET STATUS
18: MOV #1, (SP)
MOV SP,RO
*:SP,*

31: JSR PC,INIT.COM.AREA
ROR RO
BCC 43
MOV #PROTO.VIOLATION,(SP)
MOV #1,(SP)
MOV SP,RO
*:SP,*

41: MOV VEC.ADDR,RO
TRAP 36
MOV #340,(SP)
MOV #DUP,#SERVICE,(SP)
MOV VEC.ADDR,(SP)
MOV #3,(SP)
TRAP 37
MOV RDX,ADDRO
MOV 2(RO),16(SP)
MOV #1,R1
MOV R1,2(RO)
MOV PC,SET.CNTLR.CHAR
ROR RO
BCC 51
JSR PC,DECODE
set

[0]:
!Enter unit number to format
begin
if (MANUAL) then
  GMANID (UNIT MSG, SEND BUF, 0, $o$'3', 0, 3, NO)
else SND BUF = .SW UNIT NO;
if SEND_DATA (SNDB.COUNT) then DECODE ();
end;

[1]:
!Choose one of the responses
begin
if (MANUAL) then
  begin
  SND BUF = 0;
  GMANIL (EXIST MSG, SND BUF, $o$'1', YES, 1);
  if (.SNDB.<0, 1, 0 > NEQU 1)
    then begin
      GMANIL (DOWN MSG, SND BUF, $o$'2', YES, 1);
      if (.SNDB.<1, 1, 0 > NEQU 1)
        then begin
          GMANIL (INACC MSG, SND BUF, $o$'4', YES, 1);
          if (.SNDB.<2, 1, 0 > NEQU 1)
            then begin
              SND BUF<0, 1, 0 > = 1;
              PRINTB(DFLT MSG);
            end
        end
    end
  end;
else SND BUF = .SW_MODE;
if SEND_DATA (SNDB.COUNT) then DECODE ();
end;

[2]:
!Enter an eight character non-zero ASCII serial number
(Any eight characters will do)
if (MANUAL) then GMANID (SERIAL MSG, SND BUF, A, $o$'17777', 8, 8, NO)
else copy (DEF SERIAL, SND BUF, 8);
if SEND_DATA (SNDB.COUNT) then DECODE ();
end;

[3]:
!Enter current date MM-DD in exact format required
(Actually, any six characters will do)
if (MANUAL) then GMANID (DATE MSG, SND BUF, A, $o$'17777', 8, 8, NO)
else copy (DEF DATE, SND BUF, 8);
if SEND_DATA (SNDB.COUNT) then DECODE ();
end;

[otherwise]:
!This message number is unknown
begin
RET STATUS = UNKN CODE;
if Unknown message number error code
DECODE ();
end;
[2]:
begin
selectone .REC BUF (MSG NUM) of
set
[otherwise]:
begin
RET STATUS = UMN CODE;
DECODE ();
end;
tes;
end;
!Default Question

[3]:
begin
selectone .REC BUF [MSG_NUM] of
set
[0, 9]:
begin
PRINTB (FMT1, MSGADR);
end;
end;
![Informational message]

[4]:
begin
.endflg = 1;
PRINTB (FMT4, .REC_BUF[MSG_NUM]);
end;
![Termination message]

[5]:
begin
PRINTB (.FMT ERR [.REC BUF[MSG_NUM] 1]);
.endflg = 2;
end;
![Fatal error message]

[6]:
begin
!
Special message type
selectu .REC BUF [MSG NUM] of
set
always:
begin
REP STATUS = UMN CODE;
DECODE ();
end;
tes;
end;
[otherwise]:
begin
REP STATUS = UMT CODE;
!This message number is unknown
!Unknown message number error code
!Report error and die
DECODE ();
end;
tes;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
routine REV9 (retries, endflg): novalue -

begin
local
  capt,
  enum
selectoneu .REC_BUF [MSG.TYP] of
  |Select the appropriate action
set

(1):
|Question message type
begin
|Look into the send/receive data buffer at the message
|number field and see what question the remote program
|is asking. Use the Fields value to index into the
|select expression to perform the appropriate action.
|
selectoneu .REC_BUF [MSG.NUM] of
  |Select the requested question
set

(0):
|Enter current date <MM DD YYYY>
begin
|If in unattended reformat mode then give to the DM the
|date received from the init code else let the DM prompt
|the operator for the date.
|
|If (MANUAL) !Is the host running in unattended mode
then
begin
  GMAND (MSGADR, SND_BUF, A, A'177777', 8, 10, NO);
end
else
begin
C8

REV C PATCH 0  FORMATTER SECTION 1

if ..retrys
then
begin

! Clear out the date text buffer before loading in
! date and ask the operator for the date again.
!

incr : from 0 to 11 do
SNDBUF[:,1] = ZEROS;

GMANID (DATMSG, SND_BUF, A, ".0'177777'", 8, 10, NO); !Get date from operator
end;

retrys = ONE;
!Set the retry flag
end;

PRINTB (CRLF);
if SEND_DATA (SNDB_COUNT) then DECODE ();
end;

[2]:
begin

! get cylinder number
GMANID (MSGADR, SND_BUF, D, ".0'177777'", 0, 999, NO);
btod (.SNDBUF, SND_BUF);
if SEND_DATA (SNDB_COUNT) then DECODE ();
end;

[3]:
begin

GMANID (MSGADR, SND_BUF, D, ".0'177777'", 0, 11700, NO);
btod (.SNDBUF, SND_BUF);
if SEND_DATA (SNDB_COUNT) then DECODE ();
end;

[7]:
begin
!Enter a non-zero serial number
SNDB_COUNT;
if (MANUAL)
then begin
GMANID (MSGADR, SND_BUF, A, ".0'177777'", 1, 64, NO);
PRINTB (CRLF);
cptr = chipset (.SNDBUF);
cnum = 0;
while .chipstr (.cptr) < 0, 8 neau 0
do begin
   cnum = .cnum + 1;
cptr = chipset (.cptr, 1);
end;
if .cnum AND 1 neau 0
then begin
:: 2153 6
:: 2154 6
:: 2155 5
:: 2156 5
:: 2157 4
:: 2158 4
:: 2159 4
:: 2160 5
:: 2161 3
:: 2162 3
:: 2163 4
:: 2164 4
:: 2165 4
:: 2166 3
:: 2167 3
:: 2168 3
:: 2169 2
:: 2170 2
:: 2171 2
:: 2172 2
:: 2173 3
:: 2174 3
:: 2175 3
:: 2176 3
:: 2177 3
:: 2178 4
:: 2179 4
:: 2180 4
:: 2181 4
:: 2182 4
:: 2183 4
:: 2184 4
:: 2185 5
:: 2186 4
:: 2187 5
:: 2188 5
:: 2189 5
:: 2190 4
:: 2191 5
:: 2192 5
:: 2193 5
:: 2194 4
:: 2195 4
:: 2196 4
:: 2197 4
:: 2198 4
:: 2199 4
:: 2200 4
:: 2201 4
:: 2202 3
:: 2203 3
:: 2204 3
:: 2205 4

CHIPRA (.cptr)=0, B> = 'C', '
end;
cnum = .cnum + 1
end;
else
  copy (DEF_SERIAL, SND_BUF, B);
  if SEND_DATA (.cnum) then DECODE ();
end;

[otherwise]:

begin
  if (MANUAL) then
    begin
      GHANID (MSGADR, SND_BUF, D, '0177777', 0, 15, YES);
      end
else
  begin
    SND_BUF = .UNIT_NO;
    PRINTB (FM12, .UNIT_NO);
    SND_BUF = .SNODE.COUNT;
    PRINTB (CRLF);
    if SND_BUF geq 10 then
      SND_BUF = .SNODE.COUNT
    else
      SND_BUF = .SNODE.COUNT
        + '0';
    if SEND_DATA (SNODE.COUNT) then DECODE ();
    end;
end;

[2]:
get head number (CR to signal end)
begin

[2]:
!Default Question

begin
selectoneu .REC_BUF [msg_num] of
set

[1]:
Enter unit number to format <0>
begin

!If in unattended reformat mode then give to the DM the unit number obtained from the hardware P_Table else let the DM prompt the operator for the unit to format.

if (MANUAL) then
  begin
    GHANID (MSGADR, SND_BUF, D, '0177777', 0, 15, YES);
    end
else
  begin
    SND_BUF = .UNIT_NO;
    PRINTB (FM12, .UNIT_NO);
    SND_BUF = .SNODE.COUNT;
    PRINTB (CRLF);
    if SND_BUF geq 10 then
      SND_BUF = .SNODE.COUNT
    else
      SND_BUF = .SNODE.COUNT
        + '0';
    if SEND_DATA (SNODE.COUNT) then DECODE ();
    end;
end;

[2]:
get head number (CR to signal end)
begin

[2]:
!This message number is unknown
begin
  RET_STATUS = UNN_IDE;
  DECODE ();
  Report error and die
end;
end:

end;
end:

[2]:
!Default Question

begin
selectoneu .REC_BUF [msg_num] of
set

[1]:
Enter unit number to format <0>
begin

!If in unattended reformat mode then give to the DM the unit number obtained from the hardware P_Table else let the DM prompt the operator for the unit to format.

if (MANUAL) then
  begin
    GHANID (MSGADR, SND_BUF, D, '0177777', 0, 15, YES);
    end
else
  begin
    SND_BUF = .UNIT_NO;
    PRINTB (FM12, .UNIT_NO);
    SND_BUF = .SNODE.COUNT;
    PRINTB (CRLF);
    if SND_BUF geq 10 then
      SND_BUF = .SNODE.COUNT
    else
      SND_BUF = .SNODE.COUNT
        + '0';
    if SEND_DATA (SNODE.COUNT) then DECODE ();
    end;
end;

[2]:
get head number (CR to signal end)
begin

[2]:
!This message number is unknown
begin
  RET_STATUS = UNN_IDE;
  DECODE ();
  Report error and die
end;
end:

end;
end:

end;
end:

end:

end:
end:

end:
end:

end:
end:

end:
end:

end:
end:

end:
end:

end:
end:

end:
end:

end:
end:

end:
end:

end:
[6]:
begin
if (MANUAL) !Is the host running in unattended mode
then
  begin
  GMANIL (MSGADR, SND_BUF, \#O'1', YES, 1);
  SND_BUF = SND_BUF + \#C'Y' - 1;
  end;
else
  begin
  SND_BUF = \#C'N'; !Default not to continue
  end;
end;

PRINTB (CRLF);

! If SEND DATA (SNDB_COUNT) then DECODE ();
end;

[otherwise]: !This message number is unknown
begin
  GMANID (MSGADR, SND_BUF, A, \#O'177777', 1, 64, NO);
  if SEND_DATA (SNDB_COUNT) then DECODE ();
end;
tes;
end;

[3]:
begin
  ! Informational message
  PRINTB (CRLF);
  PRINTB (FMT1, MSGADR);
end;

[4]:
begin
  ! Termination message
  PRINTB (CRLF);
  PRINTB (FMT1, MSGADR);
  .endflg = 1;
  ! Mark as normal termination
end;

[5]:
begin
  ! Fatal error message
  PRINTB (CRLF);
  PRINTB (FMT1, MSGADR);
  .endflg = 2;
  ! Mark as error
end;
!Special message type

!This message number is unknown

!Unknown message number error code

!Report error and die
001250 104414 TRAP 14
001252 012716 MOV #MSGADR, (SP)
001256 012746 MOV #MT1, (SP)
001252 012746 MOV #2, (SP)
001256 012746 MOV SP, R0
001270 104414 TRAP 14
001272 012776 MOV #2, #20(SP)
001300 062706 341: AOD $10, SP
001304 000207 351: RTS PC
001306 020127 351: CMP R1, #6
001312 001004 BNE 371
001314 012737 361: MOV #4001, RET. STATUS
001320 000403 BR 381
001324 012737 371: MOV #1001, RET. STATUS
001332 004737 381: JSR PC, DECODE
001336 000207 391: RTS PC

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

```
routine btod(num, ptr) =
begin
local i, j;
i = 1;
j = 0;
while .num/.i geq 10 do i = .i x 10;
while .i geq 1 do
begin
j = .j + 1;
chptr(.ptr)<0,8> = $C0 + .num/.i;
ptr - chplus(.ptr, 1);
num = .num + .num/.i
i = .i/10;
end;
return .j;
end;
```

000000 004137 000000G BTOD: JSR R1, $SAVE3
000004 012701 000001 MOV $1, R1
000010 005002 CLR R2
000012 016646 00014 MOV 14(SP), -(SP)
000016 010146 MOV R1, -(GP)
000020 004737 00000G JSR PC, BLDIV
000024 022626 CMP (SP), (SP)
000026 000027 CMP R0, #12
000032 103410 BLO 2$
000034 010146 MOV R1, -(SP)
000036 012746 000012 MOV $12, -(SP)
JSR  PC,BLIMUL
MOV  R0,R1
CMI (SP),(SP)
BR  11
2$:  TST  R1
BEQ  3$
INC  R2
MOV  14(SP),(SP)
MOV  R1,(SP)
JSR  PC,BLDIV
MOV  R0,R3
ADD  #60,R3
MOVB  R3,B16(SP)
JSR  PC,BLIMUL
INC  16(SP)
MOV  R0,(SP)
MOV  R1,(SP)
JSR  PC,BLIMUL
SUB  R0,22(SP)
MOV  R1,(SP)
MOV  #12,(SP)
JSR  PC,BLDIV
MOV  R0,R1
ADD  #10,SP
BR  29
3$:  MOV  R2,R0
RTS  PC

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

; 2346 1
; 2347 1  end
; 2348 0  eludom

OTS external references
.GLOBL $SAVE3, $SAVE2, BLDIV, BLIMUL

PSECT SUMMARY

<table>
<thead>
<tr>
<th>Psect Name</th>
<th>Words</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1OWNS</td>
<td>2</td>
<td>RW, D, LCL, REL, CON</td>
</tr>
<tr>
<td>$CODE$</td>
<td>847</td>
<td>RO, I, LCL, REL, CON</td>
</tr>
</tbody>
</table>

Library Statistics

Symbols -------  Pages  Processing
COMMAND QUALIFIERS


Size: 847 code + 2 data words
Run Time: 00:28.2
Elapsed Time: 02:26.6
Lines/CPU Min: 4993
Lexemes/CPU-Min: 32641
Memory Used: 306 pages
Compilation Complete
0001 0 MODULE ZRQB4 (#TITLE 'RDRX DISK FORMATTER'
    IDENT = 'REV C PATCH 0',
    ADDRESSING (ABSOU)TE,)
    ENVIRONMENT (NOEIS)*)

0006 1 BEGIN
0007 1 ! SUBROUTINES
0008 1 ! Define RDRX Formatter library
0009 1 ! Define Bliss Macro require file
0010 1 LIBRARY ZRQBB0.L16;
0011 1 REQUIRE 'Blissmac.REQ';
0148 1 #include MODULE DECLARATIONS
forward routine

GET_NSD, !Get next send descriptor slot index
GET_NRD, !Get next receive descriptor slot index
LOAD_OUT#STD_BUF, !Load out standing command buffer
GET_CMD#REF, !Get unique command reference number
DECODE, novalue, !Decode return status error code
DUP#I_SERVICE, INT_LNK#TP, novalue, !Dup/UQ port interrupt service routine
CTO_WAIT, !Command time out wait
ABORT, !Abort Dup command
GET_OUST_STATUS, !Get Oust Status command
EX_LOC_PROG, !Execute Local Program command
SEND_DATA, !Send Data command
REC_DATA, !Receive Data command
SET_CNTL#CHAR, !Set Controller Characteristics command
INITI#SERVICE, INT_LNK#TP, novalue, !Initialization sequence interrupt service
IS_TIMER, !Initialization sequence time-out wait
BOOT_RDX, !Initialization sequence for RDRX controller
INIT_COM#AREA, !Initialize UQ Port communication area
The psect named "code or $code:" is redefined here to be called "sclicode." This is done to organize formatter routine code into a separate psect.

`psect code = sclicode;`

Structure declarations used within this module.

```
structure

! RDRX register accessing structure. This structure allows RDRX register accessing to be transportable between the PDP-11 and VAX Diagnostic Supervisors.

This also defines an access algorithm for VAX to allow field reference to MBA address space without generating machine checks.

RDRX [0, P, S, E] =

    begin
    local
        RC$ REG:
        RC$ REG = (RDRX + $upval=0)<0, $bupval, 0>;
        RC$ REG
        end

    <P, S, E>:
```

```
External Declaration of datums declared outside of this module.

external

Communications area declarations

COMM_AREA : blockvector [REC_ALLOCATE, SND_ALLOCATE, HDR_SIZE, 2, word],
HEAD_AREA : ref block [4, word] field (HDR_FIELD),
RECEIVE_RING : ref blockvector [REC_ALLOCATE, 2, word] field (DSC_FIELD),
SEND_RING : ref blockvector [SND_ALLOCATE, 2, word] field (DSC_FIELD),
RECV_ENVELOPE : blockvector [REC_ALLOCATE, RB_SIZE, 2, word] field (ENV_FIELD),
SEND_ENVELOPE : blockvector [SND_ALLOCATE, SB_SIZE, 2, word] field (ENV_FIELD),
RECV_ENHAD : ref block [RB_SIZE, 2, word] field (ENV_FIELD),
RECV_BUF : block [RECV_SIZE, word] field (RECV_FIELD),
SEND_BUF : vector [SEND_SIZE, word],
OUTSTD_BUF : Blockvector [REC_ALLOCATE, 2, word] field (OUTFIELD),

Miscellaneous external data declarations

RD_TABLE : word,
NSD_TABLE : word,
RDRX_ADDR : ref RDRX field (ISD_FIELD),
RET_STATUS : word,
PID_SAVE : word,
VEC_ADDR : word,
RCVER : byte,
RSVE_STRUCT : vector [4, word],
ISD_STRUCT : blockvector [4, 2, word] field (ISD_FIELD),
UNIT_ID : word,
NXT_CRN : byte,

Error Messages Structures

PFE_STRUCT : vector [22],
EMSG_STRUCT : vector [22],

Error message structure
global routine GET_NS_D

!This routine will determine which send ring descriptor the port/controller is polling and returns that desc slot number to the calling routine. This host program will call this routine each time it wishes to deposit another command into the command (send) ring.

Formal Parameters:
none

Implicit Inputs:
NSD_SLOT: Global storage for the next send descriptor slot.
Stores where the host should place this command for processing by the port/controller.

Implicit Outputs:
The global storage "NSD_slot" is updated to the present send slot where the port/controller is polling.

Completion Codes:
Returns the contents of "NSD_slot" to the calling routine.

Side Effects:
none

begin

Increment the next send descriptor_slot by one

NSD_SLOT = NSD_SLOT + 1;

Set the slot pointer back to zero if it wraps around to the top of the ring.

if .NSD_SLOT greater than SND_ALLOCATE 1 then NSD_SLOT = ZERO;

Return the next send descriptor_slot to the caller

return .NSD_SLOT;
end;

.TITLE ZRQB4 RDRX DISK FORMATTER
.IDENT /REV C /
.ENABLE AHA
.GLOBL COM_AREA, HEAD_AREA, RECEIVE_RING
.GLOBAL SEND,RING, REC,ENVELOPE, SND,ENVELOPE
.GLOBAL RDRX,ADRX, REC,BUF, SND,BUF, OUT,STD,BUF
.GLOBAL RDRX.SLOT, NSD.SLOT, RDRX.ADDR
.GLOBAL RET,STATUS, PID, SAVE, VEC, ADDR
.GLOBAL UC, VER, RSVD, STRUCT, ISD, STRUCT
.GLOBAL UNIT, NO, NXT, CRN, PFE, STRUCT, EMSG, STRUCT

.SECTION GET, NSD GLOBAL ROUTINE DECLARATIONS
.PSECT $CODE$, RO

000000  005237  000000G GET.NSD:
000000  000004  027327  000000G  000003
000012  101402
000014  005037  000000G
000020  013700  000000G
000024  000207  1:  INC NSD.SLOT
                    CMP NSD.SLOT,03
                    BLS 1:
                    CLR NSD.SLOT
                    1:  MOV NSD.SLOT,RO
                    RTS PC

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

;  1692  1
global routine GET.NRD =  ! Chooses the next receive slot

! Functional Description:
! This routine will determine which receive ring descriptor the
! port/controller's polling and returns that desc slot number to
! the calling routine. This host program will call this routine
! each time it wishes to process another receive ring descriptor.

! Formal Parameters:
none

! Implicit Inputs:
NBD_SLOT : Global storage for the next receive descriptor slot.
Stores where the port should return this command's response indicator.

! Implicit Outputs:
The global storage "Nbd_slot" is updated to the
present receive slot where the port/controller
is polling.

! Completion Codes:
Returns the contents of "Nbd_slot" to the calling routine.

! Side Effects:
none

begin
! Increment the next receive descriptor_slot by one
! NBD SLOT = NBD SLOT + 1;

! Set the slot pointer back to zero if it wraps around
! to the top of the ring.
! if .NBD SLOT gttru REC_ALLOCATE 1 then NBD SLOT = ZERO;

! Return the next receive descriptor slot to the caller
 return .NBD SLOT;
end;
ZRQB4  RDX DISK FORMATTER
REV C PATCH 0  GLOBAL ROUTINE DECLARATIONS

000014  005037  000000G
000020  013700  000000G
000024  000207

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

1-30 1
global routine LOAD OUTSTD_BUF (REF_NUM) = 

The outstanding command buffer "outstd.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 outstd.buf slot the following values:

1. This command's 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 guarantee a command loaded into the outstd.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 outstd.buf slot to the unused pool (i.e. by loading a slot's first word with '0'100000'). This routine is therefore guaranteed to return an unused outstd.buffer slot when this unique value of '0'100000' is found. To further guarantee 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 outstanding buffer slot index where this command was put is routines value and is returned to the caller.

Side Effects:

none
begin

if .OUTSTD_BUF[., CMU WRD] equl $'100000' then
  return .i1;
end;

if .OUTSTD_BUF[., CMU WRD] equl $'100000' then
  return .i1;
end;

return RET STATUS = 0BF CODE;

begin
  if .OUTSTD_BUF[., CMU WRD] equl $'100000' then
    return .i1;
end;

if .OUTSTD_BUF[., CMU WRD] equl $'100000' then
  return .i1;
end;

begin
  if .OUTSTD_BUF[., CMU WRD] equl $'100000' then
    return .i1;
end;

begin
  if .OUTSTD_BUF[., CMU WRD] equl $'100000' then
    return .i1;
end;

The buffer is full if the code reaches here. This should never happen so report an error for debug purposes.

return RET STATUS = 0BF CODE;

Report an 'OUTSTD BUF FULL' error
000052 020227 000003   CMP   R2,#3
000056 101753           BLOS   #1
000060 012700 002001   MOV   #2001,RO
000064 010037 000000G   MOV   RO,RET.STATUS
000070 000207           RTS   PC

; Routine Size:  29 words.  Routine Base:  $CODE$ + 0054
; Maximum stack depth per invocation:  4 words
global routine GET_CMDREF + ! gets next unique cmd ref number

! Functional Description:
A 32 bit unique non-zero number used to identify host commands.
Class drivers should supply a unique reference number in each
command that is sent to a DUP server. A class driver may supply
a zero reference number if it does not need to associate a command
with its end message.

Command reference numbers must be unique across all commands that
are outstanding on the same connection i.e., they must be unique
across all outstanding commands issued by a single class driver
(Host) to a single DUP server. The class driver may re-use a
commands reference number when the command is no longer
outstanding -- i.e., after receiving the commands end message or
after re-synchronizing with the DUP server. Command reference
numbers need not be unique for commands issued by different class
drivers --- i.e. commands issued by different host or commands for
different DUP servers from the same host. Therefore controllers
must internally use the combination of a command reference number
and the connection on which the command was received as the unique
identifier of an outstanding command.

This routine will generate a unique command reference number and
will search the outstanding command buffer to see if already used.
The first unused unique command reference found will be returned
to the calling routine.

Formal Parameters:
none

Implicit Inputs:
NXT_CRN This global location stores the next unique cmd
reference number to be used.

Implicit Outputs:
NXT_CRN This global location is loaded with the next
unique command reference number.

Completion Codes:
The contents of .NXT_CRN is returned to the calling routine.

Side Effects:
none

begin
local DONE:
Increment the global unique command
reference number before anything is done.

NXT_CRN = .NXT_CRN + 1;

\!
\! Repeat generating and searching the outstanding
\! command buffer until a unique command reference
\! number is found.
\!
do
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
R10

REV C PATCH O GLOBAL ROUTINE DECLARATIONS

000002 105237 0000000G
000006 123727 0000000G 000377
000014 103405
000016 112737 000001 U00000G
000024 012701 000001
000030 005000
000032 126037 0000000G 000000L
000040 001004
000042 105237 0000000G
000046 005001
000050 000405
000052 062700 000004
000056 020027 000014
000062 101763
000064 006001
000066 103347
000070 005000
000072 153700 000000G
000076 012601
000100 000207

INC B NXT.CRN
CMPB NXT.CRN,#377
BLO 20
MOVB @1,NXT.CRN
MOV @1,R1
CLR RO
CMPB OUTSTD.BUF(R0),NXT.CRN
BNE 40
INC NXT.CRN
CLR R1
BR 51
ADD #1,RO
CMP RO,#14
BLDS 51
ROP R1
BCC 18
CLR RO
BISB NXT.CRN,R0
MOV (SP),R1
RTS PC

Note: Routine Size: 35 words, Routine Base: 1CODE1 + 0146
Maximum stack depth per invocation: 2 words

1929 1
global routine DECODE : no-value -  !Decodes failing SA reg data

... Functional Description:

Due to the implementation of the DUPI and DUPI Port protocol there
are two levels at which an issued command to a port/controller
field could have an error or status other than success posted.

1. The issued command can time out.

2. An error can be posted in SA register bit 15 by the port to
report an error.

3. The issued command to the port/controller can be executed
correctly without any errors but the response packet status
field could have an error or status other than success posted.

These errors or status's returned are all returned to the host
routine which queues the DUPI command via the global storage
"RET_STATUS". The host to port/controller communications
connection having the highest priority (ie. if the SA Reg error
bit is set the DUPI interrupt routine returns a PFE_CODE and this
connection will then be called when the return from a queued
command comes back with an error code or non-successfull status
code. This is by definition when bit 0 in the returned status
is equal to 1.

An appropriate recovery action will be done for each individual
error.

Formal Parameters : none

Implicit Inputs :

RET_STATUS: Stored in this global storage is the returned error
code or non-successful status code from a queued
command.

Implicit Outputs: none

Completion Codes: none

Side Effects : All formatter errors are fatal, therefor after execution
of this routine the RODX controller is initialized
aborting any DM code running in the controller.

begin
  PRINTF (.EMSG_STRUCT [MSG0]);
end;

"Port Protocol violation" error code
A protocol violation error was detected during
host processing of an issued command.

[PVE_CODE]:
begin
  PRINTF (.EMSG_STRUCT [MSG1]);
end;

"Remote program died" error code
This indicates that the remote program running
in the DM machine did not respond within the
designated time out interval and that the progress
indicator was not increased after subsequent time out
delays. It is assumed that the remote program is dead
and is treated as a fatal error.

[RPD_CODE]:
begin
  PRINTF (.EMSG_STRUCT [MSG2]);
end;

"Port to host synchronous error" code

[PSE_CODE]:
begin
  PRINTF (.EMSG_STRUCT [MSG5]);
end;

"Message length error" code

[MLE_CODE]:
begin
  PRINTF (.EMSG_STRUCT [MSG6]);
end;

"Unknown end code" error code

[UEC_CODE]:
begin
  PRINTF (.EMSG_STRUCT [MSG7]);
end;
! Code equals \#0'201'

! Code equals \#0'301'

! Code equals \#0'401'

! Code equals \#0'501'

! Code equals \#0'601'

! Code equals \#0'701'

! Code equals \#0'1001'
begin
    PRINTF (.EMSG_STRUCT [MSG16]);
end;

; "UNKNOWN MESSAGE NUMBER" error code

[[EOF_CODE]:

    !Code equals '0001'

begin
    PRINTF (.EMSG_STRUCT [MSG19]);
end;

; Out standing buffer slots are all filled up

[[OBF_CODE]:

    !Code equals '0201'

begin
    PRINTF (.EMSG_STRUCT [MSG17]);
end;

; Out standing command buffer out of sync error

[[OSE_CODE]:

    !Code equals '3001'

begin
    PRINTF (.EMSG_STRUCT [MSG18]);
end;

; File read error from local load media

[[FRE_CODE]:

    !Code equals '5001'

begin
    PRINTF (.EMSG_STRUCT [MSG20]);
end;

; This is here to trap any unknow return status codes
; sent to this routine.

[otherwise]:

    !Code equals non of the above

begin
    PRINTF (.EMSG_STRUCT [MSG3]);
end;

tes;

; All errors are fatal so init the RDRX
; and jump to the clean up code section
; to abort this units format.

[[WRI_RDRX (RSIP, RC ALL, ONES)];

DOCLN;

    !Init the controller

[[JUMP to the clean up code section

Page 20
```
000000 010146
000002 024646
000004 013701 000000G
000010 020127 000001
000014 001007
000016 013746 000010G
000022 012746 000001
000026 010600
000030 010441
000032 000577
000034 020127 000011
000040 001007
000042 013746 0000052G
000046 012746 000001
000052 010600
000054 010441
000056 000577
000060 020127 000021
000064 001017
000066 013700 000000G
000072 016016 000002
000076 011600
000080 042700 174000
000104 006300
000106 016046 000000G
000112 012746 000001
000116 010600
000120 010441
000122 000567
000124 020127 000031
000130 001007
000132 013746 000000G
000136 012746 000001
000142 010600
000144 010441
000146 000567
000150 020127 000041
000154 001007
000156 013746 000002G
000162 012746 000001
000166 010600
000170 010441
000172 000567
000174 020127 000051
000182 001007
000186 013746 000004G
000190 012746 000001
000194 010600
```

```
DECODE : MOV R1, (SP)
CMP - (SP), - (SP)
MOV RETSTATUS, R1
CMP R1, #1
BNE 1
MOV EMSG, STRUCT+10, (SP)
MOV #1, (SP)
MOV SP, R0
TRAP 17
BR 13:
CMP R1, #11
BNE 21
MOV EMSG, STRUCT+52, (SP)
MOV #1, (SP)
MOV SP, R0
TRAP 17
BR 15:
CMP R1, #21
BNE 31
MOV RDRX, ADDR, R0
MOV 2 (RO), (SP)
MOV (SP), RO
BIC #1, SP, R0
ASL RO
MOV PFE, STRUCT (RO), -(SP)
MOV #1, (SP)
MOV SP, R0
TRAP 17
BR 17:
CMP R1, #31
BNE 41
MOV EMSG, STRUCT, -(SP)
MOV #1, -(SP)
MOV SP, R0
TRAP 17
BR 19:
CMP R1, #41
BNE 51
MOV EMSG, STRUCT+2, (SP)
MOV #1, -(SP)
MOV SP, R0
TRAP 17
BR 21:
CMP R1, #51
BNE 61
MOV EMSG, STRUCT+4, -(SP)
MOV #1, (SP)
MOV SP, R0
```
000674 012700 177777
000700 010077 000000G
000704 104444
000706 022626
000710 012601
000712 000207

MOV @1,RO
MOV RO,RRDRX.ADDR
TRAP 44
CMP (SP)++, (SP)++
MOV (SP)++,R1
RTS PC

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

1930

2197 1
global routine DUP$1 SERVICE : IN1_LNK$1P novalue = Signals receive queue entry

... 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 purger 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 UO 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_buff: 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
! Side Effects:

begin

local

TEMP, ! Holds nrD_slot + 1 reference
FOUND_CMD, ! Found command flag
REF_NUM; ! Stores response packets cmd ref number

! Before this interrupt service routine does anything
! look at the SA register for any port fatal errors
! posted. If there are errors posted then report the
! error and kick the bucket.
!
if .RDRX_ADDR [RCSA, ERR_BIT] then

begin

REF_STATUS = PFE_CODE;
DECODE ();
return;
end;

! See what kind of interrupt got us here.

We could have a:

1. Response ring transition interrupt.
2. Send ring transition interrupt.
3. A adaptor request interrupt (which is
   illegal) running under the PDP-11 Diagnostic
   supervisor and is flagged as a fatal controller
   error.)
4. Or an unknown interrupt not known by this program
   which also results in a fatal controller error.

! Check to see if we get here because of a response ring
! transition interrupt. This is more likely to be the
! most frequent interrupt so check it first.
!
if .HEAD_AREA [RSP_INT] then

begin

HEAD_AREA [RSP_INT] = ZEROS; ! Clear the interrupt indicator location
GEI_NRD();

!Get the resp slot location to process

if .RECEIVE_RING [.NRD_SLOT, OWN_BIT] nequ MOST_OWNED !Is this owned by host
then
begin
  $HOST = PSE_CODE;
  $DECODE();
  return;
end;

!Host/port is out of sequence
!Report a "Port sync error" code
!Just for show Decode kills it

Per DUP protocol once interrupted due to a response ring
interrupt, the host code should process all response packets
found in the response ring. This while loop will continue
to process the response packets in the response ring until
none remain.

while TRUE do
begin
  $BREAK;
  !Look for control c's
  Load the Reference structure "Ref_enfad" with the address
  of this response envelope to process (The minus #04' is
done to address the first word in the envelope packet
  and is equal to location "text-4").
  RET_ENFAD = (.RECEIVE_RING [.NRD_SLOT, LO_ENFAD]) - #04';

  Test the end packet for its possible three end types.
  End message opcodes (also called endcodes) are formed by adding the end
  message flag to the command opcode. For example, a READ commands end
  message contains the value OP.RED + OP.END in its opcode field. The Invalid
  command end message contains just the end message flag (i.e., OP.END) in
  its opcode field. The serious exception opcode shown above (i.e., OP.SEX +
  OP.END) in its opcode field.

  Commands opcode bits 6 and 7 indicate the type of message (command, end or
  attention message. Command opcodes bits 3 through 5 indicate the command
  category (immediate, sequential or no-sequential) and whether or not the
  command includes a buffer descriptor.

  See MSCP document appendix "A-1 NOTE:" for more information on this topic.
selectoneu .RET.ENHAD [TIPMSG] of set

[ENDMSG] :!Select the end packet size

begin

![Select off of the end code to make sure the communications mechanism transfers the correct number of bytes for this end packet. If this number of bytes transferred is not correct for the commands end packet then load the error code into return status, call decode to report the error and kick the bucket and die. This endcode is formed by adding the end message flag 01200 to the commands opcodes.

selectoneu .RET.ENHAD [ENCODE] of set

[RECEIVE DATA] command end packet

begin

[EOP.RED] :

if .RET.ENHAD [MSG.LENGTH] gtrw ESZ.RED !Is the byte count correct

begin

RET.STATUS = MLE.CODE; !Return a 'message length error code'

DECODE (); !Report the error and kick the bucket

return; !Just for show. Decode kills it

end;

![SEND DATA] command end packet

[SET CONTROLLER CHAR] command end packet (same opcode)

begin

[EOP.SED] :

[EOP.SCC] : !!!

begin

if (not (.RET.ENHAD [MSG.LENGTH] leqw ESZ.SED)) !Is the byte count correct

begin

RET.STATUS = MLE.CODE; !Return a 'message length error code'

DECODE (); !Report the error and kick the bucket

return; !Just for show. Decode kills it

end;
```
end;

"GET DUST STATUS" command end packet

[EDP.GDS]:
begin

; if RET.ENHAD (MSG.LENGTH) neq ESZ GDS !Is the byte count correct
then
begin
    RET.STATUS = MLE.CODE; !Return a "message length error code"
    DECODE (); !Report the error and kick the bucket
    return(); !Just for show. Decode kills it
end;

end;

@()

"EXECUTE SUPPLIED PROGRAM" command end packet

[EDP.ESP]:
begin

; if RET.ENHAD (MSG.LENGTH) neq ESZ.ESP !Is the byte count correct
then
begin
    RET.STATUS = MLE.CODE; !Return a "message length error code"
    DECODE (); !Report the error and kick the bucket
    return(); !Just for show. Decode kills it
end;

end;

@)

"EXECUTE LOCAL PROGRAM" command end packet

[EDP.ELP]:
begin

; if RET.ENHAD (MSG.LENGTH) neq ESZ.ELP !Is the byte count correct
then
begin
    RET.STATUS = MLE.CODE; !Return a "message length error code"
    DECODE (); !Report the error and kick the bucket
    return(); !Just for show. Decode kills it
end;

end;

@)

"ABORT PROGRAM" command end packet

```
(FOR ABT):

begin

if .RET ENHAD [MSG LENGTH] neq ESZ_AB1 
   !Is the byte count correct
then
begin

   RET_STATUS = MLE_CODE; 
   !Return a message length error code
   DECODE (); 
   !Report the error and kick the bucket
   return;
   !Just for show. Decode kills it
end;

end;

"ON LINE" command end packet.

begin

if .RET ENHAD [MSG_LENGTH] neq ESZ.ONL 
   !Is the byte count correct
then
begin

   RET_STATUS = MLE_CODE; 
   !Return a message length error code
   DECODE (); 
   !Report the error and kick the bucket
   return;
   !Just for show. Decode kills it
end;

end;

The "OP_END" end message flag all by itself tells
us that the controller is flagging us of an illegal
command sent over the connection. Error and kick the
bucket.

begin

   RET_STATUS = IVC_CODE; 
   DECODE (); 
   return;
end;

The controller is telling us that a serious exception
has occurred. Error and kick the bucket.

begin

   RET_STATUS = SEX_CODE; 
   DECODE (); 
   return;
end;
/* Unknown end packet endcode type */

/* otherwise */
begin
  RET_STATUS = UEC CODE;  /*Return an 'unknown end code*/
  DECODE ();  /*Report the error and kick the bucket*/
  return;  /*Just for show. Decode kills 't end;*/
end;

tes:

The port/controller sent the endpacket over the connection
without any problems. Now find this commands owner in the
outstanding buffer and indicate to them that the command
has been received.

REF_NUM = RET.ENTRYADOS [CMD_LREF];  /*Get this rec packets cmd ref number*/

Search the outstanding command buffer for this commands
reference number.
If found, load the buffer location with the ret entad
and set the received flag to signify that this command
has been received by this interrupt service routine.
FOUND_CMD = FALSE;  /*Clear the found cmd flag*/

incru = from 0 to REC_ALLOCATE - 1 do  /*Search the buffer*/
  if .OUTSTD_BUF [i, CMD_REF] eqv .REF NUM  /*Is this the cmd ref*/
    then
      begin
        OUTSTD_BUF [i, REC_FLG] = TRUE;  /*Indicate command is received*/
        OUTSTD_BUF [i, ENV,ADR] = RET.ENTRYADOS;  /*Return envelope adr's*/
        FOUND_CMD = TRUE;  /*Indicate it was found*/
        exitloop;  /*Exit the loop*/
      end;
      break;
  end;

if the search through the command ref
buffer failed to find this commands cmd
reference number then die.

if not .FOUND_CMD
  then
    begin
      RET_STATUS = PSE CODE;
      DECODE ();
      return;
    end;
end;

!End of ENDMSG processing

! The set controller characteristics command
! disabled the reporting of attention messages
! so treat this as a fatal error and die.
!
[ATTMSG] :

begin
  RET_STATUS = ATN_CODE;
  DECODE ();
  return;
end;

! It doesn't make any sense for this end message
! packet not to have the end message flag added
! to this command opcode as treat it as a fatal
! error and die.
!
[CMDMSG] :

begin
  RET_STATUS = CMD_CODE;
  DECODE ();
  return;
end;

! This end code type is of unknown origin so
! treat it as a fatal error and die.
!
[otherwise] :

begin
  RET_STATUS = UEC CODE; !Unknown message type code received
  DECODE ();
  return;
end;

test;

! Before we leave put this receive envelope message length
! field back to the envelope size, in bytes (Per UQ Spec).
! This size does not include the 2 UQ's words preceding the
! command text area.

+ RET_ENVAD (MSG_LENGTH) = RB SIZE+2;

! Return this receive slot descriptor back to
! the port to fulfill my part of the protocol.
!
RECEIVE_RING [.NAM SLOT, OWN BIT] = PORT OWNED;

! Look at the next response ring descriptor. If it's
! host owned then continue this process else exit the
! loop. First see if the ring reference has wrapped
! around to the top of the ring.

if `.NRD_SLOT + 1 gt_reduce REC_ALLOCATE 1
! Has the ring ref wrapped around then
! TEMP = ZERO
! Wrap it back to zero desc slot
else
! TEMP = `.NRD_SLOT + 1;
! Look at the next seq desc slot

! Now see if the next receive descriptor slot is
! host owned.

if `.RECEIVE_RING [TEMP, OWN_BIT] eq Reduce HOST_OWNED
! Are we done yet then
! GET_NRD ()
! Get the next resp desc to process
else
! exitloop;
! No more to do so exit
end;
! End of WHILE LOOP

! All response ring descriptors have been with out any
! detected errors so return to the main host code with
! an pass return code.

return RET_STATUS = PAS_CODE;
! Return a 'pass code
end;

!*****************************************************************************

!*****************************************************************************

! A send ring transition interrupt could happen if at
! some date only one descriptor slot is allocated for
! commands.

! Clear the interrupt indicator if this is true and do
! a return with no errors.

if `.HEAD_AREA [CMD_INT]
! Is this a com ring transition 'interrupt then
begin
HEAD_AREA [CMD_INT] = ZERO;
! Clear out the indicator
return;
! Continue on with the host code
end;

!*****************************************************************************

!*****************************************************************************
Check to see if an adaptor purge is being requested
by the port/controller in order to complete execution
of a issued command. Remember that this is illegal
during PDP-11 formatting and is considered to be a
fatal error.

if .HEAD_AREA (ADP.CH) neq ZERO
then
    begin
        RET_STATUS = APR CODE;
        DECODE ();
        return;
    end;

!**************************************************************************
** ENDOF ADAPTOR PURGE INTERRUPT PROCESSING ********************

The host program has been interrupted by an unknown interrupt
source if the routine program flow reaches here.

Load the error code into return status and call decode to take
appropriate action.

RET_STATUS = UIN_CODE;
DECODE ();
return;
end;

.SBTTL DUP$1.SERVICE GLOBAL ROUTINE DECLARATIONS
.DUP$1.SERVICE::

    MOV    R0, -(SP)
    MOV    R1, (SP)
    MOV    R2, (SP)
    MOV    R3, (SP)
    MOV    R4, (SP)
    MOV    R5, (SP)
    MOV    RDRX, ADDR, RO
    MOV    2(RO), (SP)
    *,RC$5,REG
    BPL    1$:
    MOV    #1, (RO)
    BIT    91, RO
    BNE    2$:
    JMP    26$:
    CLR    6(RO)
    JSR    RC, GET, NRD
    MOV    NRD.SLOT, RO
    ASL    RO
    ASL    RO
000622 000137 001306
000626 0005037 000000G 25h: JMP 4h
000632 000427 000000G BR 301
000654 0013700 000000G 26h: MOV HEAD,AREA,RO
000640 032760 000001 000004 BII 414(RO)
000646 001403 000004 BEQ 27h
000650 005060 000004 CLR 4(RO)
000654 000416 BR 301
000656 0013700 000000G 27h: MOV HEAD,AREA,RO
000662 105760 000003 TSTB 3(RO)
000666 001404 BEQ 28h
000670 012737 000201 000000G MOV #201,RET.STATUS
000676 000403 BR 29h
000670 012737 0000301 000000G 28h: MOV #301,RET.STATUS
000676 000403 JSR PC, DECODE
000712 005726 0000250 29h: TST (SP)
000714 012605 000000G MOV (SP), R5
000716 012604 MOV (SP), R4
000720 012603 MOV (SP) , R3
000722 012602 MOV (SP), R2
000724 012601 MOV (SP), R1
000726 012600 MOV (SP), R0
000730 000002 RTI

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

; 2704 1
global routine CTO_WAIT (TO_VALUE, REF_NUM, Buf4Loc) = !Controller time out wait

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.

Buf4Loc: This argument points to the outstd_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 outstd_buffer 'word zero of a command slot' is cleared out with the value 0x100000 to indicate this is an unused outstd_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 communications mechanism to complete the queued command. If the port/controller does not time out then this return status is returned as this routine's 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:
begin

! Before doing the timeout wait make sure that this buffer location
! that we're suppose to time out actually contains the command ref
! number that was sent to us via the formal argument. Error and
! kick the bucket if not the same.
!
if .OUT$STD_BUF [.BUFSIZE, CMD_REF] neq .REF_NUM !Is this the same ref_num
then
begin
  RET_STATUS = OSE_CODE; !Indicate the error code
  DECODE (); !Call decode to report the error
end;

! Loop on a one micro second delay for the number of times
! requested by the caller. After each delay see if the flag
! "rec_flg" has been set yet. Return "Ret_status" and clear the
! command word to $0'100000' to indicate this command has been
! received if this flag gets set before the timer expires.
!
incru: from 0 to .TO_VALUE do !Loop for time-out_value
begin
  DELAY (C_USE); !Do the one micro second delay
  ! Exit routine with the DSP interrupt service
  ! routines "ret_status" if "rec_flg" got set before
  ! the timer expires.
  !
end;

! If .OUT$STD_BUF [.BUFSIZE, REC_FLG] eqv TRUE !Is this command received yet
then
begin
  OUT$STD_BUF [.BUFSIZE, CMD_WRD] = $0'100000'; !Return the slot to the unused state
  return .RET_STATUS; !Return the interrupt service status
end;

BREAK; !Service any control C's
end;

! The port/controller timed out if the code
! reached here. Return an error code to
! the caller and exit the routine.

if .DRDX_ADDR (.RCSA, ERR_BIT)
 then return RET_STATUS = PFE_CODE
 else return RET_STATUS = CTO_CODE;
end;

Is the 9A error bit set
Port timed out with fatal error
Port just timed out

.GLOBL L$CPU, L$DLY

.CTO.WAIT GLOBAL ROUTINE DECLARATIONS

000000 004137 000000G

.CTO.WAIT:

JSR R1,$SAVE3

CMP -(SP),-(SP)

MOV 16(SP),R0 ; BUF$LOC,

ASL R0

ASL R0

MOV #OUT$STD$BUF,R3

ADD R0,R3

CLR R0

BISB (R3),R0

CMP R0,20(SP) ; *,REF.NUM

BEQ 1$

1$: CLR R2

I

2$: MOV L$CPU,R1 ; *,##%TMP2

3$: BEQ 6$

5$: BNE 4$

4$: CLR 2(SP) ; $MP

R0

DEC R0 ; ##%TMP1

5$: DEC R1

3$

6$: TST (R3) ;

7$: BPL 7$

11$

10$

11$

7$: TRAP 22

I

8$: CMP R2,22(SP) ; I,T0.VALUE

9$: BPL 2$

2$

10$

11$

21$

10$

11$

21$

2800

2811

2813

2815
; Routine Size: 60 words.
; Routine Base: $CODE$ = 2116
; Maximum stack depth per invocation: 8 words
global routine ABORT  

!Aborts remote program

!...

!Functional Description:

2623 The abort program command is used to terminate the execution of a remote program in an orderly fashion. When a successful response is received to this command, the remote program has stopped executing and the server is in idle state. Note that the sending of this command does not preclude further send data or receive data exchanges: On the contrary, the remote program may be designed to send out termination status and possibly even ask questions during its forced-exit sequence. The time out for this command is a fixed 10 seconds and if a response is not received by then the connection to the dust should be terminated. This command is only legal if the dust is in active state.

!Formal Parameters:

!none

!Implicit Inputs:

NSD_SLOT

This global storage gets loaded by the routine 'Get_NS' and in it is stored the next send ring descriptor slot where the port/controller should be polling on and the place to put this command's command packet.

!Implicit Outputs:

!none

!Completion Codes:

RET_STATUS:  Return status passes back to the calling routine the status of the just issued command.

Side Effects:

!Any remote program running in the controllers DM machine will be aborted.

begin

local

REF_NUM,  
ABO_BUFLOC,  
TEMP

!Stores unique cmd ref number
!Stores outstanding cmd buffer location
!A place to put the read IP register data

!Before we load up the command packet up with all this good information get the next send descriptor slot and a unique command reference number.

GET_NS()  
REF_NUM = GET_CMD1REF()  

!Get the next send desc slot
!Get a unique command ref num
!! UG Port command envelope Header field definition
!!
!! SND.ENVELOPE [.NSD_SLOT, MSG.LENGTH] = SZ ABT ; !Load the length of envelope
!! SND.ENVELOPE [.NSD_SLOT, CREDITS] = ONE; !Load credits
!! SND.ENVELOPE [.NSD_SLOT, MSG.TYPE] = 0; !Load message type
!! SND.ENVELOPE [.NSD_SLOT, CONN.ID] = 2; !Load connection ID
!!
!! DUP command envelope field definition
!!
!! SND.ENVELOPE [.NSD_SLOT, CMD.LREF] = .REF.NUM; !Define reference number
!! SND.ENVELOPE [.NSD_SLOT, CMD.HREF] = ZERO; !Hi order ref number
!! SND.ENVELOPE [.NSD_SLOT, UN.LUSED] = ZERO; !Unused low order
!! SND.ENVELOPE [.NSD_SLOT, UN.HUSED] = ZERO; !Unused hi order
!! SND.ENVELOPE [.NSD_SLOT, OPCODE] = OP_AB; !Load opcode
!! SND.ENVELOPE [.NSD_SLOT, RSVD] = ZERO; !Reserved field
!! SND.ENVELOPE [.NSD_SLOT, MODIFIER] = ZERO;
!!
!! Call the load outstanding command buffer routine
!! and load this command into the buffer. The return
!! from this routine will point us to the buffer location
!! where this command is stored. Later we can look at
!! this location to see if the interrupt service routine
!! has received and processed it.
!!
!! ABO_BUF#LOC = LOAD_OUTSTD_BUF (.REF.NUM); !Load the command
!!
!! if .ABO_BUF#LOC equl OBF.CODE then DECODE (); !Error if buffer is full
!!
!! Set the ownership bit to 1 giving this slot
!! to the port/controller
!!
!! SEND_RING [.NSD_SLOT, OWN_BIT] = PORT OWNED;
!!
!! Read the IP register to stimulate port polling
!!
!! TEMP = .RDRX_ADDR [RCIP, RC_ALL];
!!
!! Time out the port/controller processing the command.
!!
!! The first test tests the connections ability to
!! respond to this command without any errors in the 5A
!! register and for the command not timing out.
!!
!! The second tests the DUP server for good status. If
!! bad status is sent back then an error code is returned
!! to the calling routine where the routine "decode" will
!! decode and take the appropriate recovery. The time
!! out routine will loop on delaying and checking the hi
!! bit of the first word in the outstd_buf for a true.
!!
!! When true signals us that the interrupt service routine
!! has received the end packet and no connection errors
were detected.

if CTO_WAIT (3000, .REF_NUM, .ABO_BUF$LOC) then DECODE () ! Is return an error

Get the return envelope address from the out#std$buf
at this command buffer location and check the packet
for good status error and die if bad status was returned

RET_EN#AD = .OUT#STD$BUF [.ABO_BUF$LOC, ENV ADR]; ! Get the ret env adr

Now test for good status

if .RET_EN#AD [STATUS] ne#q ZERO
   test the status
else
   return RET_STATUS # RSE_CODE ! Return a "Response status err" code
   return .RET_STATUS; ! This ret_status is good or bad

end:

000000 004137 000000G
000004 005746
000006 004737 000000G
000012 004737 000146
000016 000002
000020 003746 000000G
000024 001276 000054
000030 004737 000000G
000034 012760 000014 000000G
000042 012701 000002G
000046 060001
000050 112711 000001
000054 112761 000002 000001
000062 010260 000004G
000066 005060 000006G
000072 005060 000010G
000076 005060 000012G
000102 112760 000006 000014G
000110 105060 000015G
000114 005060 000016G
000120 010216
000122 004737 000054G
000126 010001
000130 020127 020001
000134 001002
000136 004737 000250G
000142 013700 000000G
000146 006300

ABORT: JSR R1, $SAVE2

TST (SP)

JSR PC, GET, NSF

JSR PC, GET, CMD$REF

MOV R0, R2

MOV NSF.SLOT,-(SP)

MOV $54,-(SP)

JSR PC, BL$MUL

MOV #14, SND.ENVELOPE(R0)

MOV #SND.ENVELOPE+2,R1

ADD R0, R1

MOVB #1,(R1)

MOVB #2,(R1)

MOV R2, SND.ENVELOPE+4(R0) ; REF_NUM,

CLR SND.ENVELOPE+6(R0)

CLR SND.ENVELOPE+10(R0)

CLR SND.ENVELOPE+12(R0) ; REF_NUM,

CLR SND.ENVELOPE+14(R0)

MOV #6, SND.ENVELOPE+14(R0)

CLR SND.ENVELOPE+15(R0)

CLR SND.ENVELOPE+16(R0)

MOV R2,(SP) ; REF_NUM,

JSR PC, LOAD, OUT#STD$BUF

MOV R0, R1 ; * AB$ Buf LOC

CMP R1, $2001

BNE 1

JSR PC, DECODE

MOV NSF.SLOT, R0
**global routine GET_DUST_STATUS**

*Gets DUPS server status*

**Functional Description:**

This command allows the host program to interrogate the DUPS server to determine its characteristics, its state and the state of the program currently running (if any). It is legal in either idle or active state and does not affect the state of server. It has a fixed timeout interval of 3 seconds. If the response times out, the host should break the connection.

**Formal Parameters:**

- None

**Implicit Inputs:**

- NSD_SLOT
  
  This global storage gets loaded by the routine 'Get.Nsd' and in it is stored the next send ring descriptor slot where the port/controller should be polling on and the place to put this command packet.

**Implicit Outputs:**

- None

**Completion Codes:**

- RET_STATUS: Return status passes back to the calling routine the status of the just issued command.

**Side Effects:**

```c
begin

local
  REF_NUM,          ! Stores unique cmd ref number
  CDS_BUFLOC,      ! Stores outstanding cmd buffer location
  TEMP;            ! A place to put the IP read data

! Before we load up the command packet up
! with all this good information get the
! next send descriptor slot and a unique
! command reference number.

GET_NSD ();        ! Get the next send desc slot
REF_NUM = GET_CMDREF ();  ! Get a unique command ref num

! UQ Port command envelope header field definition

SND_ENVELOPE [.NSD_SLOT, MSG_LENGTH] = SZ_GDS;       ! Load the envelope size
SND_ENVELOPE [.NSD_SLOT, CREDITS] = ONE;             ! Load the credit size
SND_ENVELOPE [.NSD_SLOT, MSG_TYPE] = 0;              ! Load the message type (Sequential)
SND_ENVELOPE [.NSD_SLOT, CONN_ID] = 0;               ! Load the connection ID (DUP)
```
; 3001  2 ; DUP generic command envelope field definition
; 3002  2 ;
; 3003  2 ; SND_ENVELOPE [.NSD_SLOT, CMD_LREF] = .REF_NUM; !Load command reference number
; 3004  2 ; SND_ENVELOPE [.NSD_SLOT, CMD_HREF] = ZERO; !Command reference low order
; 3005  2 ; SND_ENVELOPE [.NSD_SLOT, UN_LUSED] = ZERO; !Low order unused
; 3006  2 ; SND_ENVELOPE [.NSD_SLOT, UN_HUSED] = ZERO; !Hi order unused
; 3007  2 ; SND_ENVELOPE [.NSD_SLOT, OPCODE] = OP_GDS; !Load opcode
; 3008  2 ; SND_ENVELOPE [.NSD_SLOT, RSV0] = ZERO; !Reserved field
; 3009  2 ; SND_ENVELOPE [.NSD_SLOT, MODIFIER] = ZERO; !Load modifier field
; 3010  2 ;
; 3011  2 ; Call the load outstanding command buffer routine
; 3012  2 ; and load this command into the buffer. The return
; 3013  2 ; from this routine will point us to the buffer location
; 3014  2 ; where this command is stored. Later we can look at
; 3015  2 ; this location to see if the interrupt service routine
; 3016  2 ; has received and processed it.
; 3017  2 ;
; 3018  2 ; GDS_BUFLOC = LOAD_OUTSTD_BUF (.REF_NUM); !Load the command
; 3019  2 ;
; 3020  2 ; if .GDS_BUFLOC equ 0BF_CODE then DECODE (); !Error if buffer is full
; 3021  2 ;
; 3022  2 ;
; 3023  2 ; set the ownership bit to 1 giving this slot
; 3024  2 ; to the port/controller
; 3025  2 ;
; 3026  2 ; SEND_RING (.NSD_SLOT, OWN_BIT) = PORT_OWNED;
; 3027  2 ;
; 3028  2 ; Read the IP register to stimulate port polling
; 3029  2 ;
; 3030  2 ; TEMP = .ADRX_ADDR (RCIP, RC_ALL);
; 3031  2 ;
; 3032  2 ; Time out the port/controller processing the command.
; 3033  2 ;
; 3034  2 ; The first test tests the connections ability to
; 3035  2 ; respond to this command without any errors in the SA
; 3036  2 ; register and for the command not timing out.
; 3037  2 ;
; 3038  2 ; The second tests the DUP server for good status. If
; 3039  2 ; bad status is sent back then an error code is returned
; 3040  2 ; to the calling routine where the routine "decode" will
; 3041  2 ; decode and take the appropriate recovery. The time
; 3042  2 ; out routine will loop on delaying and checking the hi
; 3043  2 ; bit of the first word in the out_bitd_buf for a true.
; 3044  2 ; when true signals us that the interrupt service routine
; 3045  2 ; has received the end packet and no connection errors
; 3046  2 ; were detected.
; 3047  2 ;
; 3048  2 ;
; 3049  2 ; if CTO WAIT ('3000', .REF_NUM, .GDS_BUFLOC) then DECODE (); !Is return an error
; 3050  2 ;
; 3051  2 ;
; 3052  2 ;
; 3053  2 ; Get the return envelope address from the out_bitd_buf
REIAD  * .OUTSTD BUF (.GDS_BUF$LOC, ENV ADR); !Get the ret env adr
  3058 2
  3059 2 ! Now test for good status
  3060 2
  3061 2 :if .REIAD [STATUS] nequ ZERO  !Test the status
  3062 2 then
  3063 2  return RET_STATUS = RSE_CODE  !Return a "Response status err" code
  3064 2 else
  3065 2  return .REIAD STATUS;  !This ret status is good or bad
  3066 2
  3067 2 end:

  00000 004137 000000G
  00004 005746
  00006 004737 000000'!
  00012 004737 000146'!
  00016 010002
  00020 013746 000000G
  00022 012746 000054
  00030 004737 000000G
  00034 012760 000014 000000G
  00042 012701 000002G
  00046 060001
  00050 112711 000001
  00054 112761 000002 000001
  00062 010260 000004G
  00066 005060 000006G
  00072 005060 000106G
  00076 005060 001106G
  00084 112760 000001 00014G
  00092 012760 000015G
  00096 112760 000016G
  00104 010216
  00122 004737 000054'!
  00126 001001
  00130 020127 002001
  00134 001002
  00136 004737 000250'
  00142 013700 000000G
  00146 060300
  00150 060300
  00152 063700 000000G
  00156 052760 100000 000002
  00164 017766 000000G 000004
  00172 016600 000004
  00176 012716 003000
global routine EX SUP_PROG  !Executes supplied program

! Functional Description:
This command causes the server to transfer the program from host
memory to an area in the controller and start its execution. The
host supplies the address and length (in bytes) of a buffer
containing the program header and initial load; the starting
address of the program, its memory requirements and any relocation information
needed to run under the server are in the program header in a format
which is none of the host business. This command is only legal when
the server is in the idle state and return of a successful end packet
puts the server into active state.

The time out interval for this command is 30 seconds.

! Formal Parameters:
none

! Implicit Inputs:
NSD_SLOT
This global storage gets loaded by the routine
'Get_nsd' and in it is stored the next send ring
descriptor slot where the port/controller should be polling on and the place to put this command
when no packet.

! Implicit Outputs:
AZFMT: Azfmt is the vector produced by DMCONV program and
is declared in module AZKEL6.
DMSA: These three bound addresses point to specific area
in the DM code buffer 'azftr' and are used to
define the buffer descriptors within this command.

! Completion Codes:
RET_STATUS: Return status passes back to the calling routine
the status of the just issued command.

! Side Effects:
The DM machine in the controller goes from the idle state to the
active state on return of a successful return packet.

begin
local
REF_NUM, !Stores unique cmd ref number
ESP_BUFLOC, !Stores outstanding cmd buffer location
TEMP; !A place to put the read IP register data

! Before we load up the command packet up
; C 3123 1 ! with all this good information get the 1
; next send descriptor slot and a unique 1
; command reference number. 1
; 1
; GET_NSD (); 1 !Get the next send desc slot 1
; REF_NUM = GET_CMD1REF (); 1 !Get a unique command ref num 1
; 1
; UO Port command envelope Header field definition 1
; 1
; SND_ENVELOPE [ .NSD_SLOT, MSG_LENGTH] = SZ_ESP; 1 !Load the message length 1
; SND_ENVELOPE [ .NSD SLOT, CREDITS ] = ONE; 1 !Load the credits field 1
; SND_ENVELOPE [ .NSD SLOT, MSG_TYPE ] = 0; 1 !Define the msg type 'Sequential' 1
; SND_ENVELOPE [ .NSD SLOT, CONN_ID ] = 2; 1 !Define the connection ID as DUP 1
; 1
; DUP generic command envelope field definition 1
; 1
; SND_ENVELOPE [ .NSD SLOT, CMD_LREF ] = .REF_NUM; 1 !Load command ref number 1
; SND_ENVELOPE [ .NSD SLOT, CMD_HREF ] = ZERO; 1 !Zero Hi order word of cmd ref 1
; SND_ENVELOPE [ .NSD SLOT, UN_LUSED ] = ZERO; 1 !Not used in DUP implementation 1
; SND_ENVELOPE [ .NSD SLOT, UN_HUSED ] = ZERO; 1 !Not used in DUP implementation 1
; SND_ENVELOPE [ .NSD SLOT, OPCODE ] = OP ESP; 1 !Load the command op-code 1
; SND_ENVELOPE [ .NSD SLOT, RSVD ] = ZERO; 1 !Not used 1
; SND_ENVELOPE [ .NSD SLOT, MODIFIER ] = ZERO; 1
; 1
; Command specific command envelope field definition 1
; 1
; Byte count of initial transfer (from bytes 0-3 1
; of the program header). 1
; 1
; SND_ENVELOPE [ .NSD SLOT, BLO_CNT ] = .AZFMTR [ WRDO ]; 1 !Byte count low word 1
; SND_ENVELOPE [ .NSD SLOT, BHI_CNT ] = .AZFMTR [ WRDI ]; 1 !Byte count high word 1
; 1
; Buffer descriptor definition for initial load. First 1
; byte of this buffer is byte 0 of program header. 1
; 1
; SND_ENVELOPE [ .NSD SLOT, BPA_L ] = HDSA; 1 !Low unibus adrs <0-15> 1
; SND_ENVELOPE [ .NSD SLOT, BPA_HI ] = ZERO; 1 !Unibus adrs bits <16-17> 1
; SND_ENVELOPE [ .NSD SLOT, QBUS_EXT ] = ZERO; 1 !Qbus extension adrs 1
; SND_ENVELOPE [ .NSD SLOT, RSV1 ] = ZERO; 1 !Reserved field 1
; SND_ENVELOPE [ .NSD SLOT, UBA_CHAN ] = ZERO; 1 !Unibus adaptor channel number 1
; SND_ENVELOPE [ .NSD SLOT, RSV0 ] = ZERO; 1 !These next four words are not 1
; SND_ENVELOPE [ .NSD SLOT, RSV1 ] = ZERO; 1 !used in the DUP implementation 1
; SND_ENVELOPE [ .NSD SLOT, RSV2 ] = ZERO; 1
; SND_ENVELOPE [ .NSD SLOT, RSV3 ] = ZERO; 1
; 1
; These next field definitions are the same 1
; as above except they are for the overlay 1
; buffer descriptors. To make life easy for 1
; me I'll use the same names and just prefix 1
; them with a 1 for uniqueness. 1
; 1
; The overlay area immediately follows the
! Initial load image in the program image.

SND_ENVELOPE [.NSD_SLOT, #BPA_LO] = OVSA;  // Low unibus addr <0-15>
SND_ENVELOPE [.NSD_SLOT, #BPA_HI] = ZERO;  // Unibus addr bits (16-17)
SND_ENVELOPE [.NSD_SLOT, #BUS_EXT] = ZERO;  // Unibus extension addr
SND_ENVELOPE [.NSD_SLOT, #RSV1] = ZERO;  // Reserved field
SND_ENVELOPE [.NSD_SLOT, #RSV2] = ZERO;  // Unibus adaptor channel number
SND_ENVELOPE [.NSD_SLOT, #RSV3] = ZERO;  // These next four words are not
SND_ENVELOPE [.NSD_SLOT, #RSV4] = ZERO;  // used in the DUP implementation
SND_ENVELOPE [.NSD_SLOT, #RSV5] = ZERO;  //
SND_ENVELOPE [.NSD_SLOT, #RSV6] = ZERO;  //

! Call the load outstanding command buffer routine
! and load this command into the buffer. The return
! from this routine will point us to the buffer location
! where this command is stored. Later we can look at
! this location to see if the interrupt service routine
! has received and processed it.

ESP_BUF#LOC = LOAD_OUT#STD_BUF (.REF_NUM);  // Load the command

! 1f .ESP_BUF#LOC equl OBF_CODE then DECODE ();  // Error if buffer is full

! Set the ownership bit to 1 giving this slot
! to the port/controller
SEND_RING [.NSD_SLOT, OWN_BIT] = PORT_OWNED;

! Read the IP register to stimulate port polling
TEMP = .RDRV_ADDR [RCIP, RC_ALL];

! Time out the port/controller processing the command.

! The first test tests the connections ability to
! respond to this command without any errors in the SA
! register and for the command not timing out.

! The second tests the DUP server for good status. If
! bad status is sent back then an error code is returned
! to the calling routine where the routine "decode" will
! decode and take the appropriate recovery. The time
! out routine will loop on delaying and checking the hi
! bit of the 1st word in the out#std#buf for a true.

! When true signals us that the interrupt service routine
! has received the end packet and no connexion errors
! were detected.

! If CTO_WAIT (#o'3000', .REF_NUM, .ESP_BUF#LOC) then DECODE ();  // Is return an error

!
/*
 * Get the return envelope address from the out_stdBuf
 * Get this command's buffer location and check the packet
 * for good status error and die if bad status was returned
 *
RET_EN$AD = .OUT$STD_BUF [.ESP BUF$LOC, ENVADR]; !Get the ret env adr

! Now test for good status

if .RET_EN$AD [STATUS] nequ ZERO
then
  return RET_STATUS = RSE CODE
else
  return .RET_STATUS;

end;
*/
global routine EX LOC, PROC - !Executes local program

... Functional Description:
Receipt of this command causes the controller to search its local
and/or the next send descriptor slot where the port/controller should
be cycling on and the place to put this command command.

Formal Parameters:
none

Implicit Inputs:
NSD, SLOT This global storage gets loaded by the routine

GET_NS D and in it is stored the next send ring
descriptor slot where the port/controller should
be cycling on and the place to put this command.

Implicit Outputs:
none

Completion Codes:
RET, STATUS: Return status passes back to the calling routine
the status of the just issued command.

Side Effects:
The DM machine in the controller goes from the idle state to the
active state on return of a successful return packet.

begin
local
REF, NUM, !Stores unique cmd ref number
ELP, BUF LOC, !Stores outstanding cmd buffer location
TEMP; !A place to store the read IP register data

Before we load up the command packet up
with all this good information get the
next send descriptor slot and a unique
command reference number.

GET_NS ( );
REF, NUM = GET_CMD, REF ( ) ; !Get the next send desc slot

UQ Port command envelope Header field definition

SND, ENVELOPE [ NSD, SLOT, MSG, LENGTH ] = S7 ELP; !Load the message size
SND, ENVELOPE [ NSD, SLOT, CREDITS ] = ONE; !Load the credit size
SND, ENVELOPE [ NSD, SLOT, MSG, TYPE ] = 0; !Define the msg typ 'Sequential'
SDN_ENVELOPE [.NSD_SLOT, CMD [REF] = .REF NUM]; !User the command ref num
SDN_ENVELOPE [.NSD_SLOT, CMD [HREF] = ZERO; !Zero the Hi order cmd ref num
SDN_ENVELOPE [.NSD_SLOT, UN LUSED] = ZERO; !Not used in DUP implementation
SDN_ENVELOPE [.NSD_SLOT, UN MOD] = ZERO; !Not used in DUP implementation
SDN_ENVELOPE [.NSD_SLOT, OPCODE] = OP_ELP; !Load this commands op-code
SDN_ENVELOPE [.NSD_SLOT, RSVD] = ZERO; !Not used field
SDN_ENVELOPE [.NSD_SLOT, MODIFIER] = DUP STND; !Define modifiers for this cmd

Command specific command envelope field definition

SDN_ENVELOPE [.NSD_SLOT, PN 0] = 'FO'; !Program name word 0
SDN_ENVELOPE [.NSD_SLOT, PN 1] = 'RP'; !Program name word 1
SDN_ENVELOPE [.NSD_SLOT, PN 2] = 'AT'; !Program name word 2

Call the load outstanding command buffer routine
and load this command into the buffer. The return
from this routine will point us to the buffer location
where this command is stored. Later we can look at
this location to see if the interrupt service routine
has received and process it.

ELP_BUFLOC = LOAD_OUTSTND_BUF (.REF NUM); !Load the command

if ELP_BUFLOC equ OBF CODE then DECODE (); !Error if buffer is full

Set the ownership bit to 1 giving this slot
to the port/controller

SEND_RING [.NSD_SLOT, OWN BIT] = PORT OWNED;

Read the IP register to stimulate port polling

TEMP = RDRX ADDR [RCIP, RC ALL];

Time out the port/controller processing the command.

The first test tests the connection ability to
respond to this command without any errors in the SA
register and for the command not timing out.

The second tests the DUP server for good status. If
bad status is sent back then an error code is returned
to the calling routine where the routine "decode" will
decode and take the appropriate recovery. The time
out routine will loop on delaying and checking the hi
bit of the first word in the outstd.buf for a true.
When true signals us that the interrupt service routine
has received the end packet and no connection errors
were detected.

if CTO_WAIT (3000, .REF_NUM, .ELP_BUF+LOC) then DECODE (); !Is return an error

! Get the return envelope address from the outSTD_buf

let this commands buffer location and check the packet

for good status error and die if bad status was returned

RET.ENHAD = .OUTSTD_BUF [.ELP_BUF+LOC, ENV ADR]; !Get the ret env adr

! Now test for good status

if .RET.ENHAD [STATUS] neaq ZERO then

! Test the status

return RET.STATUSS + RSE_CODE

else

return .RET.STATUS;

!This ret status is good or bad

end;

SBTITL EX, LOC, PROG GLOBAL ROUTINE DECLARATIONS

EX, LOC, PROG:

JSR R1, @SAVE2

TST .(SP)

JSR PC, GET, NSD

JSR PC, GET, CMD+REF

MOV RO, R2

MOV NSD, SLOT, -(SP)

MOV @54, -(SP)

JSR PC, BLIPMUL

MOV #60, SND, ENVELOPE(R0)

MOV #50, SND, ENVELOPE+2, R1

ADD RO, R1

MOV #1, R1

MOV #2, R1

MOV #2, SND, ENVELOPE+4(R0)

REL, NUM, *

CLR SND, ENVELOPE+6(R0)

CLR SND, ENVELOPE+10(R0)

CLR SND, ENVELOPE+12(R0)

CLR #3, SND, ENVELOPE+14(R0)

CLR #3, SND, ENVELOPE+15(R0)

MOV #1, SND, ENVELOPE+16(R0)

MOV #4, SND, ENVELOPE+20(R0)

MOV #4, SND, ENVELOPE+22(R0)

MOV #52101, SND, ENVELOPE+24(R0)

MOV R2, -(SP)

REL, NUM, *

JSR PC, LOAD, OUTSTD_BUF

MOV RO, R1

CNP R1, #2001

REL, #32, BUF+LOC

REL, #32, BUF+LOC,*
global routine SEND_DATA (mslen)  

Functional Description:
These commands are used to communicate between the initiating host program and the remote program. Both send and receive commands specify a send buffer descriptor and a byte count. In the case of send data, the information in the buffer is read by the remote program and a send data response sent back to the host to acknowledge receipt. In the case of receive data, the remote program writes data into the buffer up to the amount specified by the byte count and then sends a receive data response to the host to notify it of the transmission.

The send data and receive data commands are only legal when the server is in the active state. If the remote program terminates abnormally, putting the server back in the idle state, outstanding send data and receive data commands may be lost. In the event that the specified timeout interval is exceeded, the host program should issue a get dust status command to see if the remote program is still running (i.e., the dup server is active); if it is, the progress indicator should be remembered and the timeout interval should be re-installed. If the second timeout expires without a response and a second get dust status shows the remote program having made no progress, in the interim then the program should be considered broken and should be aborted.

Formal Parameters:
none

Implicit Inputs:
NSD_SLOT
This global storage gets loaded by the routine 'Alloc' and in it is stored the next send ring descriptor slot where the port/controller should be polling on and the place to put this commands command packet.

Implicit Outputs:
none

Completion Codes:
RET_STATUS: Return status passes back to the calling routine the status of the just issued command.

Side Effects:
none

begin
local
REF NUM,
SND_BUFFLOC,
TEMP;

!Stores unique cmd ref number
!Stores outstanding cmd buffer location
!A place to put read IP register data
Before we load up the command packet up
with all this good information get the
next send descriptor slot and a unique
command reference number.

GET_NS2();  //Get the next send desc slot
REF_NUM = GET_CMD1REF();  //Get a unique command ref num

UQ Port command envelope Header field definition

SND_ENVELOPE [.NSD.SLOT, MSG.LENGTH] = 52 SED;  //Load the message size
SND_ENVELOPE [.NSD.SLOT, CREDITS] = ONE;  //Load the credit size
SND_ENVELOPE [.NSD.SLOT, MSG_TYPE] = 0;  //Define the message type 'Sequential'
SND_ENVELOPE [.NSD.SLOT, CONN_ID] = 2;  //Define the connection ID 'DUP'

DUP generic command envelope field definition

SND_ENVELOPE [.NSD.SLOT, CMD_LREF] = .REF NUM;  //Load command reference number
SND_ENVELOPE [.NSD.SLOT, CMD_HREF] = ZERO;  //Zero Hi order cmd ref number
SND_ENVELOPE [.NSD.SLOT, UN_LUSED] = ZERO;  //Not used in DUP implementation
SND_ENVELOPE [.NSD.SLOT, UN_HUSED] = ZERO;  //Not used in DUP implementation
SND_ENVELOPE [.NSD.SLOT, OPCODE] = OP SED;  //Load this command's op code
SND_ENVELOPE [.NSD.SLOT, RSV0] = ZERO;  //Not used field
SND_ENVELOPE [.NSD.SLOT, MODIFIER] = ZERO;  //Define the command modifiers

Command specific command envelope field definition

Byte count of transfer

SND_ENVELOPE [.NSD.SLOT, BLO_CNT] = .mlen;  //Byte count low word
SND_ENVELOPE [.NSD.SLOT, BMI_CNT] = ZERO;  //Byte count high word

Buffer descriptor definition

SND_ENVELOPE [.NSD.SLOT, BPA.LO] = SND_BUF1;  //Buffer physical addr <0-15>
SND_ENVELOPE [.NSD.SLOT, BPA.HI] = ZERO;  //Buffer physical addr bits <16-17>
SND_ENVELOPE [.NSD.SLOT, OBUS_EXT] = ZERO;  //O.Bus extension addr
SND_ENVELOPE [.NSD.SLOT, RSV1] = ZERO;  //Reserved field
SND_ENVELOPE [.NSD.SLOT, UBA_CHAN] = ZERO;  //Unibus adaptor channel number
SND_ENVELOPE [.NSD.SLOT, RSV2] = ZERO;  //These next four words are not
SND_ENVELOPE [.NSD.SLOT, RSV3] = ZERO;  //used in the UQ Port implementation

Call the load outstanding command buffer routine
and load this command into the buffer. The return
from this routine will point us to the buffer location
where this command is stored. Later we can look at
this location to see if the interrupt service routine
has received and process it.
SNDBUFFLOC = LOAD_OUTBUF BUF (REF_NUM); ; Load the command

if .SNDBUFFLOC ealu OBFS_CODE then DECODE (); ; Error if buffer is full

; Set the ownership bit to 1 giving this slot
to the port/controller
SEND_RING [.NSD_SLOT, OWN_BIT] = PORT_OWNED;
; Read the IP register to stimulate port polling
TEMP = .RDRX_ADDR [RCIP, RC_ALL];

; Time out the port/controller for the response from this
command.

If the controller times out then:

1. See what kind of error was returned. If the error
is a type other than a CTO_CODE (controller time out)
then call routine Decode which does the appropriate
action based on the error.

2. If the returned error is an CTO_CODE then do a get
status and check the progress indicator to look
for an increase, indicating that the remote program is
still running and is not dead.

If the indicator hasn't changed then assume that the
remote program is dead and return an error code of
RPD_CODE (remote program dead code) and exit.

If the indicator has changed then assume that the
remote program is still running, save a copy of its
value and reinstate the controller time out delay and
repeat the loop.

As long as the progress indicator in the remote program
is still increasing this loop will be repeated for ever.

If the controller doesn't time then return with the return
code returned from routine CTO_WAIT () which could be either
a success or error code by definition of this host code.

while TRUE do ; Repeat for ever
begin
break; ; Flag control C's

! Do a controller time out and determine if the controller
! has processed the command or if a fatal error has occurred.
! 3534 3
! 3535 3
! 3536 3
! 3537 3
! 3538 4
! 3539 4
! 3540 4
! 3541 4
! 3542 4
! 3543 4
! 3544 4
! 3545 4
! 3546 4
! 3547 4
! 3548 4
! 3549 5
! 3550 5
! 3551 5
! 3552 5
! 3553 5
! 3554 5
! 3555 5
! 3556 5
! 3557 5
! 3558 5
! 3559 5
! 3560 5
! 3561 5
! 3562 5
! 3563 5
! 3564 5
! 3565 5
! 3566 5
! 3567 5
! 3568 5
! 3569 5
! 3570 4
! 3571 4
! 3572 4
! 3573 3
! 3574 4
! 3575 4
! 3576 4
! 3577 4
! 3578 4
! 3579 4
! 3580 4
! 3581 4
! 3582 4
! 3583 4
! 3584 4
! 3585 4
! 3586 4

if CTO_WANT (3000, &REF_NUM, &SND_BUFLOC) then
  begin
    if the return status code eqs a CTO_CODE
    then see if the remote program is still
      running. If it is then save the progress
      indicator and repeat the loop else call
      routine Decode ()
    !
    if &RET_STATUS eqs CTO_CODE
    then
      begin
        REF_NUM = &REF_NUM;
        if GET_DUST STATUS () then DECODE ()
        !Get the dust status
        if &RET_ENCINAD [PLO_IND] gt0 &PID_SAVE
          then
            PID_SAVE = &RET_ENCINAD [PLO_IND]
            !Still running save Pid
          else
            return RET_STATUS = &RPD_CODE
            !No progress so flag error
        end
      else
        The return status code was not a controller time
        out code so something else is wrong. Call the
        routine Decode () to find out what went wrong.
        DECODE ()
    end
  end
else
  begin
    ! The command has been received by the interrupt service.
    Get this commands return envelope address out of the
    out std buf and check for good return status error and
    die if bad status.
    RET_ENCINAD = &OUT1STD_BUF [.SND_BUFLOC, ENV_ADR];
    !Get the ret env addr
    ! Test for good status
.GLOBAL ROUTINE DECLARATIONS

if .RET.EN#AD [STATUS] neq ZERO
then
    return .RET.STATUS * RSE CODE
else
    return .RET.STATUS;
end;

! Test the status
! Return a "Response status err" code
! This res_status is good or bad

! It won't compile without this here

SBTTL SEND.DATA GLOBAL ROUTINE DECLARATIONS

JSL R1, ISSUE2
JSR PC, GET.NSD
JSR PC, GET.CMD#REF
MOV RO, R2
MOV NSD.SLOT, -(SP)
MOV *54, -(SP)
JSR PC, BL #MUL
MOV *54, SND.ENVELOPE(RO)
MOV *54, SND.ENVELOPE-2, R1
ADD RO, R1
MOV *01, (R1)
MOV *02, (R1)
MOV R2, SND.ENVELOPE-4(RO)
CLR SND.ENVELOPE-6(RO)
CLR SND.ENVELOPE-10(RO)
CLR SND.ENVELOPE-12(RO)
MOV *4, SND.ENVELOPE-14(RO)
CLRB SND.ENVELOPE-15(RO)
CLRB SND.ENVELOPE-16(RO)
MOV 16(SP), SND.ENVELOPE-20(RO)
CLR SND.ENVELOPE-22(RO)
MOV *54, SND.ENVELOPE-24(RO)
MOV *54, SND.ENVELOPE-26, R1
ADD RO, R1
CLRB (R1)
CLRB 1(R1)
CLR SND.ENVELOPE-30(RO)
CLR SND.ENVELOPE-32(RO)
CLR SND.ENVELOPE-34(RO)
CLR SND.ENVELOPE-36(RO)
MOV R2, -(SP)
JSR PC, LOAD.OUT#STD.BUF
MOV RO, R1
CMP R1, 02001
BNE 1$ JSR PC, DECODE
global routine REC_DATA

Function Description:

These commands are used to communicate between the initiating host program and the remote program. Both send and receive commands specify a host buffer descriptor and a byte count. In the case of receive data, the information in the buffer is read by the remote program and a send data response sent back to the host to acknowledge receipt. In the case of receive data, the remote program writes data into the buffer up to the amount specified by the byte count and then sends a receive data response to the host to notify it of the transmission.

The send data and receive data commands are only legal when the server is in the active state. If the remote program terminates abnormally, putting the server back in the idle state, outstanding send data and receive data commands may be lost. In the event that the specified timeout interval is exceeded, the host program should issue a get duplex status command to see if the remote program is still running (i.e., the dup server is active); if it is, the progress indicator should be remembered and the timeout interval should be re-installed. If the second timeout expires without a response and a second get duplex status shows the remote program having made no progress in the interim, then the program should be considered broken and should be aborted.

Formal Parameters:

none

Implicit Inputs:

NSD_SLOT

This global storage gets loaded by the routine and in it is stored the next send ring descriptor slot where the port/controller should be polling on and the place to put this command to it.

Implicit Outputs:

none

Completion Codes:

RET_STATUS:

Return status passed back to calling routine.

Side Effects:

none

begin

local

REF_NUM,

REC_BUFFLOC,

TEMP;

! Stores unique cmd ref number

! Stores outstanding cmd buffer location

! A place to put read IP register data
Before we load up the command packet up
with all this good information get the
next send descriptor slot and a unique
command reference number.

GET_NSD();  //Get the next send desc slot
REF_NUM = GET_CMD4REF();  //Get a unique command ref num

UQ Port command envelope Header field definition

SND_ENVELOPE [NSD_SLOT, MSG_LENGTH] = SZ_REP;  //Load message length
SND_ENVELOPE [NSD_SLOT, CREDITS] = ONE;  //Load credit size
SND_ENVELOPE [NSD_SLOT, MSG_TYPE] = 0;  //Define message type 'Sequential'
SND_ENVELOPE [NSD_SLOT, CONN_ID] = 2;  //Define connection ID 'DUP'

DUP generic command envelope field definition

SND_ENVELOPE [NSD_SLOT, CMD_LREF] = REF_NUM;  //Load command reference number
SND_ENVELOPE [NSD_SLOT, CMD_HREF] = ZERO;  //Zero Hi order cmd ref num
SND_ENVELOPE [NSD_SLOT, UN_LUSED] = ZERO;  //Un used in DUP implementation
SND_ENVELOPE [NSD_SLOT, UN_MUSED] = ZERO;  //Un used in DUP implementation
SND_ENVELOPE [NSD_SLOT, DPCODE] = DP_REP;  //Duplicate commands op-code
SND_ENVELOPE [NSD_SLOT, RSV1] = ZERO;  //Not used field
SND_ENVELOPE [NSD_SLOT, MODIFIER] = ZERO;  //Define this commands modifiers

Command specific command envelope field definition

Byte count of transfer

SND_ENVELOPE [NSD_SLOT, BLO_CNT] = RECEB_SIZE;  //Byte count low word
SND_ENVELOPE [NSD_SLOT, BHI_CNT] = ZERO;  //Byte count high word

Buffer descriptor definition

SND_ENVELOPE [NSD_SLOT, BPA_LO] = REC_BUF;  //Low unibus addr <0-15>
SND_ENVELOPE [NSD_SLOT, BPA_HI] = ZERO;  //Unibus addr bits <16-17>
SND_ENVELOPE [NSD_SLOT, BPA_EXT] = ZERO;  //Reserved extension addrs
SND_ENVELOPE [NSD_SLOT, RSV1] = ZERO;  //Reserved field
SND_ENVELOPE [NSD_SLOT, UBA_CHAN] = ZERO;  //Unibus adaptor channel number
SND_ENVELOPE [NSD_SLOT, RSV0] = ZERO;  //These next four words are not
SND_ENVELOPE [NSD_SLOT, RSV1] = ZERO;  //Used in the UQ Port implementation
SND_ENVELOPE [NSD_SLOT, RSV2] = ZERO;  //Used
SND_ENVELOPE [NSD_SLOT, RSV3] = ZERO;  //Used

Call the load outstanding command buffer routine
and load this command into the buffer. The return
from this routine will point us to the buffer location
where this command is stored. Later we can look at
this location to see if the interrupt service routine
has received and process it.
REC_BUFLOC = LOAD_OUT_SBUF (.REF_NUM); 
!Load the command
if REC_BUFLOC equ OBFS杼E Code then DECODE (); 
!Error if buffer is full

!Set the ownership bit to 1 giving
!this slot to the port/controller
SEND_RING [.NSD_SLOT, OWN_BIT] = PORT OWNED;
!Read the IP register to stimulate port polling
TEMP = .RDRX_ADDR [RCIP, RC_ALL];

!Time out the port/controller for the response from this command.

If the controller times out then:

1. See what kind of error was returned. If the error
is a type other than a CTO_CODE (controller time out)
then call routine Decode which does the appropriate
action based on the error.

2. If the returned error is an CTO_CODE then do a get

data status and check the progress indicator to look
for an increase, indicating that the remote program is
still running and is not dead.

If the indicator hasn't changed then assume that the
remote program is dead and return an error code of
RPD_CODE (remote program dead code) and exit.

If the indicator has changed then assume that the
remote program is still running, save a copy of its
value and reinstate the controller time out delay and
repeat the loop.

As long as the progress indicator in the remote program
is still increasing this loop will be repeated for ever.

If the controller doesn't time out then return with the return code
returned from routine CTO_WAIT (), which could be either a success
or error code by definition of this host code.

while TRUE do 
!Repeat for ever
begin
    BREAK; 
!Flag control C's
    !Do a controller time out and determine if the controller
    !has processed the command or if a fatal error has occurred.
if CTO_WAIT (3000, .REF NUM, .REC BUF LOC) !Is return an error then
    begin
        if the return status code ends a CTO CODE then see if the remote program is still running. If it is then save the progress indicator and repeat the loop else call routine Decode ().
        if RET_STATUS ends CTO_CODE !Is this a controller time out then
            begin
                REF_NUM = .REF_NUM;
                if GET_DUS1_STATUS () then DECODE (); !Get the dust status
                if RET_ENIAD [PLO.IND] neq .PID_Save !Any progress been made then
                    PID_SAVE = RET_ENIAD [PLO.IND] !Still running save PID
                else
                    return RET_STATUS = RPD_CODE; !No progress so flag error
                end
            else
                !The return status code was not a controller time out code so something else is wrong. Call the routine Decode () to find out what went wrong.
                !DECODE ()
            end
        end
    else
        !The command has been received by the interrupt service.
        begin
            !Get this command's return envelope address out of the outstd_BUF and check for good return status error and die if bad status.
            RET_ENIAD = .OUTSTD_BUF [.REC BUF LOC, ENV ADR]; !Get the ret en. adr
            !Test for good status
            if RET_ENIAD [STATUS] neq zero !Test the status then

return RET STATUS - RSE CODE
else
    return RET STATUS;
end;

!Return a "Response status err" code

!This ret status is good or bad

!It won't compile without this here

000000 004137 000000G  
REC.DATA:  
SRTL  REC.DATA GLOBAL ROUTINE DECLARATIONS

JSR  R1, SAVE2
TST  - (SP)
JSR  PC, GET, NSD
JSR  PC, GET, CMD, REF
MOV  R0, R2
MOV  NSD, SLOT, - (SP)
MOV  @54, (SP)
JSR  PC, BLINE
MOV  @54, SND.ENVELOPE (RO)
MOV  @SND.ENVELOPE + 2, R1
ADD  R0, R1
MOV  @1, (R1)
MOV  @0002, R1
MOV  R2, SND.ENVELOPE + 4 (RO)
CLR  SND.ENVELOPE + 8 (RO)
CLR  SND.ENVELOPE + 10 (RO)
CLR  SND.ENVELOPE + 12 (RO)
MOV  @5, SND.ENVELOPE + 14 (RO)
CLR  SND.ENVELOPE + 15 (RO)
CLR  SND.ENVELOPE + 16 (RO)
MOV  @10, SND.ENVELOPE + 20 (RO)
CLR  SND.ENVELOPE + 21 (RO)
CLR  SND.ENVELOPE + 22 (RO)
MOV  @REC.BUF, SND.ENVELOPE + 24 (RO)
MOV  @SND.ENVELOPE + 26, R1
ADD  R0, R1
CLR  (R1)
CLR  1 (R1)
CLR  SND.ENVELOPE + 30 (RO)
CLR  SND.ENVELOPE + 32 (RO)
CLR  SND.ENVELOPE + 34 (RO)
CLR  SND.ENVELOPE + 36 (RO)
MOV  R2, (SP)
JSR  PC, LOAD.OUT + STD.BUF
MOV  R0, R1
MOV  @REC.BUF + LOC
CMP  R1, @2001
BNE  1
MOV  NSD, SLOT, RO
global routine SET_CNTLRCCHAR

Functional Description:
The SET CONTROLLER CHARACTERISTICS command is used to set host
settable unit characteristics and obtain those unit
characteristics that are essential for proper class driver
operation. This command never alters the unit's state
(“unit online”, “unit-available”, “unit-offline”). It is
meaningless to set host settable characteristics for a unit
that is “unit-available” or “unit-offline”.

Formal Parameters:
none

Implicit Inputs:
NSD SLOT
This global storage gets loaded by the routine
Get_nsd and it is stored the next send ring
descrriptor slot where the port/controller should
be polling on and the place to put this commands
command packet.

Implicit Outputs:
none

Completion Codes:
RET_STATUS: Return status passes back to the calling routine
the status of the just issued command.

Side Effects:
Any previously defined controller characteristics will possibly
be altered after execution of this command.

begin
local
REF_NUM, !Stores unique cmd ref number
SCC_BUFFLOC, !Stores outstanding cmd buffer location
TEMP1 !A place to put read IP register data

Before we load up the command packet up
with all this good information get the
next send descriptor slot and a unique
command reference number.

GET_NSD(); !Get the next send desc slot
REF_NUM = GET_COMMAND(); !Get a unique command ref num
!
! UQ Port command envelope Header field definition
SND_ENVELOPE [NSD SLOT, MSG_LENGTH] = SZ_SCC;
!Load message length
SND_ENVELOPE [NSD SLOT, MSG CREDITS] = ONE;
!Load credit size
SND_ENVELOPE [NSD SLOT, MSG TYPE] = 0;
!Define message type 'Sequential'
SND_ENVELOPE [NSD SLOT, MSG CONN ID] = 0;
!Define connection ID 'DUP'

! MSCP generic command envelope field definition

SND_ENVELOPE [NSD SLOT, CMD_REF] = REF_NUM;
!Load command reference number
SND_ENVELOPE [NSD SLOT, CMD_HREF] = ZERO;
!Zero Hi order cmd ref num
SND_ENVELOPE [NSD SLOT, UN_LUSED] = ZERO;
!Not used in DUP implementation
SND_ENVELOPE [NSD SLOT, UN_MUSED] = ZERO;
!Not used in DUP implementation
SND_ENVELOPE [NSD SLOT, OPCODE] = OP_SCC;
!Load this commands op-code
SND_ENVELOPE [NSD SLOT, RSVD] = ZERO;
!Not used field
SND_ENVELOPE [NSD SLOT, MODIFIER] = ZERO;
!Define this commands modifiers

! Command specific command envelope field definition

SND_ENVELOPE [NSD SLOT, MSCP VER] = ZERO;
!MSCP version
SND_ENVELOPE [NSD SLOT, CTL_FLAGS] = ZERO;
!Controller flags
SND_ENVELOPE [NSD SLOT, MOST TOVL] = ZERO;
!Most time out value
SND_ENVELOPE [NSD SLOT, RSIVD] = ZERO;
!Reserved
SND_ENVELOPE [NSD SLOT, TD0] = ZERO;
!Time and Date word 0
SND_ENVELOPE [NSD SLOT, TD1] = ZERO;
!Time and Date word 1
SND_ENVELOPE [NSD SLOT, TD2] = ZERO;
!Time and Date word 2
SND_ENVELOPE [NSD SLOT, TD3] = ZERO;
!Time and Date word 3
SND_ENVELOPE [NSD SLOT, CPLO] = ZERO;
!Cntlr dep parameter lo word
SND_ENVELOPE [NSD SLOT, CPHI] = ZERO;
!Cntlr dep parameter hi word

! Call the load outstanding command buffer routine
! and load this command into the buffer. The return
! from this routine will point us to the buffer location
! where this command is stored. Later we can look at
! this location to see if the interrupt service routine
! has received and processed it.

SCC_BUFFLOC = LOAD OUT STD BUF (.REF NUM);
!Load the command

if .SCC_BUFFLOC eq 08F CODE then DECODE ();
!Error if buffer is full

! Set the ownership bit to 1 giving this slot
! to the port/controller
SEND_RING [NSD_SLOT, OWN_BIT] = PORT OWNED;
! Read the IP register to stimulate port polling
TEMP = .RDRX_ADDR [RCIP, RC ALL];
! Time out thru port/controller processing the command.
ENVELOPE (2)
! The first test tests the connections ability to
! respond to this command without any errors in the SA
register and for the command not timing out.

The second tests the DUP server for good status. If
the status is sent back then an error code is returned
and the appropriate recovery. The timeout routine will loop on delaying and checking the hi:
bit of the first word in the outstdbuf for a true.
When true signals us that the interrupt service routine
has received the end packet and no connection errors
were detected.

if CTO_WAIT (3000, .REF NUM, .SCC_BUF&LOC) then DECODE (); !Is return an error

Get the return envelope address from the outstdbuf
at this command's buffer location and check the packet
for good status error and die if bad status was returned

RET ENV& = .OUTSTD_BUF [.SCC_BUF&LOC, ENV_ADDR]; !Get the ret env addr

Now test for good status

!Test the status

return RET STATUS = RSE CODE !Return a "response status err" code
else
return .RET STATUS; !This ret_status is good or bad
MOV _4, SND.ENVELOPE, 14(RO)  
CLR SND.ENVELOPE, 15(RO)  
CLR SND.ENVELOPE, 16(RO)  
CLR SND.ENVELOPE, 20(RO)  
CLR SND.ENVELOPE, 22(RO)  
CLR SND.ENVELOPE, 24(RO)  
CLR SND.ENVELOPE, 26(RO)  
CLR SND.ENVELOPE, 30(RO)  
CLR SND.ENVELOPE, 32(RO)  
CLR SND.ENVELOPE, 34(RO)  
CLR SND.ENVELOPE, 36(RO)  
CLR SND.ENVELOPE, 40(RO)  
CLR SND.ENVELOPE, 42(RO)  
MOV R2, (SP)  
JSR PC, LOAD, OUT$STD.BUF  
MOV RO, R1  
CMP R1, #2001  
BNE #1  
MOV R0, N5D.SLOT, RO  
ASL RO  
ADD SEND.RING, RO  
BIS #100000, 2(RO)  
MOV @RDX, ADDR, 4(RO)  
MOV 4(SP), RO  
MOV #5670, (SP)  
MOV R2, (SP)  
MOV R1, (SP)  
JSR PC, CTQ.WAIT  
CMP (SP)++, (SP)++  
DXX 60, 022662  
ROR RO  
ROR RO  
BCC 21  
JSR PC, DECODE  
MOV R1, RO  
MOV R1, RO  
MOV OUT$STD.BUF, 2(RO), RET.EN$AD  
MOV OUT$STD.BUF, 2(RO), RO  
TST 16(RO)  
BEQ 31  
MOV #31, RO  
MOV RO, RET.STATUS  
BR #41  
MOV RET.STATUS, RO  
ADD #6, SP  
RTS PC

Routine Size: 114 words, Routine Base: CODE$1, 4514
Maximum stack depth per invocation: 9 words

3961 1
global routine ON LINE = !Makes a unit come online to a host

Functional Description:
The online command is used to bring a unit "unit-online", set
host settable unit characteristics and obtain those unit
characteristics that are essential for proper class driver
operation. The unit is spun-up, if necessary, and is heads
are loaded prior to returning the online command's end
message. Host settable characteristics are set exactly as if
a set unit characteristics command were issued. Host settable
characteristics are set after the unit has been successfully spun up
and any other validity checks have succeeded. Note that the unit's
host settable characteristics are not altered if the unit is already
"unit-online".

Formal Parameters:
none

Implicit Inputs:
NSD_SLOT
This global storage gets loaded by the routine
'Get NSD' and in it is stored the next send ring
descriptor slot where the port/controller should
be polling on and the place to put this commands
command packet.

Implicit Outputs:
none

Completion Codes:
RET_STATUS: Return status passes back to the calling routine
the status of the just issued command.

Side Effects:
Any previously defined controller characteristics will possibly
be altered after execution of this command.

begin
local
REF(NUM,
ONL_BUFLOC,
TEMP;

Before we load up the command packet up
with all this good information get the
next send descriptor slot and a unique
command reference number.

GET NSD ();
C 4015  1  REF_NUM = GET_CMDREF();  !Get a unique command ref num
C 4016  1
C 4017  1
C 4018  1
C 4019  1  SND_ENVELOPE [.NSD_SLOT, MSG_LENGTH] = SZ_ONL;  !Load message length
C 4020  1  SND_ENVELOPE [.NSD_SLOT, CREDITS] = ONE;  !Load credit size
C 4021  1  SND_ENVELOPE [.NSD_SLOT, MSG_TYPE] = 0;  !Define message type 'Sequential'
C 4022  1  SND_ENVELOPE [.NSD_SLOT, CONN_ID] = 0;  !Define connection ID
C 4023  1
C 4024  1
C 4025  1
C 4026  1  SND_ENVELOPE [.NSD_SLOT, CMD_LREF] = REF_NUM;  !Load command reference number
C 4027  1  SND_ENVELOPE [.NSD_SLOT, CMD_MREF] = ZERO;  !Zero Hi order cmd ref num
C 4028  1  SND_ENVELOPE [.NSD_SLOT, UNIT_NUM] = UNIT_NO;  !Select unit to bring online
C 4029  1  SND_ENVELOPE [.NSD_SLOT, UN_MUSED] = ZERO;  !Not used in DUP implementation
C 4030  1  SND_ENVELOPE [.NSD_SLOT, OPCODE] = OP_ONL;  !Load this commands op-code
C 4031  1  SND_ENVELOPE [.NSD_SLOT, RSV0] = ZERO;  !Not used field
C 4032  1  SND_ENVELOPE [.NSD_SLOT, MODIFIER] = ZERO;  !Define this commands modifiers
C 4033  1
C 4034  1
C 4035  1
C 4036  1  SND_ENVELOPE [.NSD_SLOT, RSV10] = ZERO;  !Reserved
C 4037  1  SND_ENVELOPE [.NSD_SLOT, UNIT_FLAGS] = ZERO;  !Unit Flag field
C 4038  1  SND_ENVELOPE [.NSD_SLOT, RSV140] = ZERO;  !Reserved field
C 4039  1  SND_ENVELOPE [.NSD_SLOT, RSV141] = ZERO;  !Reserved field
C 4040  1  SND_ENVELOPE [.NSD_SLOT, RSV142] = ZERO;  !Reserved field
C 4041  1  SND_ENVELOPE [.NSD_SLOT, RSV143] = ZERO;  !Reserved field
C 4042  1  SND_ENVELOPE [.NSD_SLOT, RSV144] = ZERO;  !Reserved field
C 4043  1  SND_ENVELOPE [.NSD_SLOT, RSV145] = ZERO;  !Reserved field
C 4044  1  SND_ENVELOPE [.NSD_SLOT, DDP_LO] = ZERO;  !Device dependent parameter
C 4045  1  SND_ENVELOPE [.NSD_SLOT, DDP_HI] = ZERO;  !Device dependent parameter
C 4046  1  SND_ENVELOPE [.NSD_SLOT, CS_ADMW_UNIT] = ZERO;  !Shadow unit
C 4047  1  SND_ENVELOPE [.NSD_SLOT, CS_PY_SPEED] = ZERO;  !Copy speed
C 4048  1
C 4049  1  !Call the load outstanding command buffer routine
C 4050  1  and load this command into the buffer. The return
C 4051  1  will point us to the buffer location
C 4052  1  where this command is stored. Later we can look at
C 4053  1  this location to see if the interrupt service routine
C 4054  1  has received and process it.
C 4055  1
C 4056  1  ONL_BUFLOC = LOAD OUTSTDBUF (.REF_NUM);  !Load the command
C 4057  1
C 4058  1
C 4059  1  if .ONL_BUFLOC eqw OBJ_CODE then DECODE ();  !Error if buffer is full
C 4060  1
C 4061  1
C 4062  1  SET OWN BIT 1 giving this slot
C 4063  1  to the port/contoller
C 4064  1  SEND_RING [.NSD_SLOT, OWN BIT] = PORT_OWNED;
C 4065  1
C 4066  1  Read the IP register to stimulate port polling
C 4067  1
C 4068 1  TEMP = .RDRX ADDR (RCIP, RC ALL);
C 4069 1  !
C 4070 1  !  Time out the port/controller processing the command.
C 4071 1  !
C 4072 1  !  The first test tests the connections ability to
C 4073 1  !  respond to this command without any errors in the SA
C 4074 1  !  register and for the command not timing out.
C 4075 1  !
C 4076 1  !  The second tests the DUP server for good status. If
C 4077 1  !  bad status is sent back then an error code is returned
C 4078 1  !  to the calling routine where the routine “decode” will
C 4079 1  !  decode and take the appropriate recovery. The time
C 4080 1  !  out routine will loop on delaying and checking the hi
C 4081 1  !  bit of the first word in the .OUTSTD_BUF for a true.
C 4082 1  !  When true signals us that the interrupt service routine
C 4083 1  !  has received the end-packet and no connection errors
C 4084 1  !  were detected.
C 4085 1  !
C 4086 1  !
C 4087 1  if CTO_WAIT (3000, .REF_NUM, .ONL_BUF#LOC) then DECODE (); !Is return an error
C 4088 1  !
C 4089 1  !
C 4090 1  !
C 4091 1  !  Get the return envelope address from the .OUTSTD_BUF
C 4092 1  !  at this commands buffer location and check the packet
C 4093 1  !  for good status error and die if bad status was returned
C 4094 1  !
C 4095 1  !
C 4096 1  !
C 4097 1  !  RET_EN$AD = .OUTSTD_BUF (.ONL_BUF#LOC, ENV_ADR); !Get the ret env adr
C 4098 1  !
C 4099 1  !
C 4100 1  !
C 4101 1  !  if RET_EN$AD [STATUS] neq ZERO then !Test the status
C 4102 1  !
C 4103 1  !
C 4104 1  !
C 4105 1  !  return RET_STATUS = RSE_CODE
C 4106 1  !This ret_status is good or bad
global routine INITI_SERVICE : INT LNK$TYP novalue = Init sequence interrupt catcher

***

Functional Description:

During the initialization sequence, the IE bit is defined to be zero. This means that the host is not requesting interrupts at the completion of steps 1-3.

Note that no interrupt will be generated at the completion of step 4 since this step requires only a small number of time.

This interrupt service routine serves to catch any interrupts that the controller might issue during the initialization sequence. The interrupt is ignored and control is returned.

Formal Parameters:

none

Implicit Inputs:

none

Implicit Outputs:

none

Completion Codes:

none

Side Effects:

none

begin
begin
return;
return;
end;
end;

SBITL INITI_SERVICE GLOBAL ROUTINE DECLARATIONS

INTI_SERVICE::

RTI

Routine Size: 1 word.
Routine Base: $CODE$ + 5060
Maximum stack depth per invocation: 0 words
global routine IS_TIMER (SEQ_NO) =  

Functional Description:

Steps 1-3 of the init sequence, each are required to complete within 10 seconds. If any of these steps fails to complete within that period, this is to be treated as a host detected fatal error.

This routine will do one us delays for a total of 10 seconds. After each delay the step field is examined to see if this init sequence has completed.

Formal Parameters:

SEQ_NO: Indicated which init step is presently being performed within the RDRX init sequence.

Implicit Inputs:

none

Implicit Outputs:

none

Completion Codes:

TRUE: Indicates to the calling routine that the indicated init sequence step has timed out.

FALSE: Indicates to the calling routine that the indicated init sequence step has not timed out.

Side Effects:

If the init sequence step times out and an error is posted in the 99 register then the routine decode will be called.

begin

local

STEP_VAL : word;  !Temp storage of step value

STEP_VAL = ZERO;  !Make sure the loc is zeroed out

;!

; Select the step value expected from this init sequence step.

;!

selectoneu .SEQ_NO of

set

[0] :
4195 2 STEP_VAL = '0b0001';  !Step 1 binary value
4197 2 [1]:
4198 2 STEP_VAL = '0b0010';  !Step 2 binary value
4199 2 [2]:
4200 2 STEP_VAL = '0b0100';  !Step 3 binary value
4201 2
4202 2 [3]:
4203 2 STEP_VAL = '0b1000';  !Step 4 binary value
tes;
4206 2
4207 2 ! Loop on the 100 micro second delay until:
4208 2 ! either the expected step field is read in
4209 2 ! the SA register or the step times out.
4210 2 :
4211 2 incr TIM_OUT from 0 to 15000 do  !Delay for 10 seconds
4212 2 begin
4213 3 !Do the delay
4214 3 DELAY(C_US);
4215 3 end;
4216 2 ! Check the step bit to see if it is set yet.
4217 2 ! If it is set then return a false indicating
4218 2 ! the completion else continue delaying.
4219 2 :
4220 2 if .RDRX_ADDR [RCSA, STP_FIELD] equ .STEP_VAL then return FALSE;
4221 2
4222 2 BREAK;
4223 2 !Service any control C's
4224 2 end;
4225 2
4226 2 ! This step has not completed within the specified
4227 2 ! 10 second time interval. Test the SA register
4228 2 ! for any errors posted and report errors if any.
4229 2 ! Return a true to the caller indicating the error.
4230 2 :
4231 2 if .RDRX_ADDR [RCSA, ERR_BET] then  !Is the error bit set
4232 2 begin
4233 3 !Indicate the port/fatal error code
4234 3 RET_STATUS = PFE_CODE;
4235 3 !Report the error
4236 3 DECODE ();
4237 3 end;
4238 2 return TRUE;
4239 2 end;
4240 2
4241 2
4242 1
Routine Size: 18 words
Routine Base: $CODE+ 5062
Maximum stack depth per invocation: 9 words
global routine BOOT RDRX - !Performs RDRX init sequence

Functional Description:
This routine performs the initialization sequence of the RDRX
RDRX controller.
The initialization procedure serves to:
1. Identify the parameters of the host resident communications
region to the port.
2. Provide a confidence check of port/controller integrity.
3. Bring the port/controller online to the host (note that the
devices attached to the controller are not thereby brought
online to the class driver.)

Formal Parameters:
one

Implicit Inputs:
ISO.STRUCT Stores the init sequence read and write data defined
for this program and controller.

Implicit Outputs:
one

Completion Codes:
Success: Is returned to the calling routine if this initialization
sequence was executed successfully.
Failure: Is returned to the calling routine if this initialization
sequence was not executed successfully.

Side Effects:
Any DM code that might have been running in the DM machine will be
aborted.
Any outstanding commands or response pertaining to a process using
the controller will be lost.

begin
local TEMP : word; !Temporary storage location

!The host begins the initialization sequence
!either by issuing a bus init or by writing
!any value into the IP register; the port must
!guarantee that the host will read zeros in SA
on the next bus cycle. Initialization then
sequences through steps 1-4 as per VAXSP.DOC
version 1.5.

Write to the IP register and start the init
sequence going.

!:RCAP (RCAP, RC ALL, ONES):

This incr loop performs all four steps of the
initialization sequence described above. The
SA write and read data is preset into the
structure "STATE_STRUCT" and stands for
"Initialization Sequence Data_STRUCT".

If a step time-out error occurs, the test
invoking this routine will take the necessary
retry procedure. A return code of failure is
returned.

If any SA register compare error is detected after
a step completion, the routine Decode will decode the
error and load statistical tables up pertinent data.

incr SEQ_NO from STEP1 to STEP4 do  !Do the four init seq steps
begin

! Wait for the controller to load the SA reg
! up with the step data.

if IS_TIMER (.SEQ_NO)  !Did the Controller time out
then
begin

!DO SOME STAT TABLE UP DATA TO SHOW
!THE TIME OUT

PRINTF (.EMSG_STRUCT [MSG10]);
return FAILURE;
end;

! The controller did not time out so read the SA register
for the expected step data and compare it to the good
! data stored in STATE_STRUCT.

! If the read SA data is not what we expect, return a
Note that the reserved fields read in the SA register are or'ed with all ones to mask out the field before compared to the expected data stored in the structure "ISO STRUCT".

```assembly
if TEMP neq .ISD_STRUCT [.SEQ.NO, ISRQ, ISR ALL] !Compare read to expected then
    begin
        ! Load some statistical table up with some data to indicate that the init sequence had some trouble.
        PRINTF (.EMSG_STRUCT [MSG11]);
        return FAILURE;
    end;
```

We need to save the micro code version number if this is step4.
Save the ucode version in UC_VER if this is step4.

```assembly
if .SEQ.NO equ STEP4 !Is this seq step 4 then
    begin
        UC_VER = .RDRX_ADDR [RCSA, S4R_VER]; !Save the ucode version
    end;
```

This step read data is what we expected so write the SA register with this steps write data stored in ISO_STRUCT.

```assembly
WRT_RDRX (RCSA, RC_ALL, .ISD_STRUCT [.SEQ.NO, ISWRT, ISW ALL]); !Write the SA register
```

The controller initialization sequence was done successfully so return a success code.

```assembly
return SUCCESS;
```
Routine Size: 85 words, Routine Base: $CODE$: 5316
Maximum stack depth per invocation: 13 words
global routine INIT_COM_AREA - !Init of DUP Protocol communication area

::

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.
side effects:

begin

make sure that the controller has done its part of the DUP protocol by clearing out the ring buffers.

if the rings are not cleared out then return with an error code of true.

incru : from 2 to RING_SIZE = 1 do !test all blocks for zeros

incru j from WR00 to WR01 do !test all words for zeros

if .COM AREA [.i,.j, WORD REF] nequ ZERO then return CIE CDF;

! the port did its part of the protocol so now define the address locations of the HEAD AREA,

RECEIVE_RING and SEND_RING from the contiguous storage declared by COM AREA.

HEAD_AREA = COM_AREA;
RECEIVE_RING = COM_AREA [REC_BASE];
SEND_RING = COM_AREA [SDN_BASE];

not quite sure if the port has to clear out the header area of the communications area
so i'll clear it out here just in case.

incru : from WR00 to WR01 do

HEAD_AREA [.i,.WORD REF] = ZEROS;

load up the Send Ring descriptors with an envelope address.
define the "Flag bit" to = 1 (interrupt requested), define the "Ownership bit" to = 0 (owned by host) and load the Reserved field with zeros (per DPU spec).

incru : from 0 to SND_ALLOCATE = 1 do
begin
SEND_RING [.; LO_ENIAD] = SND_ENVELOPE [.; CMD_LREF]; !Low order envelope address for all sys
SEND_RING [.; MI_ENIAD] = ZERO; !High-order portion of an 18 bit U/Q bus addr
SEND_RING [.; QB_EXT] = ZERO; !Q_bus extension
SEND_RING [.; D1R5VD] = ZERO; !Reserved field
SEND_RING [.; FLAG_BIT] = SET FLG; !Flag bit whose meaning varies depending on disc state
SEND_RING [.; OWN_BIT] = MOST-Owned; !Indicates whether disc is host or port owned
end;

! Load up the Receive Ring descriptors with an envelope
! address, define the "Ownership bit" = 1 (owned by port),
! define the "Flag bit" to = 1 (interrups requested) and
! the reserved field set to zeros (per DUP spec).
!
incr i from 0 to REC_ALLOCATE - 1 do
begin
RECEIVE_RING [.; LO_ENIAD] = REC_ENVELOPE [.; CMD_LREF];
RECEIVE_RING [.; MI_ENIAD] = ZEROS;
RECEIVE_RING [.; QB_EXT] = ZEROS;
RECEIVE_RING [.; D1R5VD] = ZEROS;
RECEIVE_RING [.; FLAG_BIT] = SET_FLG;
RECEIVE_RING [.; OWN_BIT] = PORT-Owned;
end;

! Reset the communications area pointer to
! their initial state.
!
NRD_SLOT = 1; !Start ring pointer at zero
NSD_SLOT = 1; !Start ring pointer at zero
NXT_CMN = 0; !Start unique cmd ref num at one
!
! Set the response envelope message length size equal
! to the buffer size in bytes starting at text + 0.
!
incr i from 0 to REC_ALLOCATE - 1 do
RECV_ENVELOPE [.; MSG_LENGTH] = RB_SIZE+2; !Convert to bytes before loading
!
! Init the outstanding command buffer as follows:
! 1. Indicate that all slots are unused by loading
!    the unique value "0100000".
! 2. Clear the envelope addr words to zero.
!
incr i from 0 to REC_ALLOCATE - 1 do
begin
OUTSTD BUF [.; CMD WRD] = &H'100000'; !Define the slot as unused
OUTSTD_BUF [.i., ENV ADDR] = ZERO;

end;

end.

:* No errors detected by this routine so
*: return with an non error code of false.

return PAS_CODE;

end;

end.

5BITL INIT.COM.AREA GLOBAL ROUTINE DECLARATIONS

0000 004137 000000G

INIT.COM.AREA:

JSR R1. #SAVE3

MOV #4.R1  
| J.

MOV R1,R0

ADD R2.R0  
| J.*

ASL R0

TST COM.AREA(RO)

BEQ 31

MOV 01.R0

RTS PC

31: INC R2  
| J.

CMP R2,#1  
| J.*

BLS 21

ADD #2.R1

CMP R1,#22

B 18

MOV #COM.AREA,HEAD.AREA

MOV #COM.AREA.10,RECEIVE.RING

MOV #COM.AREA.30,SEND.RING

CLR RO  
| I.

CLR RO

MOV RO,#1

ADD HEAD,AREA.R1

CLR (R1)

ADD #2.R0  
| J.*

CMP RO,#6

BLS 41

CLR R3  
| I.

MOV R3.R1  
| J.*

MOV R1,#2

ADD SEND.RING,R2

MOV R3.(SP)

MOV #54.(SP)

JSR PC.BEMUL

ADD #SEND.ENVELOPE+4.R0

MOV RO,#2

MOV RO,#2

ADD SEND.RING.R0

MOV #SEND.RING+4
global routine copy (ff, ttt, length) = !copy a string

begin
  incr i from 0 to .length by 2 do
    ttt := ttt . i = . (ff . i)
  end;
### GLOBAL ROUTINE DECLARATIONS

<table>
<thead>
<tr>
<th>Address</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>004137</td>
<td>COPY GLOBAL ROUTINE DECLARATIONS</td>
</tr>
<tr>
<td>0004</td>
<td>005001</td>
<td>JSR R1,$SAVE2</td>
</tr>
<tr>
<td>0006</td>
<td>004111</td>
<td>CLR R1</td>
</tr>
<tr>
<td>0010</td>
<td>010102</td>
<td>BR 2$</td>
</tr>
<tr>
<td>0012</td>
<td>066002</td>
<td>MOV R1,R2</td>
</tr>
<tr>
<td>0016</td>
<td>010100</td>
<td>ADD 12(SP),R2</td>
</tr>
<tr>
<td>0020</td>
<td>066000</td>
<td>MOV R1,R0</td>
</tr>
<tr>
<td>0024</td>
<td>010112</td>
<td>ADD 14(SP),R0</td>
</tr>
<tr>
<td>0026</td>
<td>062701</td>
<td>ADD P2,R1</td>
</tr>
<tr>
<td>0032</td>
<td>020166</td>
<td>CMP R1,10(SP)</td>
</tr>
<tr>
<td>0036</td>
<td>01764</td>
<td>BLD 11</td>
</tr>
<tr>
<td>0040</td>
<td>012700</td>
<td>MOV @-1,R0</td>
</tr>
<tr>
<td>0044</td>
<td>000207</td>
<td>RTS PC</td>
</tr>
</tbody>
</table>

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

### Library Statistics

<table>
<thead>
<tr>
<th>File</th>
<th>Total</th>
<th>Loaded</th>
<th>Percent</th>
<th>Mapped</th>
<th>Processing Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISK#USER2:[YOUNG.FMT]ZQBB0.L16;A</td>
<td>358</td>
<td>313</td>
<td>87</td>
<td>18</td>
<td>00:00:1</td>
</tr>
</tbody>
</table>

### OTS external references

*GLOBL $SAVE4, $SAVE3, $SAVE2, BL$MUL*
GLOBAL ROUTINE DECLARATIONS

COMMAND DUAL FILES


job:

1614 code, 0 data words

Run time: 01:09.3
Elapsed time: 01:21.1
new CPU Min: 966.4
remaining CPU Min: 19086
Memory used: 914 pages
Some job on Complete
MODULE ZROBS

IDENT - 'REV C PATCH 0'.

ADDRESSING MODE (ABSOLUTE).

ENVIRONMENT (NOEIS)).

BEGIN

LIBRARY 'ZROBS.LIB'.

!Define RDRx Formatter library

REQUIRE 'BLSPAC.REQ'.

!Define Bliss macro require file

#SUBTTL 'LAST ADDRESS AND SETUP SECTION'
The LASTAD macro must be the final statement (except .end) in a program. The call generates an even address reflecting the first word of memory unused by the program.

LASTAD

! Hardcoded P-TABLES

These optional hardware P-TABLES are located (when present) between the "LASTAD" macro and the "END" statement. These hardware P-TABLES are above and beyond the default hardware P-TABLE located in the main body of the program. These P-TABLES wind up appended to the BIN file of the diagnostic, just as though the supervisor or the "SETUP" utility had built them there. Thus the diagnostic can be "pre-parameterized" by the programmer.

If this hardcoded P-TABLE section is not wanted then define "number" in the BONSETUP macro to zero and omit BGNTAB and ENDTAB macros.

Coding sample is as follows:

LASTAD

BONSETUP (Number) : Number of P-TABLES

BGNPTAB (DATA)

BGNPTAB (DATA)

ENDPTAB

BGNPTAB (DATA)

ENDPTAB

ENDSETUP

.END

BONSETUP (1):

BGNPTAB

(ENDTAB

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

ORION  =  5       ; RELATIVE CPU SPEEDS
LCP    =  1
LCPTYP =  3       ; KDF11 PROCESSOR TYPE CODE

.SETCPU:

14 000000 010046  MOV  RO-, (SP)
15 000002 012767 000001 000000  MOV  #LCP, LCPU
16 000010 000007  MPRT
17 000012 020027 000003  CMP  RO, #LCPTYP  ; 11 23?
18 000016 001403  BEQ  10$  ; YES, MULTIPLIER OF 1
19 000020 012767 000005 000000  MOV  #ORION, LCPU  ; NO, SET MULTIPLIER OF 5
20 000026 012600 10$: MOV  (SP)+, RO
21 000030 00207  RTS  PC

.END
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000</td>
<td>001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000032</td>
<td>002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**MVS DETECTED:** 0

**TOTAL MEMORY USED:** 45 WORDS (1 PAGES)
**HIMEM MEMORY:** 20060 WORDS (77 PAGES)
**USED TIME:** 00:00:06
**CPU.OBJ,SETCPU.LIS,SP,SETCPU.MAC**
MACRO DEFINITIONS

WRI_RDRX (0, FIELDNam, IMAGE) :

begin
  local
    RCIM_REG;
    RCIM_REG = .RDRX_ADDR [0, RC_ALL];
    ATSM_REG = .field:and (FIELDNam), IMAGE;
    (.RDRX_ADDR, #upvar=0), .RCIM_REG;
end,

up Protocol Macros Declarations

REC BASE =
  HDR_SIZ, 0, 0, 16, 0%,
SND_BASE =
  HDR_SIZ, REC_ALLOCATE, 0, 0, 16, 0%,

macro to clear out the DUP send data text buffer
before requesting input form operator. This by
default puts a null byte at the end of the asc:
input.
CLR_SBUF =
  incr x from 0 to SNDB SIZE - 1 do
  SND_BUF [x] = ZEROS;#

macro to clear out the DUP send data text buffer
before requesting input form operator. This by
default puts a null byte at the end of the asc:
input.
CLR_RBUF =
  incr x from 0 to ESZ.RED 1 do
  msgadr [x] = ZEROS;#

general purpose word reference field select
WORD_REF =
  0, 16, 0%;
DEFINITIONS

BIT15 = "100000",
BIT14 = "40000",
BIT13 = "20000",
BIT12 = "10000",
BIT11 = "4000",
BIT10 = "2000",
BIT09 = "1000",
BIT08 = "400",
BIT07 = "200",
BIT06 = "100",
BIT05 = "40",
BIT04 = "20",
BIT03 = "10",
BIT02 = "4",
BIT01 = "2",
BIT00 = "1",

EF32:EF17 RESERVED FOR SUPERVISOR TO PROGRAM COMMUNICATION

EF_START = 32,
EF_RESTART = 31,
EF_CONTINUE = 30,
EF_NEW = 29,
EF_PWR = 28,

LEVEL DEFINITIONS

PRI07 = "340",
PRI06 = "300",
PRI05 = "240",
PRI04 = "200",
PRI03 = "140",
PRI02 = "100",
PRI01 = "40",
PRI00 = "0",
<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>VL</td>
<td>4'</td>
</tr>
<tr>
<td>JT</td>
<td>10'</td>
</tr>
<tr>
<td>DR</td>
<td>20'</td>
</tr>
<tr>
<td>DU</td>
<td>40'</td>
</tr>
<tr>
<td>SR</td>
<td>100'</td>
</tr>
<tr>
<td>JAM</td>
<td>200'</td>
</tr>
<tr>
<td>IOE</td>
<td>400'</td>
</tr>
<tr>
<td>NNT</td>
<td>1000'</td>
</tr>
<tr>
<td>RJ</td>
<td>2000'</td>
</tr>
<tr>
<td>XE</td>
<td>4000'</td>
</tr>
<tr>
<td>BE</td>
<td>10000'</td>
</tr>
<tr>
<td>ER</td>
<td>20000'</td>
</tr>
<tr>
<td>DE</td>
<td>40000'</td>
</tr>
<tr>
<td>DOE</td>
<td>100000'</td>
</tr>
</tbody>
</table>
MSG0 = 0,
MSG1 = 1,
MSG2 = 2,
MSG3 = 3,
MSG4 = 4,
MSG5 = 5,
MSG6 = 6,
MSG7 = 7,
MSG8 = 8,
MSG9 = 9,
MSG10 = 10,
MSG11 = 11,
MSG12 = 12,
MSG13 = 13,
MSG14 = 14,
MSG15 = 15,
MSG16 = 16,
MSG17 = 17,
MSG18 = 18,
MSG19 = 19,
MSG20 = 20,
MSG21 = 21,
MSG22 = 22,
MSG23 = 23,
MSG24 = 24,
MSG25 = 25,
MSG26 = 26,
MSG27 = 27,
MSG28 = 28,
MSG29 = 29,
MSG30 = 30,
ALURE = 0,
SUCCESS = 1,
ILK0 = 0,
ILK1 = 1,
ILK2 = 2,
ILK3 = 3,
AEO0 = 0,
AEO1 = 1,
AEO2 = 2,
AEO3 = 3,
AEO4 = 4,
AEO5 = 5,
AEO6 = 6,
SRD = 0,
SWRT = 1,
TEP1 = 0,
TEP4 = 1,
RUO = 1,
ALSE = 0,
LR_FLAG = 1,
NE = 1,
ERO = 0,
NES = '177777',
EROS = 0,
LIMIT = 16,
SIZE = 8192,
CT_SIZE = 4096,

!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
!Initialization sequence step one
!Initialization sequence step two
!True indicator
!False indicator
!Set the indicated flag to one
!Clear the indicated flag to zero
!Ones data type
!Zero data type
!All ones data type
!All zeros data type
!Allowable number of units to format
!Size of DM buffer size
!Size of Ift buffer size

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

MOD_MSG = 'b'00',
TIM_MSG = 'b'01',
ND_MSG = 'b'10',

!Command opcode message type
!Attention opcode msg type
!End opcode message type
Error code literals

PAS_CODE = #0'00',  !Pass code
CIE_CODE = #0'01',  !Communication area init error
CIO_CODE = #0'11',  !Controller time out error
PFQ_CODE = #0'21',  !Port fatal error
AIEEE_CODE = #0'31',  !Response status error
PVE_CODE = #0'41',  !Host/Controller out of sequence
APD_CODE = #0'51',  !Remote program died error code
PSE_CODE = #0'61',  !Port/host synchronous error
MLE_CODE = #0'71',  !Message length error code
UEC_CODE = #0'101',  !Unknown odd code error
APR_CODE = #0'201',  !Adapter purge request error
UNL_CODE = #0'301',  !Unknown interrupt error code
ATN_CODE = #0'401',  !Attention msg received
CMD_CODE = #0'501',  !Command msg received
SEX_CODE = #0'601',  !Serious exception error received
IVC_CODE = #0'701',  !Invalid command error received
UPM_CODE = #0'1001',  !Unknown message type
OBF_CODE = #0'2001',  !Outstanding buffer full error
OSE_CODE = #0'3001',  !Outstanding buffer out of sync error
UMN_CODE = #0'4001',  !Unknown message number
FRE_CODE = #0'5001',  !File read error code from load media

Dup Protocol literals

Note:
The values assigned to the literals
REC_SIZ, SND_SIZ are represented in
powers of 2 notation per DUP spec.

By redefining these two literals the
communications area allocation and 'init'
sequence is automatically handled and
no other parameter modification is needed.

Further more the send and receive rings
can be of different lengths. However the
maximum value allowed is 7 (2^7 = 128 slots)
per DUP spec.

REC_SIZ = 2,  !Define number of receive slots
SND_SIZ = 2,  !Define number of send slots
In one tricky: (Hdr siz).
The communication area is defined to be a contiguous Blockvector (two words per block)
data segment of size. Rec allocate + SND allocate + Hdr siz. The two words per block coming from the
two words needed to represent the ring descriptors.

Hdr siz is then really * 2 or 4 words of storage
to represent the interrupt and purge indicators.

MDR SIZ = 2.
!Define com area header size 4 words

REC_ALLOCATE = 1*REC SIZ,
!Define receive ring allocation
SND_ALLOCATE = 1*SND SIZ,
!Define send ring allocation

Ring size equals the total number of ring descriptors
(number of 2 word blocks) allocated within the
communications area.

RING SIZE = REC_ALLOCATE + SND_ALLOCATE + HDR_SIZ.

The RB_SIZ by definition, DUP spec. must be
a minimum of 60 bytes long (30 words) 64 bytes
overall. The additional 2 words for the UQ
port information is accounted for in azkel2
when the storage is allocated.

RB_SIZ = 30.
!Number of "words" in response buffer

The biggest command I'll ever send will
be (I hope) will be 40 bytes, 42 overall,
and again the UQ port 2 word is allowed
for 'n azkel2.

SB_SIZ = 20,
!Number of "words" in send buffer

SND_SIZ = 37,
SND_COUNT = 16,
REC_SIZ = 120,
PORT_OWNED = 1,
HOST_OWNED = 0,
Delay literal values

An argument of one results in a 100us delay.

A roman numeral notation is used to denote values of delay arguments.
The notation is as follows:

I (1) V (5) X (10) L (50) C (100) D (500) M (1000)

Any symbol following another of equal or greater value adds to its value, as II = 2, XI = 11.

Any symbol proceeding one of greater value subtracts from the second and the remainder added to the first as XIV = 14, LIX = 59.

```
ONE_SEC  = 10000,
C_US    = 1,
CC_US   = 2,
CCC_US  = 3,
XC_US   = 4,
D_US    = 5.
```

One second delay argument
100 micro sec delay argument
200 micro sec delay argument
500 micro sec delay argument
1000 micro sec delay argument
5000 micro sec delay argument

RDRX register offsets

RCIP = 0,
RCSA = 1.

Command packet opcodes
(See note following)

```
OP_ABD = \#o11,
OP_ACC = \#o20,
OP_AVL = \#o10,
OP_CCD = \#o21,
OP_CMP = \#o40,
OP_DAP = \#o12,
OP_ERS = \#o22,
OP_FLU = \#o23,
OP_GCS = \#o02,
OP_GUS = \#o03,
OP_ONL = \#o11,
OP_RDR = \#o41,
OP_RPL = \#o24,
OP_SCC = \#o04,
OP_SUC = \#o12,
OP_WR = \#o42,
OP_MRD = \#o30,
OP_MWR = \#o31.
```

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
Maintenance read command
Maintenance write command
End message and serious exception encodings
(see note following)

OP END = \$'200', !End packet flag
OP SEX = \$'0', !Serious exception end packet

MSCP Attention message encodes

OP AFA = \$'100', !Available attention message
OP DUN = \$'101', !Duplicate unit number attention message
OP ACX = \$'102', !Access path attention message
OP RCM = \$'103', !Reset command limit attention message

The following are the dup op code commands

OP GOS = \$'1', !Get dust status
OP ESP = \$'2', !Execute supplied program
OP ELP = \$'3', !Execute local program
OP SED = \$'4', !Send data
OP RDC = \$'5', !Receive data
OP ABT = \$'6', !Abort program

NOTE:

End message opcodes (also called endcodes) are formed by adding the end message flag to the command opcode. For example, a READ commands end message contains the value OP.RED + OP.END in its opcode field. The invalid command end message contains just the end message flag (i.e., OP.END) in its opcode field. The serious exception opcode shown above (i.e., OP.SEX + OP.END) in its opcode field.

Commands opcode bits 6 and 7 indicate the type of message (command, end or attention message). Command opcodes bits 3 through 5 indicate the command category (immediate, sequential or no-sequential) and whether or not the command includes a buffer descriptor.

See MSCP document appendix "A 1 NOTE:" for more information on this topic.

! DUP endcode message types

<p>| EOP_GOS | OP_GOS | OP_END | !Get dust status |
| EOP_ESP | OP_ESP | OP_END | !Execute supplied program |
| EOP_ELP | OP_ELP | OP_END | !Execute local program |
| EOP_RDC | OP_RDC | OP_END | !Receive data |
| EOP_SED | OP_SED | OP_END | !Send data |
| EOP_ABX | OP_ABX | OP_END | !Abort program |</p>
<table>
<thead>
<tr>
<th>MSCP encode message types</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOP ABO</td>
</tr>
<tr>
<td>EOP ACC</td>
</tr>
<tr>
<td>EOP AVL</td>
</tr>
<tr>
<td>EOP CCD</td>
</tr>
<tr>
<td>EOP CMP</td>
</tr>
<tr>
<td>EOP ERS</td>
</tr>
<tr>
<td>EOP FLU</td>
</tr>
<tr>
<td>EOP GCS</td>
</tr>
<tr>
<td>EOP GUS</td>
</tr>
<tr>
<td>EOP ONL</td>
</tr>
<tr>
<td>EOP RDL</td>
</tr>
<tr>
<td>EOP RPL</td>
</tr>
<tr>
<td>EOP SCC</td>
</tr>
<tr>
<td>EOP SUC</td>
</tr>
<tr>
<td>EOP WR</td>
</tr>
<tr>
<td>EOP MND</td>
</tr>
<tr>
<td>EOP MWR</td>
</tr>
<tr>
<td>EOP SEX</td>
</tr>
</tbody>
</table>

!Abort command
!Access command
!Available command
!Compare controller data command
!Compare host data command
!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
!Maintenance read command
!Maintenance write command
!Serious exception

Dup Command message envelope byte sizes beginning at text0 of the command envelope.

SZ CDS = 12,
SZ ESP = 40,
SZ ELP = 48,
SZ PED = 28,
SZ SED = 28,
SZ ABT = 12,

!Execute status size
!Execute supplied program size
!Execute local program size
!Receive data size
!Send data size
!Abort program size

MSCP Command message envelope byte sizes beginning at text0 of the command envelope.

SZ SCC = 32,
SZ ONL = 36,

!Set Controller characteristics
!On line command
The following are the expected number of bytes in a command end packet transmitted by the communications mechanism to the host.

! DUP command end message sizes
! ESZ_GDS * DECEMAL '22',
! ESZ_ESP * DECEMAL '12',
! ESZ_ELP * DECEMAL '16',
! ESZ_RED * DECEMAL '48',
! ESZ_SED * DECEMAL '48',
! ESZ_ABT * DECEMAL '12',

! MSCP command end message sizes
! ESZ_SCC * DECEMAL '28',
! ESZ_ONL * DECEMAL '44',

! DUP STANDALONE FLAG MODIFIER
! DUP_STND = 1,

! JDT Trap Vector
! O_TVEC = #0 14 ;
FIELD DECLARATIONS

Definitions:

ISO.FIELD - Initialization Sequence Data Field

ISRD  - Initialization Sequence Read = 0

ISWRT - Initialization Sequence Write = 1

S1R   - Step One Read

S1W   - Step One Write

etc.
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
STEP_FILD = [11, 4, 0],  ! All step bit fields
SA_GO = [0, 1, 0],  ! Status register Go bit
ERR_CODE = [0, 11, 0],  ! SA register fatal error code

: Step one read SA register field reference

S1R_STEP = [11, 1, 0],  ! Step one step bit
S1R_NV = [10, 1, 0],  ! No host 'inter vec settable adrs
S1R_QB = [9, 1, 0],  ! 22-bit addressing support
S1R_DI = [8, 1, 0],  ! Enhanced diag implementation
S1R_RSVD = [0, 8, 0],  ! Reserved field

: Step one write SA register field reference

S1W_WR = [14, 1, 0],  ! Diag wrap around
S1W_CRING = [11, 3, 0],  ! Number of C-ring slots 'pwrs of 2'
S1W_ARRING = [8, 3, 0],  ! Number of A-ring slots 'pwrs of 2'
S1W_IE = [7, 1, 0],  ! Init Sequence interrupt request
S1W_VADR = [0, 7, 0],  ! Interrupt vector address

: Step two read SA register field reference

S2R_STEP = [12, 1, 0],  ! Step two step bit
S2R_TYP = [8, 0, 0],  ! Port type number
S2R_BIT7 = [7, 1, 0],  ! Echoed IE bit from step one write
S2R_WR = [6, 1, 0],  ! Echoed bit 14 from step one write
S2R_CRING = [5, 3, 0],  ! Echoed bits 3-5 from step one write
S2R_ARRING = [0, 3, 0],  ! Echoed bits 0-2 from step one write

: Step two write SA register field reference

S2W_LRBASE = [0, 16, 0],  ! Ring base lower address
S2W_PI = [0, 1, 0],  ! Adapter purge interrupt request

: Step three read SA register field reference

S3R_STEP = [13, 1, 0],  ! Step three step b'it
S3R_RSVD = [8, 3, 0],  ! Reserved
S3R_IE = [7, 1, 0],  ! Echoed IE bit from step one write
S3R_VADR = [0, 7, 0],  ! Echoed VADR from step one write

: Step three write SA register field reference

S3W_PP = [15, 1, 0],  ! Purge & Poll test request
S3W_HBASE = [0, 15, 0],  ! Ring base high address
! Step four read SA register field reference

S4R_STEP = [14, 1, 0], !Step four step bit
S4R_RSVD = [8, 3, 0], !Reserved
S4R_MOD = [4, 4, 0], !Controller u CODE version
S4R_VER = [0, 4, 0], !Controller u CODE version

! Step four write SA register field reference

S4W_RSVD = [8, 8, 0], !Reserved
S4W_BURST = [2, 6, 0], !Max number longwords per NPR xfer
S4W_LF = [1, 1, 0], !Last fail request
S4W_GO = [0, 1, 0] !Go bit
tes,
! old declaration to define the interrupt
! indicator and purge words of the communications
! area.

HDR FIELD =
  set
| Header Word Ringbase  4
| RESERVE  [0, 0, 16, 0],
| Header Word Ringbase  3
| RSV  [1, 0, 8, 0],
| ADP CH  [1, 8, 8, 0],
| Header Word Ringbase  2
| CMD INT  [2, 0, 16, 0],
| Header Word Ringbase  1
| RSP_INT  [3, 0, 16, 0]
FIELD declaration to define the fields within
the send and receive ring descriptors:

DSC FIELD
  set
  
  ! Low order envelope address
  [O_ENPAD = [0, 0, 16, 0],
  ! High order 18 bit unibus or Qbus address
  [HI_ENPAD = [1, 0, 2, 0],
  ! QBus extension address
  [QB_EXT = [1, 2, 4, 0],
  ! Reserver field
  [RSRD = [1, 6, 8, 0],
  ! Flag b't
  [FLAG BIT = [1, 14, 1, 0],
  ! Ownership b't
  [OWN_BIT = [1, 15, 1, 0]
  yes.
field declaration to define the fields within
message envelope buffers.

ENV FIELD *
    set
    ! UQ Port envelope header field declaration
    !
    MSG_LENGTH = [0, 0, 16, 0].    ! Message length
    CREDITS = [1, 0, 4, 0].          ! Credits
    MSG_TYPE = [1, 4, 4, 0].         ! Message type
    CONN_ID = [1, 8, 8, 0].          ! Connection ID
DUP/MSCP command and response envelope header

field declarations

CMD_DIR = [2.0, 16.0], !Command Ref number low word
CMD_DIR = [3.0, 16.0], !Command Ref number high word
UNIT_NUM = [4.0, 16.0], !Unit selection field
UN_LUSED = [5.0, 16.0], !Unused low word
UN_HUSED = [6.0, 16.0], !Unused high word
TPIMSG = [6.0, 16.0], !End code message type
OPCODE = [6.0, 16.0], !Opcode
ENDCODE = [6.0, 16.0], !Opcode
ENDFLAG = [6.0, 16.0], !End message flag field
RSVD = [6.0, 16.0], !Reserved
STATUS = [7.0, 16.0], !Status
MODIFIER = [7.0, 16.0], !Modifier

DUP command and response envelope parameter

field declarations

ABORT command and response envelope parameter

field declarations

No parameters declared

response FIELD

No parameters declared

GET DUST STATUS command and response envelope

parameter Field declaration

COMMAND FIELD

No parameters declared

response FIELD

response FIELD

PLO_EXT = [8.0, 16.0], !Program Extension low word
PHI_EXT = [9.0, 16.0], !Program Extension high word
FLGS = [9.0, 16.0], !Flags
FLG_B0 = [9.0, 16.0], !Flag field bit 0
FLG_B1 = [9.0, 16.0], !Flag field bit 1
FLG_B2 = [9.0, 16.0], !Flag field bit 2
FLG_B3 = [9.0, 16.0], !Flag field bit 3
PLO_IND = [10.0, 16.0], !Progress indicator low word
PHI_IND = [11.0, 16.0], !Progress indicator high word
TIM_OUT = [12.0, 16.0], !Time out
EXECUTE SUPPLIED PROGRAM command and response
envelope parameter field declaration

COMMAND FIELD

BLO_CNT = [8, 0, 16, 0],
BHI_CNT = [9, 0, 16, 0],
BPA_LO = [10, 0, 16, 0],
BPA_HI = [11, 0, 16, 0],
OBUS_EXT = [11, 2, 4, 0],
RSV = [11, 6, 2, 0],
UBA_CHAN = [11, 8, 8, 0],
RSV0 = [12, 0, 16, 0],
RSV1 = [13, 0, 16, 0],
RSV2 = [14, 0, 16, 0],
RSV3 = [15, 0, 16, 0].

These next field definitions are the same
as above except they are for the overlay
buffer descriptors. To make life easy for
me I'll use the same names and just prefix
the names with a $ for uniqueness.

BPA_LO = [16, 0, 16, 0],
BPA_HI = [17, 0, 2, 0],
OBUS_EXT = [17, 2, 4, 0],
RSV = [17, 6, 2, 0],
UBA_CHAN = [17, 8, 8, 0],
RSV0 = [18, 0, 16, 0],
RSV1 = [19, 0, 16, 0],
RSV2 = [20, 0, 16, 0],
RSV3 = [21, 0, 16, 0].

response FIELD
No parameters declared

EXECUTE LOCAL PROGRAM command and response
parameter field declaration

COMMAND FIELD

PN.0 = [8, 0, 16, 0],
PN.1 = [9, 0, 16, 0],
PN.2 = [10, 0, 16, 0],
version = [8, 0, 16, 0],
TIME OUT = [9, 0, 8, 0],
FLAGS = [9, 8, 8, 0].

response FIELD
END DATA RECEIVE DATA command and response
parameter field declaration

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 declaration

COMMAND FIELD
MSCP_VER = [ 8, 0, 16, 0],
CTL_FLAGS = [ 9, 0, 16, 0],
HOST_TOV = [ 10, 0, 16, 0],
RSVD = [ 11, 0, 16, 0],
T4D.0 = [ 12, 0, 16, 0],
T4D.1 = [ 13, 0, 16, 0],
T4D.2 = [ 14, 0, 16, 0],
T4D.3 = [ 15, 0, 16, 0],
CDP_LO = [ 16, 0, 16, 0],
CDP_HI = [ 17, 0, 16, 0],

RESPONSE FIELD
MSCP_VER = [ 8, 0, 16, 0],
CTL_FLAGS = [ 9, 0, 16, 0],
CTL_TOV = [ 10, 0, 16, 0],
CSVRSN = [ 11, 0, 8, 0],
CMVRSN = [ 11, 8, 8, 0],
CID.0 = [ 12, 0, 16, 0],
CID.1 = [ 13, 0, 16, 0],
CID.2 = [ 14, 0, 16, 0],
CID.3 = [ 15, 0, 16, 0],
/* ONLINE COMMAND command and response 
   parameter field declaration */

/* COMMAND FIELD */
RSVD*  [ 8, 0, 16, 0], ![Reserved]
UNIT_FLAGS  [ 9, 0, 16, 0], ![Unit flag field]
RSVD10  [ 10, 0, 16, 0], ![Reserved field]
RSVD11  [ 11, 0, 16, 0], ![Reserved field]
RSVD12  [ 12, 0, 16, 0], ![Reserved field]
RSVD13  [ 13, 0, 16, 0], ![Reserved field]
RSVD14  [ 14, 0, 16, 0], ![Reserved field]
RSVD15  [ 15, 0, 16, 0], ![Reserved field]
DPD_LO  [ 16, 0, 16, 0], ![Device dependent parameter]
DPD_HI  [ 17, 0, 16, 0], ![Device dependent parameter]
SHADOW_UNIT  [ 18, 0, 16, 0], ![Shadow unit]
COPY_SPEED  [ 19, 0, 16, 0], ![Copy speed]

/* RESPONSE FIELD */
MULTI_UNIT_CODE  [ 8, 0, 16, 0], ![Multi-unit code]
UNIT_FLAGS  [ 9, 0, 16, 0], ![Same as cmd field]
RSVD10  [ 10, 0, 16, 0], ![Same as cmd field]
RSVD11  [ 11, 0, 16, 0], ![Same as cmd field]
UID_0  [ 12, 0, 16, 0], ![Unit ident word 0]
UID_1  [ 13, 0, 16, 0], ![Unit ident word 1]
UID_2  [ 14, 0, 16, 0], ![Unit ident word 2]
UID_3  [ 15, 0, 16, 0], ![Unit ident word 3]
MTD_LO  [ 16, 0, 16, 0], ![Media type ident word 0]
MTD_HI  [ 17, 0, 16, 0], ![Media type ident word]
SHADOW_UNIT  [ 18, 0, 16, 0], ![Same as cmd field]
SHA_STATE  [ 19, 0, 16, 0], ![Shadow state]
USZ_LO  [ 20, 0, 16, 0], ![Unit size lo word]
USZ_HI  [ 21, 0, 16, 0], ![Unit size hi word]
VSN_LO  [ 22, 0, 16, 0], ![Volume serial num lo word]
VSN_HI  [ 23, 0, 16, 0], ![Volume serial num hi word]

! Receive command buffer field definition

! REC_FIELD:
  set
  MSG_NUM = [0, 0, 12, 0],
  MSG_TYP = [0, 12, 4, 0],
  MSG_TXT = [1, 0, 16, 0]
  tes,

!

! Outstanding command buffer field declarations

! OUTIFIELD:
  set
  CMD_WRD = [0, 0, 16, 0],
  REC_FLG = [0, 15, 1, 0],
  CMD_REF = [0, 0, 8, 0],
  ENV_ADDR = [1, 0, 16, 0]
  tes,

! Command word ref "word 0 of slot"
! Command received indicator flag
! Command reference field
! Envelope address field
**linkage**

---

**Call linkage**

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Value return register, non-preserved</td>
</tr>
<tr>
<td>1 4</td>
<td>Preserved</td>
</tr>
<tr>
<td>5</td>
<td>Argument pointer</td>
</tr>
<tr>
<td>6</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>7</td>
<td>Program counter</td>
</tr>
</tbody>
</table>

`CALL_LNK$Typ = call (standard),`  

---

**Int linkage**

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Default usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-5</td>
<td>Preserved</td>
</tr>
<tr>
<td>6</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>7</td>
<td>Program pointer</td>
</tr>
</tbody>
</table>

`INT_LNK$Typ = interrupt (standard);`
CODES FOR SUPERVISOR SERVICE CALLS

CRESERV = 0,
CSET = 54,
CABSUB = 55,
CSEG = 56,
CIL = 53,
CDBU = 57,
CSES = 52,
CINIT = 9,
CICLEAN = 10,
CITPRI = 11,
CIPNB = 12,
CIPNX = 13,
CIPNF = 15,
CINLP = 16,
CIRLA = 17,
CBRK = 18,
CMESG = 19,
CIDRP = 20,
CIDRP = 21,
CGET = 22,
CGETW = 23,
CIGPL = 24,
CMEM = 25,
CEXIT = 26,
CRESET = 27,
COPEN = 28,
CICLDS = 29,
CICVET = 31,
CISVEC = 32,
CISPRI = 33,
CGRPD = 34,
CGMAN = 35,
CDLCP = 36,
CICFG = 37,
CIFLAG = 38,
CMAN = 40,
CIDDD = 41,
CIADL = 42,
CIERSF = 44,
CIERRDF = 45,
CIERRAW = 46,
CIERSOF = 47,
CIERR = 48,
CIAT = 49,
CICLK = 50,
CIQIO = 52;

!RESERVED FOR THE SUPERVISOR.
!BLISS EOET = SIGNIFY END OF EOET.
!BLISS EBSUB = SIGNIFY START OF SUBTEST.
!BLISS EBSUB = SIGNIFY END OF SUBTEST.
!BGNSEG = SIGNIFY START OF SEGMENT.
!BLISS EBSEG = SIGNIFY END OF SEGMENT.
!BLISS CKL = CHECK IF LOOP ON ERROR, WITHOUT A LABEL.
!READBA = READ TYPE OF BUS.
!BLISS ESCAPE = ESCAPE INNER BLOCK.
!ENDINIT = SIGNIFY END OF INITIALIZE CODE.
!ENDCLN = SIGNIFY END OF CLEAN-UP CODE.
!TRAPPRI = SET TRAP PRIORITY.
!PRINTB = PRINT BASIC EXPANSION OF ERROR Info.
!PRINTX = PRINT NON-BASIC EXPANSION OF ERROR INFO.
!PRINT = PRINT A STATISTICAL REPORT.
!PRINTF = PRINT A FORCED MESSAGE.
!INLOOP = TEST IF PROGRAM IS IN AN ERROR LOOP.
!IFLAGS = RETRIEVE FLAG SETTINGS OF THE IDR.
!BREAK = BREAK TO THE SUPERVISOR.
!ENDMSG = END A BLOCK OF ERROR REPORTING CODE.
!IDORP = CALL STATISTICAL REPORT CODE.
!ENDORP = SIGNIFY END OF STATISTICAL REPORT.
!GETWD = GET A WORD FROM AN XXDP DATA FILE.
!GETWP = GET A WORD FROM AN XXDP DATA FILE.
!GETPARS = GET PARAMETERS OF THE LOAD DEVICE.
!MEMORY = GET POINTER TO FREE MEMORY.
!BLISS EXIT = UNCONDITIONALLY EXIT TEST, SUBTEST, OR SEGMENT.
!RES = BUS RESET.
!OPEN AN XXDP DATA FILE FOR READ.
!CLOSE THE OPEN XXDP DATA FILE.
!CLVEC = RESTORE AN INTERRUPT VECTOR.
!SETVEC = SETUP AN INTERRUPT VECTOR.
!GETPRI = GET PRIORITY LEVEL.
!SETPRI = SET PRIORITY LEVEL.
!GPHAW = GET M.W. P-TABLE ADR. SPECIFIED
!GMAN = GET MANUAL INTERVENTION.
!DOCL = CALL CLEANUP CODE.
!CLRE = CLEAR EVENT FLAG.
!SET = SET EVENT FLAG.
!READ = READ EVENT FLAG.
!MANUAL = DETERMINE IF MANUAL INTERVENTION USED.
!IDDD = DROP UNITS.
!ENDADL = SIGNIFY END OF ADD UNITS CODE.
!END = SIGNIFY END OF DROP UNITS CODE.
!ERR = SYSTEM FATAL ERROR.
!ERRDF = DEVICE FATAL ERROR.
!ERRHAW = MEDIA HARD ERROR.
!ERRSOFT = MEDIA SOFT ERROR.
!ERROR = DIAG. MODIFIABLE ERROR CALL.
!ENDD = SIGNIFY END OF AUTO CODE.
!CLOCK = GET ADDRESS OF CLOCK TABLE.
!BIO CALLS.
literal

GET PARAMETER CODES

REQUEST TYPES (BITS 0, 1, 2)
GIPRM = 0,
GIPRMA = 1,
GIPRMD = 2,
GIDISP = 3,
GIXFFR = 4.

DEFAULT (BIT 3)
GANO = 0 * 8,
GAYES = 1 * 8.

RADIX (BITS 4, 5, 6)
GIRADD = 0 * 16,
GIRADD = 1 * 16,
GIRADD = 2 * 16,
GIRADD = 5 * 16,
GIRADD = 6 * 16.

OFFSET (BITS 9-15)
GOFFSET = 254,
GOFFSET = 1 * 256,
XOFFSET = 1 * 256.

COUNT (BIT 7)
GCTTOP = 1 * 128.

EXCEPTION (BIT 8)
EXCP = 1 * 256.
GLOLIM = 1,
GHILIM = 2.

TRANSFER CONDITIONS
XALWAYS = 0 * 16,
XTRUE = 1 * 16,
XFALSE = 2 * 16.

macro
ERROR =
LERROR (CERROR)
;

'Nkage
LERROR = trap:
macro ERRDF (ERNUM, MSGPTR, ROUT) =
begin
  builtin DECX;
  if length gtr 3
    then errormacro ('TOO MANY ARGUMENTS SPECIFIED')
  else
    DECX (C$ERDF, ERNUM, MSGPTR, ROUT);
end

macro ERRHDF (ERNUM, MSGPTR, ROUT) =
begin
  builtin DECX;
  if length gtr 3
    then errormacro ('TOO MANY ARGUMENTS SPECIFIED')
  else
    DECX (C$ERHDF, ERNUM, MSGPTR, ROUT);
end

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

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

end
MACRO FEQUAL
    BEGIN
        GLOBAL LITERAL
    \;
    FUNCTION LEVEL I/O DEFINITIONS
    \;
    Q.IOFL = 2,
    Q.IOFN = 4,
    Q.IOEF = 6,
    Q.IOEB = 10,
    Q.IOAE = 12,
    Q.IOPL = 14,
    IS.SUC = 1;
    END
    \;
MACRO EQUALS
    BEGIN
        GLOBAL LITERAL
    \;
    BIT DEFINITIONS
    \;
    BIT15 = \#01000000
    BIT14 = \#00100000
    BIT13 = \#00010000
    BIT12 = \#00001000
    BIT11 = \#00000100
    BIT10 = \#00000010
    BIT09 = \#00000001
    BIT08 = \#00000000
    BIT07 = \#00000000
    BIT06 = \#00000000
    BIT05 = \#00000000
    BIT04 = \#00000000
    BIT03 = \#00000000
    BIT02 = \#00000000
    BIT01 = \#00000000
    BIT00 = \#00000000
    \;
    BIT9 = BIT09
    BIT8 = BIT08
    BIT7 = BIT07
    BIT6 = BIT06
    BIT5 = BIT05
    BIT4 = BIT04
    BIT3 = BIT03
    BIT2 = BIT02
    BIT1 = BIT01
    BIT0 = BIT00
    \;
\;
SEQ Gee's
EVENT FLAG DEFINITIONS
EF52: EF17 RESERVED FOR SUPERVISOR TO PROGRAM COMMUNICATION

EF START = 32.
EF RESTART = 31.
EF CONTINUE = 30.
EF NEW = 29.
EF PWR = 28.

START COMMAND WAS ISSUED
RESTART COMMAND WAS ISSUED
CONTINUE COMMAND WAS ISSUED
A NEW PASS HAS BEEN STARTED
A POWER-FAIL/POWER-UP OCCURRED

PRIORITY LEVEL DEFINITIONS
PRI07 = #o340.,
PRI06 = #o300.,
PRI05 = #o240.,
PRI04 = #o200.,
PRI03 = #o140.,
PRI02 = #o100.,
PRI01 = #o40.,
PRI00 = #o0.

OPERATOR FLAG BITS
EVL = #o4.,
LDT = #o10.,
ADR = #o20.,
IDU = #o40.,
ISR = #o100.,
UAM = #o200.,
BOE = #o400.,
PNT = #o1000.,
PRI = #o2000.,
IXE = #o4000.,
IBE = #o10000.,
IER = #o20000.,
LOE = #o40000.,
HDE = #o100000.

MACRO
BREAK =
T$RAP (CIBRK)

MACRO
BRESET =
T$RAP (CIRESET)
macro
  DELAY (MULT) =
  begin
    external L$CPU, L$DLY;
    local $TMP2;
    $TMP2 = L$CPU + MULT;
    while $TMP2 neq 0 do
      begin
        decru $TMP1 from L$DLY to 1 do
          begin
            local $TMP : volatile;
            $TMP = 0;
            end;
        end;
        $TMP2 = $TMP2 1;
      end;
  end;

macro
  FLAGS (LOC) =
    LOC = L$FLAGS (C$FLA)
  *;

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

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

  linkage
    L$READBUS = trap : VALUECBIT clearstack;

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

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

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

  linkage
    L$INLOOP = trap : VALUECBIT clearstack;
macro
"MANUAL =
(L$MANUAL (C$MANI))"
#
make
L$MANUAL : trap : VALUEBIT clearstack;
#
macro
CKLOO =
"f (TSRAPRO (C$CLP1)) then
leave T$STAG ;
#
macro
ESCAPE =
"f (TSRAPRO (C$ESCAPE)) then
leave T$STAG ;
#
macro
EXIT
begin
TSRAPRO (C$EXIT);
leave T$STAG ;
end;
#
macro
EXIT_TST =
begin
TSRAPRO (C$EXIT);
return;
end;
#
compileme
T$BGNST = 0,
T$BGNSUB = 0,
T$BGNSEG = 0,
T$STNUM = 1,
T$SUBNUM = 1,
T$SEGNUM = 1,
T$PUSHLEV = 0,
T$POPLEV = 0,
T$TEMP = 0,
T$CODE = 0,
C$COUNT = 0;

!THIS FLAG IS SET AT BGNST AND CLEARED AT ENDST
!THIS FLAG IS SET AT BGNSUB AND CLEARED AT ENDSUB
!THIS FLAG IS INCREMENTED AT BGNSEG AND DECREMENTED AT ENDSSEG
!THIS VARIABLE IS USED TO GENERATE THE TEST LABEL
!THIS VARIABLE IS USED TO GENERATE THE SUBROUTINE LABEL
!THIS VARIABLE IS USED TO GENERATE THE SEGMENT LABEL
!THIS VARIABLE IS USED WITHIN SEGMENT NESTINGS
!THIS VARIABLE IS USED WITHIN SEGMENT NESTINGS
!THIS VARIABLE IS USED TO GENERATE GPRM'S/GMANI'S
!COUNTER USED IN VARIOUS MACROS
macro
    BGNINI :
        routine LINIT : novalue :
            begin
                #;
            end;

    macro
        BGNAU :
            routine LAU : novalue :
                begin
                    #;
                end;

    macro
        BGNAUTO :
                routine LAUTO : novalue :
                        begin
                            #;
                        end;

    macro
        BGNCLN :
                routine LCLEAN : novalue :
                        begin
                            #;
                        end;

    macro
        BGNDU :
                routine LDU : novalue :
                        begin
                            #;
                        end;

    macro
        BGNRT :
                routine LRPT : novalue :
                        begin
                            #;
                        end;

    macro
        LASTAD :
            psect
code = $XYZ$;

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

`bind1up (NUM)`

```assembly
  global bind
  tpsnum = NUM;
```

**compiletime**

`psptr = 0, THIS USED TO BE TRUE BUT CALCULATIONS FOR POINTERS WERE OFF`

```assembly
  psptr = 4,
  tpsnum = 0;
  assign (tspnum, NUM)
  psect plt + $x1z$;
```

**macro**

`endsetup`

```assembly
  if tpsnum neq 0
    then errormacro ("ERROR IN PTABLE BLOCK")
    global tpsfree : vector [1]
      psect ($x1z$) initial (0);
    this location may get overwritten with no problem
  end;
```

**macro**

`bgntab`

```assembly
  assign (tspnum, tspnum + 1)
  ptabs ()
```

**macro**

`ptabs[]`

```assembly
  assign (psptr, psptr + (2 * (#length + 4)))
  this used to be true but calculations for pointers were off
  assign (psptr, psptr + (2 * (#length + 2)))
  if tpsnum eq 2
    then
      bind $lasl = uplit (0);
      else
      bind $name ("$lasl", $number (tpnum)) = uplit (lslast + psptr);
    then
      bind $name ("$rem", $number (tpnum)) = plt (#remaining);
```

**SEQ 0254**
macro
ENDTAB -
assign (PTNUM, PTNUM 0)

; ; ; ; ; ; ; ;

macro $STAG -, name (((ST), nnumber ($TSTNUM), nnumber ($SUBNUM), nnumber ($SEGNUM))

; ; ; ; ; ; ; ;

macro BGNST,
, if ($BGNSUB neq 0
        then Errormacro ('''BGNST'' IN SUB')
        exitmacro
, if ($BGNSEG neq 0
        then Errormacro ('''BGNST'' IN SEG')
        exitmacro
, if ($BGNST neq 0
        then Errormacro ('''BGNST'' IN TST')
        exitmacro

; ; ; ; ; ; ; ;
assign ($PUSHLEV, 0)
; ; ; ; ; ; ; ;
assign ($PUSHLEV, 8)
; ; ; ; ; ; ; ;
assign ($SEGNUM, 0)
; ; ; ; ; ; ; ;
assign ($SEGNUM, 0)
; ; ; ; ; ; ; ;
assign ($BGNST, 1)

routine name ('''ST'', nnumber ($TSTNUM)); novalue *
begin
label $STAG;
$STAG: begin

; ; ; ; ; ; ; ;

macro BGNSEG,
; ; ; ; ; ; ; ;
assign ($PUSHLEV, $PUSHLEV + 1)
; ; ; ; ; ; ; ;
assign ($SEGNUM, $SEGNUM + 1)
; ; ; ; ; ; ; ;
assign ($SEGNUM, $SEGNUM + 1)
; ; ; ; ; ; ; ;
assign ($BGNSEG, 1)
; ; ; ; ; ; ; ;
if ($PUSHLEV gr 8
        then Errormacro ('''TOO MANY ''BGNSEG''S'')

        do begin
            label $STAG;
            $STAG: begin
            $STAGPRO (C$SEG);
macro BGNSUB
  
  if $BGNSUB neq 0
  then
    errormacro ( BGNSUB IN SEG )
  endmacro
  
  if $BGNSUB neq 0
  then
    errormacro ( BGNSUB IN SUB )
  endmacro
  
  assign ($BGNSUB, 1)
  assign ($SUBNUM, $SUBNUM + 1)
  do begin
    label $TAG:
    $TAG: begin
    TIRAPRO (C$SUB);
  end:

macro BGNSMG (NAME) |
  fortran routine $name (M$+, NAME) : novalue |
  global routine NAME : novalue |
  begin |
  $name (M$+, NAME) (); |
  LMSG (C$MSG); |
  end |
  routine $name (M$+, NAME) : novalue |
  begin |
  end:

linkage
  LMSG : trap:

macro BGNSRV (ISR) |
  global routine ISR (OLDPSW, OLDPC) : L$ISR novalue |
  begin |
  end:

linkage
  LISR : interrupt:

macro ENDSRV |
  end:
macro
  ENADV MODPRI (PRIORIT) *
  
  *f PRIORIT gtr 7
  then
    errormacro (PRIORIT MUST BE 0 TO 7)
  else
    endmacro
  else
    OLOPSW <5,3> = PRIORIT;
  end
  ENDSRV

end

macro
  ENDMGR

end

macro
  ENDTINIT
  global routine L$INIT : novalue *
  begin
    LINIT ();
    TSRAP (C$INIT);
  end

end

macro
  ENDNAU
  global routine L$AU : novalue *
  begin
    LAU ();
    TSRAP (C$AU);
  end

end

macro
  ENDAUTO
  global routine L$AUTO : novalue *
  begin
    LAUTO ();
    TSRAP (C$AUTO);
  end

end

macro
  ENDCALLN
  global routine L$CLEAN : novalue *
  begin
    LCLEAN ();
    TSRAP (C$CLEAN);
  end

end
macro
ENDDU:
end;
global routine LIDU : novalue :=
begin
LIDU (;
1TRAP (C\$DU));
end;
*

macro
ENDRPT:
end;
global rout'ne LIRPT : novalue :=
begin
LRPT ();
1TRAP (C\$RPT);
end;
*

macro
ENDTST :=
endif #BGNTST eq 0
then
errormacro ('MISSING "BGNTST"')
exitmacro
endif;
end

global routine name ('T', #number (#TSTNUM)) : novalue :=
do
name ('T', #number (#TSTNUM)) ()
while 1TRAP (C\#ETS1);
assign (#TSTNUM, #TSTNUM + 1)
assign (#BGNTST, 0)
if (#PUSHLEV + #POPLEV) 8 neq 0
then errormacro ('TEST CONTAINS UNEQUAL "BGNSEG"S AND "ENDSEG"S')
endif;

macro
ENDSUB :=
endif #BGNSUB eq 0
then
errormacro ('MISSING "BGNSUB"')
exitmacro
endif;
end
while (1TRAPRO (C\$ESUB));
assign (#BGNSUB, 0)
*
macro
   ENDSEL
   `assign ($POPELV, $POPELV 1)
   `if $POPELV <= 0
   `then `errormacro ('TOO MANY 'ENDSEG'S')
   `else

   THE TEST FOR PROPER NESTING IS IN MACRO 'ENDTST'
   `end
   `end
   while ($TRAPRO ($ESESEG))
   `assign ($BGNSEG, 0)
   `end

linkage
   TRAPRO = trap (register = 0);

macro
   SETVEC (VECADD, ROUT, PRIOR) =
   L1SETVEC ($C$VEC, PRIOR, ROUT, VECADD, 3)
   `end

linkage
   L1SETVEC = trap (standard, standard, standard, standard, standard);

macro
   SETPRI (PRIOR) =
   L1SETPRI ($C$PRI, PRIOR)
   `end

linkage
   L1SETPRI = trap (register = 0);

macro
   CLRVEC (VECADD) =
   L1CLRVEC ($C$VEC, VECADD)
   `end

linkage
   L1CLRVEC = trap (register = 0);

macro
   PRINTB [] =
   begin
   `builtin sp;
   `name ('DERR', `length) ($C$NB, REVERSEACTUALS ($remaining), `length, .sp);
   `end
   `end
makro
  PRINT'x []
  begin
    builtin sp;
    $name ('DIERR', $length) (C$PNTX, REVERSEACTUALS ($remaining), $length, .sp);
  end

makro
  PRINS []
  begin
    builtin sp;
    $name ('DIERR', $length) (C$PNTS, REVERSEACTUALS ($remaining), $length, .sp);
  end

makro
  PRINTF []
  begin
    builtin sp;
    $name ('DIERR', $length) (C$PNTF, REVERSEACTUALS ($remaining), $length, .sp);
  end

makro
  MAKE A ARGUMENT LIST THAT IS THE REVERSE OF THE ONE PASSED IN.

  REVERSEACTUALS (A) [] =
  $if $length eq 1
    $then A
  $else REVERSEACTUALS ($remaining), A
  $endif

linkage
  DIERR1 = trap (standard, standard, register = 0);

linkage
  DIERR2 = trap (standard, standard, standard, register = 0);

linkage
  DIERR3 = trap (standard, standard, standard, standard, register = 0);

linkage
  DIERR4 = trap (standard, standard, standard, standard, standard, register = 0);

linkage
  DIERR5 = trap (standard, standard, standard, standard, standard, standard, register = 0);

linkage
  DIERR6 = trap (standard, standard, standard, standard, standard, standard, standard, register = 0);
`LINKAGE`  
`DIERR7: trap (standard, standard, standard, standard, standard, standard, standard, register = 0);`

`LINKAGE`  
`DIERR8: trap (standard, standard, standard, standard, standard, standard, standard, standard, register = 0);`

`LINKAGE`  
`DIERR9: trap (standard, standard, standard, standard, standard, standard, standard, standard, standard, register = 0);`

`MACRO`  
`GPHARD (UNIT, POINTER) = (POINTER - LIGPHARD (C1GPARD, UNIT));`

`LINKAGE`  
`LIGPHARD: trap (register = 0);`

`MACRO`  
`CLOSE = TIRAP (C$CLOS);`

`MACRO`  
`OPEN (FILENAME) = D$FILE (C$OPEN, FILENAME);`

`MACRO`  
`GETWORD (DEST) = GET_DATA (C$GETw, DEST);`

`MACRO`  
`GETBYTE (DEST) = GET_DATA (C$GETb, DEST);`

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

`LINKAGE`  
`D$GET: trap (register = 0); VALUECBIT clearchk;`
macro DEFPR1 (PRI0)
  PRI0 = ($FILE ($GROUP))
end:

macro DORPT
  L$DORPT ($DORPT)
end:

 linkage L$DORPT = trap:

macro DODU (UNIT):
  L$DODU ($DODU, UNIT)
end:

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

macro DOCLN
  L$DOCLN ($DOCLN)
end:

 linkage L$DOCLN = trap:

macro READEF (EFN):
  L$READEF ($REFG, EFN)
end:

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

macro CLOCK (TYPE, POINTER): if identical (TYPE, L)
  then M$CLK (POINTER, #o'114')
  else exitmacro
else if identical (TYPE, P)
  then M$CLK (POINTER, #o'120')
  else exitmacro
else errormacro ('TYPE MUST BE ' OR P')
end.
macro MISCLK (POINTER, KIND) *
    begin
        if MISCLK (CICLCK, KIND) = 1 then (builtin r0; POINTER * .r0; 1)
        else (0)
    end

1'\text{linag}
LSCLOCK = trap (register * 0): VALUECBIT clearstack;

1'\text{linkage}
DISFILE = trap (register * 0);

1'\text{linkage}
TSRAP - trap;

macro GMANIL (MSGADR, DATADR, MASK, DFLT) *
    begin
        builtin DECX;
        assign (TICODE, GIFRML or G1RADL)
        MIDFLT (DFLT, TITEMP);
        assign (TICODE, TICODE or TITEMP)
        DECX (C$GMAN, $0\text{404}', DATADR, TICODE, MSGADR, MASK);
    end

macro GMANID (MSGADR, DATADR, RADIX, MASK, LOW, HIGH, DFLT) *
    begin
        builtin DECX;
        assign (TICODE, GIPRMD)
        M$RAD (RADIX, TITEMP);
        assign (TICODE, TICODE or TITEMP)
        MIDFLT (DFLT, TITEMP);
        assign (TICODE, TICODE or TITEMP)
        DECX (C$GMAN, $0\text{406}', DATADR, TICODE, MSGADR, MASK, LOW, HIGH);
    end
macro CMNIA (MSGADR, DATADR, RADIX, LOW, HIGH, DFLT) *
begin
    builtin DECX;
    if identical (A, RADIX) then errormacro ('INVALID RADIX')
    if assign (TICODE, GIPRMA) then assign (TICODE, TICODE) or ITEMP)
    if assign (TICODE, TICODE or ITEMP) then assign (TICODE, TICODE or ITEMP)
    assign (TICODE, TICODE or ITEMP)
    DECX (CIGMAM, #0 405, DATADR, TICODE, MSGADR, LOW, HIGH);
end
*

macro MIRAD (IN, OUT) *
if identical (IN, B) then
    assign (OUT, GIRADB)
    exitmacro
if identical (IN, D) then
    assign (OUT, GIRADD)
    exitmacro
if identical (IN, L) then
    assign (OUT, GIRADL)
    exitmacro
if identical (IN, A) then
    assign (OUT, GIRADA)
    exitmacro
if errormacro ('ILLEGAL RADIX')
*

macro MIDFLT (DFLT, OUT) *
if identical (YES, DFLT) then assign (OUT, GYES)
else if identical (NO, DFLT) then assign (OUT, GNO)
else errormacro ('INCORRECT DEFAULT')
*

macro
BGNPROT (A, B, C) =
   initial (A, B, C);
end:

macro
ENDPROT
end:

macro
DISPATCH (DSNUM) =
global L*PCNT : vector [1] psect ($code$)
   initial (DSNUM);
   assign (C*TSCT, 1)
   FORDEL (DSNUM)
   !DECLARE ROUTINE NAMES FORWARD OR 'EXTERNAL
   assign (C*TSCT, 1)
global L*DISPATCH : vector [DSNUM] psect ($code$)
   initial (LOADS (DSNUM))
end:

macro
LOADS (TSNMS) [ ] =
   assign (C*IITCNT, TSNMS)
   name (T, $number (C*TSCT))
   assign (C*IITCNT, C*IITCNT - 1)
   assign (C*TSCT, C*TSCT + 1)
   if $number (C*IITCNT) eq 0
      then exitmacro
   else
      LOADS (C*IITCNT)
end:
macro
FORDEC (T$NMS) [] :
    $assign (C$ITCNT, T$NMS)
    @ external routine $name (\$IT, $number (C$ITSCF)) : novalue:
    $assign (C$ITCNT, C$ITCNT + 1)
    $if $number (C$ITCNT) eq 0

    $then $exitmacro
    $else FORDEC (C$ITCNT)
    $end_

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

@
macro

$POINT (PTRA)[] =

if $identical (ALL, PTRA)
  assign (OSBNSFT, 1)
  assign (OSBNSPT, 1)
  assign (OSGSW, 1)
  assign (OSAPS, 1)
  assign (OSAU, 1)
  assign (OSDU, 1)
  assign (OHERRLH, 1)
  assign (OSSETUP, 1)
  assign (OSPINTER, 1)
  return

else:
  if $identical (NONE, PTRA)
    assign (OSPINTER, 1)
    return
  else:
    if $identical (SFT, PTRA)
      assign (OSBNSFT, 1)
      assign (OSPINTER, 1)
      return
    else:
      if $identical (APT, PTRA)
        assign (OSBNSPT, 1)
        assign (OSPINTER, 1)
        return
\texttt{if} \texttt{identical} (\texttt{APTS}, \texttt{PNTR})
\texttt{then}
\texttt{assign} (\texttt{O1APTS}, \texttt{1})
\texttt{assign} (\texttt{O1POINTER}, \texttt{1})
\texttt{endif}
\texttt{if} \texttt{identical} (\texttt{AU}, \texttt{PNTR})
\texttt{then}
\texttt{assign} (\texttt{O1AU}, \texttt{1})
\texttt{assign} (\texttt{O1POINTER}, \texttt{1})
\texttt{endif}
\texttt{if} \texttt{identical} (\texttt{DU}, \texttt{PNTR})
\texttt{then}
\texttt{assign} (\texttt{O1DU}, \texttt{1})
\texttt{assign} (\texttt{O1POINTER}, \texttt{1})
\texttt{endif}
\texttt{if} \texttt{identical} (\texttt{IBL}, \texttt{PNTR})
\texttt{then}
\texttt{assign} (\texttt{O1IBL}, \texttt{1})
\texttt{assign} (\texttt{O1POINTER}, \texttt{1})
\texttt{endif}
\texttt{if} \texttt{identical} (\texttt{SETUP}, \texttt{PNTR})
\texttt{then}
\texttt{assign} (\texttt{O1SETUP}, \texttt{1})
\texttt{assign} (\texttt{O1POINTER}, \texttt{1})
\texttt{endif}
\texttt{assign} (\texttt{O1POINT}, \texttt{#remaining})

MACRO

HEADER (FILNAM, REVLEV, DEPO, TIME, TYPE, PRI) -

IFDEF DPINTER eq 0
  THEN
    #errormacro ('MISSING POINTER MACRO')
  END
ENDIF

EXTERNAL

!TO RESOLVE REFERENCES IN OTHER SKELS
LISTOK, TPThY, L*lRPT, L*INIT, L*CLEAN,
L*LAST, L*HARD, L*DIVTP, L*DESC, L*DU, L*AU, L*AUTO;
COMPLETIME

C*REVISION = 3,
C*EDIT = 3;

FORWARD

ERRTYP : vector [1],
L*SWLEN : vector [1],
L*HMLEN : vector [1],
L*PROT : vector [3],
L*DISPATCH : vector [DSNR deficits OF TESTS];
GLOBAL BIND

L*ERRBL = ERRTYP,
L*SW = L*SWLEN + 2,
L*HM = L*HMLEN + 2;
COMPLETIME EMT, ELOAD = #o'104035';
#assign (C*COUNT, 8) #charcount (FILNAM));
IFDEF #charcount (FILNAM) gtr 7
  THEN
    #errormacro ('NAME TOO BIG')
  END
ENDIF

IFDEF ('FILNAM, rep C*COUNT of byte (0));
GLOBAL L*REV : vector [1] psect ('code');
  INITIAL (byte (REVLEV), byte (DEPO));
GLOBAL BIND L*DEPO = L*REV = 1;
GLOBAL L*INIT : vector [1] psect ('code');
  INITIAL (IFDEF O*SETUP eq 0
    THEN 0
    ELSE TPThY
  )
GLOBAL L*TIML : vector [1] psect ('code');
  INITIAL (TIME);
GLOBAL L*$HCP : vector [1] psect ('code');
  INITIAL (L*HARD);
GLOBAL L*$PCP : vector [1] psect ('code');
  INITIAL (IFDEF O*BGNSFT eq 0
    THEN 0
    ELSE L*SOFT
  )
GLOBAL L*$TPP : vector [1] psect ('code');
  INITIAL (L*HMW);
GLOBAL L*$SPP : vector [1] psect ('code');
  INITIAL (IFDEF O*GNSW eq 0
    THEN 0
    ELSE L*SW
  )
ENDIF;
global LSLADP : vector [1] psect ($\text{code}$)
  initial (LSLAST);
global LSISTA : vector [1] psect ($\text{code}$)
  initial (0);
global LICO : vector [1] psect ($\text{code}$)
  initial (0);
s if (1 - TYPE) neq 0
  then s if TYPE neq 0
  then
    ererrormacro ('DIAG. TYPE MUST BE 0 OR 1')
    errormacro
  s fi:
  s fi;
global L$\text{DTYPE}$ : vector [1] psect ($\text{code}$)
  initial (TYPE);
global L$\text{DIPT}$ : vector [1] psect ($\text{code}$)
  initial (0);
global L$\text{DTP}$ : vector [1] psect ($\text{code}$)
  initial (LIDISPATCH);
global L$\text{DPRIO}$ : vector [1] psect ($\text{code}$)
  initial (PRI);
global L$\text{ENVI}$ : vector [1] psect ($\text{code}$)
  initial (0);
global L$\text{EXP1}$ : vector [1] psect ($\text{code}$)
  initial (0);
global L$\text{MPREV}$ : vector [1] psect ($\text{code}$)
  initial (byte (C\$REVISION), byte (C\$EDIT));
global L$\text{REF}$ : vector [2] psect ($\text{code}$)
  initial (0, 0);
global L$\text{SPC}$ : vector [1] psect ($\text{code}$)
  initial (0);
global L$\text{DEVP}$ : vector [1] psect ($\text{code}$)
  initial (LID$\text{DTYPE}$);
global L$\text{RPP}$ : vector [1] psect ($\text{code}$)
  initial ($\text{s f O$\text{BNGRP}$ eq 0$
  \text{then 0
  \text{else L$\text{EXP1}$}$
  s fi);}$
global L$\text{EXP4}$ : vector [1] psect ($\text{code}$)
  initial (0);
global L$\text{EXPS}$ : vector [1] psect ($\text{code}$)
  initial (0);
global L$\text{AUT}$ : vector [1] psect ($\text{code}$)
  initial ($\text{s f OS$\text{AU}$ eq 0
  \text{then 0
  \text{else L$\text{AUT}$}$
  s fi);}$
global L$\text{OUT}$ : vector [1] psect ($\text{code}$)
  initial ($\text{s f GI$\text{DU}$ eq 0$
  \text{then 0
  \text{else L$\text{OUT}$}$
  s fi);}$
global L$\text{LDUN}$ : vector [1] psect ($\text{code}$)
  initial (0);
global L$\text{DESP}$ : vector [1] psect ($\text{code}$)
  initial (L$\text{DESC}$);
global L$\text{LOAD}$ : vector [1] psect ($\text{code}$)
  initial (EMT.ES$\text{LOAD}$);
global L$10TP : vector [1]
  initial (if OHEARTBL eq 0
    then 0
    else L$10TP
  );

  global L$10CP : vector [1]
  initial (L$10INIT);
  global L$10CPP : vector [1]
  initial (L$10CLEAN);
  global L$10ACP : vector [1]
  initial (L$10AUTO);
  global L$10PR : vector [1]
  initial (L$10PROT);
  global L$10TEST : vector [1]
  initial (0);
  global L$10OLY : vector [1]
  initial (0);
  global L$10MILE : vector [1]
  initial (0);
  global L$10CPU : vector [1]
  initial (1);

%:

:=

; compulsative

X$FER_SKIP = #0'001004',

T$HILIM = 0,
T$HOLIM = 0,
T$EXCP = 0,
C$ATLO = 0,
GP$COUNT = 0,
C$AETHI = 0;

\text{macro}

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

%:

\text{macro}

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

%:

\text{macro}

XFER (LAB) =
  M$XFER (LAB, X$ALWAYS);

%:
macro XFER (LAB) :
  M$FER (LAB, X$FALSE)
#
macro XFER (LAB) :
  M$FER (LAB, X$TRUE)
#
macro M$FER (LAB, FLAVOR) :
  forward $name ('$L', LAB);
  ! GENERATE THE XFER :
  psect own * $code$;
  own $name ('$L', LAB);
  initial ((($name ('$L', LAB) - $name ('$S', LAB)) * X$OFFSET) +
            (G$FER + FLAVOR)),
  psect own * $own$;
#
macro $L (LAB) :
  ! GEN A LABEL FOR OFFSET CALC AND FILL IT
  ! WITH A NO OP
  psect own * $code$;
  own $name ('$L', LAB) : initial (X$FER SKIP);
#
macro BGNHARD :
  forward L$NHARD : vector[1];
  global L$HARDLN : vector[1] psect ($code$)
    initial (((L$NHARD - L$HARDLN) / 2) 1);
  global bind L$HARD = L$HARDLN + 2;
#
macro BGNSFT :
  forward L$NSFT : vector[1];
  global L$SFTLN : vector[1] psect ($code$)
    initial (((L$NSFT - L$SFTLN) / 2) 1);
  global bind L$SFT = L$SFTLN + 2;
#
macro ENDHARD :
  global L$NHARD : vector[1] psect ($code$);
macro
  GPRMA (MSGADR, DATADR, RADIX, LOW, HIGH, DFLT, COUNT) *
  "assign (GP$ COUNT, GP$ COUNT + 1) !GEN. UNIQUE NAMES
  "assign (T$EXCP, 0)
  "assign (T$CODE, 0)
  "assign (C$COUNT, 4) !4 ARGS, COUNT IS 'OPTIONAL
  NO.A RADIX (RADIX);
  NO.ODD ADR (DATADR);
  OFFSET SIZE (DATADR);
  "assign (T$CODE, GPRMA * (DATADR + G$OFFSET))
  MSRAD (RADIX, T$TEMP);
  "assign (T$CODE, T$CODE or T$TEMP)
  MIDFLY (DFLT, T$TEMP);
  "assign (T$CODE, T$CODE or T$TEMP)
  M$EXCP (LOW, HIGH) !LOAD LOW AND HIGH LIMITS.
  "if COUNT gtr 1
    "assign (T$CODE, T$CODE or G$CNTOP)
    "assign (C$COUNT, C$COUNT + 1)
  "endif
  global $name ( GP$, $number (GP$ COUNT)); vector [C$COUNT] psect ($code$)
  if (C$ATLO or C$ATHI) neq 0
    "if COUNT gtr 1
      "if (COUNT / 2);
      "else
      "endif
    "endif
  "assign (C$ATLO, 0) !
**macro**

GPRML (MSGADR, DATADR, MASK, DFLT, COUNT) *
*assign (GP$ COUNT, GP$ COUNT + 1) !GEN. UNIQUE NAMES
*assign (T$EXCP, 0)
*assign (T$CODE, 0)
*assign (C$COUNT, 5)
NO A RADIX (RADIX);
NO ODD ADR (DATADR);
OFFSET SIZE (DATADR);
*assign (T$CODE, GP$PRML * (DATADR + G$OFFSET))
M$RAD (T$CODE, T$TEMP);
*assign (T$CODE, T$CODE or T$TEMP)
M$DPLT (DFLT, T$TEMP);
*assign (T$CODE, T$CODE or T$TEMP)
M$EXCP (LOW,HIGH) !LOAD LOW AND HIGH LIMITS.
*'f COUNT gr 1
*then
*assign (T$CODE, T$CODE or C$CNTOP)
*assign (C$COUNT, C$COUNT + 1)
*'*
global *name ('GP$', #number (GP$ COUNT)): vector [C$COUNT] psect ($code$)
initial (T$CODE, MSGADR, MASK, T$LOLIM, T$HILIM
*if (C$THLO or C$THHI) neq 0
*then . T$EXCP
*'*
*if COUNT gr 1
*then . (COUNT / 2));
*else);
*'*
*assign (C$THLO, 0)
*assign (C$THHI, 0)
*'*

**macro**

GPRML (MSGADR, DATADR, MASK, DFLT, COUNT) *
*assign (GP$ COUNT, GP$ COUNT + 1) !GEN. UNIQUE NAMES
*assign (T$EXCP, 0)
*assign (T$CODE, 0)
*assign (C$COUNT, 3)
NO A RADIX (RADIX);
OFFSET SIZE (DATADR);
*assign (T$CODE, GP$PRML * (DATADR + G$OFFSET))
M$RAD (L, T$TEMP);
*assign (T$CODE, T$CODE or T$TEMP)
M$DPLT (DFLT, T$TEMP);
*assign* (T$CODE, T$CODE or T$TEMP)
*'* COUNT gr 1
*then
*assign (T$CODE, T$CODE or C$CNTOP)
*assign (C$COUNT, C$COUNT + 1)
*'*
global *name ('GP$', #number (GP$ COUNT)): vector [C$COUNT] psect ($code$)
initial (T$CODE, MSGADR, MASK;
*if COUNT gr 1
*then . (COUNT / 2));
*else);
*'*
*assign (C$THLO, 0)
*assign (C$THHI, 0)
*'*
MACRO M$EXCP (LOW, HIGH) :
  IF (C$ATLO or C$ATHI) neq 0
    ASSIGNED (C#CODE, C#QUE or G$EXCP)
    ASSIGNED (C#COUNT, C#COUNT + 1)
  ELSE
    IF C$ATLO neq 0
      THEN
        ASSIGNED (T$EXCP, T$EXCP or G$LOLIM)
        ASSIGNED (T$LOLIM, (LOW / 2))
      ELSE
        ASSIGNED (T$LOLIM, LOW)
    ELSE
      IF C$ATHI neq 0
        THEN
          ASSIGNED (T$EXCP, T$EXCP or G$HILIM)
          ASSIGNED (T$HILIM, (HIGH / 2))
        ELSE
          ASSIGNED (T$HILIM, HIGH)
  END
ENDIF

MACRO GP$ATLO (OFFSET) :
  ASSIGN (C$ATLO, 1)
  OFFSET
ENDIF

MACRO GP$ATHI (OFFSET) :
  ASSIGN (C$ATHI, 1)
  OFFSET
ENDIF

MACRO NO_A_RADIX (RADIX) :
  IF identical (A, RADIX)
    THEN errormacro ('INVALID RADIX')
  END
ENDIF

MACRO NO_ODD_ADDR (OFFSET) :
  ASSIGNED (T$TEMP, OFFSET and 1) !YE OLD MASK
  IF T$TEMP neq 0
    THEN errormacro ('OFFSET IS ODD')
  END
offset size (offset) =
if (g@fsize offset) < 0
  errormacro ("OFFSET TOO BIG")
endif

macro DISPLAY (arg) =
global gp@disp : vector [2] psect (#code#)
  initial (gp@disp, arg);
end