```
EEEEEEEEEEEEEEEEE   RRRRRRRRRRRR       FFFFFFFFFFFFFFFF
EEEEEEEEEEEEEEEEE   RRRRRRRRRRRR       FFFFFFFFFFFFFFFF
EEEEEEEEEEEEEEEEE   RRRRRRRRRRRR       FFFFFFFFFFFFFFFF
EEE                 RRR        RRR     FFF
EEE                 RRR        RRR     FFF
EEE                 RRR        RRR     FFF
EEE                 RRR        RRR     FFF
EEE                 RRR        RRR     FFF
EEE                 RRR        RRR     FFF
EEEEEEEEEEEE        RRRRRRRRRRRR       FFFFFFFFFFFF
EEEEEEEEEEEE        RRRRRRRRRRRR       FFFFFFFFFFFF
EEEEEEEEEEEE        RRRRRRRRRRRR       FFFFFFFFFFFF
EEE                 RRR    RRR         FFF
EEE                 RRR    RRR         FFF
EEE                 RRR    RRR         FFF
EEE                 RRR      RRR       FFF
EEE                 RRR      RRR       FFF
EEE                 RRR      RRR       FFF
EEEEEEEEEEEEEEEE    RRR        RRR     FFF
EEEEEEEEEEEEEEEE    RRR        RRR     FFF
EEEEEEEEEEEEEEEE    RRR        RRR     FFF
```

```
MM      MM  EEEEEEEEEE  MM      MM    000000    RRRRRRR    YY      YY   SSSSSSSS
MM      MM  EEEEEEEEEE  MM      MM   000000     RRRRRRR    YY      YY   SSSSSSSS
MMMM  MMMM  EE          MMMM  MMMM  00    00    RR    RR   YY      YY   SS
MMMM  MMMM  EE          MMMM  MMMM  00    00    RR    RR   YY      YY   SS
MM  MM  MM  EE          MM  MM  MM  00    00    RR    RR     YY  YY     SS
MM  MM  MM  EE          MM  MM  MM  00    00    RR    RR     YY  YY     SS
MM      MM  EEEEEEEE    MM      MM  00    00    RRRRRRRR       YY       SSSSSS
MM      MM  EEEEEEEE    MM      MM  00    00    RRRRRRRR       YY       SSSSSS
MM      MM  EE          MM      MM  00    00    RR  RR         YY             SS
MM      MM  EE          MM      MM  00    00    RR  RR         YY             SS
MM      MM  EE          MM      MM  00    00    RR    RR       YY             SS
MM      MM  EE          MM      MM  00    00    RR    RR       YY             SS
MM      MM  EEEEEEEEEE  MM      MM   000000     RR    RR       YY       SSSSSSSS
MM      MM  EEEEEEEEEE  MM      MM   000000     RR    RR       YY       SSSSSSSS
```

```
LL            IIIIII      SSSSSSSS
LL            IIIIII      SSSSSSSS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II        SSSSSS
LL              II        SSSSSS
LL              II              SS
LL              II              SS
LL              II              SS
LL              II              SS
LLLLLLLLLL    IIIIII    SSSSSSSS
LLLLLLLLLL    IIIIII    SSSSSSSS
```

```
0001   C
0002   C Version:       'V04-000'
0003   C
0004   C*************************************************************************
0005   C*                                                                      *
0006   C*    COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                           *
0007   C*    DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.            *
0008   C*    ALL RIGHTS RESERVED.                                             *
0009   C*                                                                      *
0010   C*    THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED *
0011   C*    ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE *
0012   C*    INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER *
0013   C*    COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY *
0014   C*    OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY *
0015   C*    TRANSFERRED.                                                      *
0016   C*                                                                      *
0017   C*    THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE *
0018   C*    AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT *
0019   C*    CORPORATION.                                                      *
0020   C*                                                                      *
0021   C*    DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS *
0022   C*    SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.          *
0023   C*                                                                      *
0024   C*                                                                      *
0025   C*************************************************************************
0026   C
0027   C
0028   C       Author Brian Porter                Creation Date 30-DEC-1980
0029   C
0030   C       Modified by:
0031   C
0032   C       V03-012 SAR0258        Sharon A. Reynolds     25-Apr-1984
0033   C               - Fixed a problem in the micro vax memory support.
0034   C               - Added an sye update that adds decoding for
0035   C               MA780 pcsr.
0036   C
0037   C       V03-011 SAR0246        Sharon A. Reynolds,     9-Apr-1984
0038   C               Added micro vax memory support.
0039   C
0040   C       V03-010 SAR0202        Sharon A. Reynolds,    27-Feb-1984
0041   C               Added the MEMORY_REGISTER_UV1 routine. Because memory
0042   C               support for the micro vax is not ready it will output a
0043   C               message. This was done so that the link of ERF would not
0044   C               fail.
0045   C
0046   C       V03-009 SAR0178        Sharon A. Reynolds,    30-Nov-1983
0047   C               Fixed the entry headers for 11/750 and 11/730 memories.
0048   C
0049   C       V03-008 SAR0169        Sharon A. Reynolds,    28-Oct-1983
0050   C               Added an SYE update that:
0051   C               - Modified MS780E reporting to weight array numbers
0052   C                 of the second cotnroller by 8.
0053   C               - Also added code to count multiple errors being
0054   C                 logged by one controller.
0055   C
0056   C       V03-007 SAR0083        Sharon A. Reynolds,    20-Jun-1983
0057   C               Changed the carriage control in the 'format' statements
```

```
0058    C           for use with ERF.
0059    C
0060    C    V03-006 SAR0058         Sharon A. Reynolds,     2-Jun-1983
0061    C           Made 'memory_display' a subroutine and added a routine
0062    C           that gets the necessary queue information so that 'memory_q'
0063    C           and 'memory_display' can be linked with separate images.
0064    C           Removed brief and cryptic code.
0065    C
0066    C    v03-005 BP0005          Brian Porter,           14-APR-1983
0067    C           11/730 syndrome bits are inverted so...
0068    C
0069    C    v03-004 BP0004          Brian Porter,           08-MAR-1983
0070    C           Corrected 11/730 memory size.
0071    C
0072    C    v03-003 BP0003          Brian Porter,           09-NOV-1982
0073    C           Corrected bank value for m8750 memories.
0074    C
0075    C    v03-002 BP0002          Brian Porter,           13-AUG-1982
0076    C           Corrected ms780e memory start address.
0077    C
0078    C    v03-001 BP0001          Brian Porter,           04-APR-1982
0079    C           Corrected output conversion error.
0080    C**
0081
0082          Subroutine MEMORY (lun,option)
0083
0084    C++
0085    C    Functional description:
0086    C
0087    C    This module extracts the varies arguments from the memory error
0088    C    entry and calls memory_q.  The format of a memory error entry is as
0089    C    follows.  The area occupied by the adaptor TR and registers is
0090    C    repeated for the 'number of controllers' times.  The register area
0091    C    size is dependent on the memory controller of the system being logged.
0092    C    The error pc and error psl are manufactured by VMS and have no meaning,
0093    C    therefore they are not printed in any of the reports.
0094    C
0095    C    +----------------------------------+
0096    C    :                                  :
0097    C    +--                              --+
0098    C    :           header space           :
0099    C    +--                              --+
0100    C    :                                  :
0101    C    +--                              --+
0102    C    :                                  :
0103    C    +----------------------------------+
0104    C    :       number of controllers      :
0105    C    +----------------------------------+
0106    C    :   adprtor tr# (or equivalent)    :
0107    C    +----------------------------------+
0108    C    :                                  :
0109    C    +--                              --+
0110    C    :                                  :
0111    C    +--      memory registers        --+
0112    C    :                                  :
0113    C    +--                              --+
0114    C    :                                  :
```

```
0115    C    +-----------------------------------+
0116    C    :             error pc              :
0117    C    +-----------------------------------+
0118    C    :             error psl             :
0119    C    +-----------------------------------+
0120    C--
0121
0122
0123
0124
0125         include 'src$:msghdr.for /nolist'
0184         include 'src$:syecom.for /nolist'
0312
0313
0314         byte            lun
0315
0316         character*1     option
0317
0318         integer*4       buffer(0:120)
0319         integer*4       controller_count
0320         Integer*4       I
0321         Integer*4       lib$extzv
0322         integer*4       compress4
0323         integer*4       error_array
0324         integer*4       error_bank
0325         integer*4       error_bit
0326         integer*4       page_count
0327         integer*4       array_code
0328         integer*4       decode_ecc
0329
0330         equivalence     (emb(16),buffer)
0331         equivalence     (buffer(0),controller_count)
0332
0333         logical*1       l0011
0334         logical*1       l0016
0335
0336
0337         if (
0338       1 lib$extzv(24,8,emb$l_hd_sid) .eq. 255
0339       1 .or.
0340       1 lib$extzv(24,8,emb$l_hd_sid) .eq. 1
0341       1 ) then
0342
0343         j = 0
0344
0345         do 20,i = 1,controller_count
0346
0347         if (option .ne. 'R'
0348       1 .and.
0349       1 j .eq. 0) then
0350
0351         call header (lun)
0352
0353         if (emb$w_hd_entry .eq. 5) then
0354
0355         call logger (lun,'SBI ALERT')
0356
```

```
0357                else if (emb$w_hd_entry .eq. 8) then
0358
0359                call logger (lun,'FATAL MEMORY ERROR')
0360                else
0361
0362                call logger (lun,'MEMORY ERROR')
0363                endif
0364                endif
0365
0366
0367         c
0368         c      MS780C
0369         c
0370                if (lib$extzv(5,3,buffer(2 + j)) .eq. 0) then
0371
0372                if (emb$w_hd_entry .ne. '05'x) then
0373
0374                if (lib$extzv(28,1,buffer(4 + j)) .eq. 1) then
0375
0376                if (lib$extzv(4,1,buffer(2 + j)) .eq. 0) then
0377
0378                call memory_g (emb$l_hd_sid,buffer(1 + j),
0379              1 lib$extzv(24,4,buffer(4 + j)),lib$extzv(21,1,buffer(4+j)),
0380              1 decode_ecc(lib$extzv(0,8,buffer(4 + j)),buffer(2 + j)))
0381
0382                else if (lib$extzv(4,1,buffer(2 + j)) .eq. 1) then
0383
0384                call memory_g (emb$l_hd_sid,buffer(1 + j),
0385              1 lib$extzv(24,4,buffer(4 + j)),lib$extzv(23,1,buffer(4+j)),
0386              1 decode_ecc(lib$extzv(0,8,buffer(4 + j)),buffer(2 + j)))
0387                endif
0388                endif
0389                endif
0390
0391         c
0392         c Full report output the TR# and call the MS780C routine to
0393         c decode/output the remainder of the entry.
0394         c
0395                if (option .eq. 'S') then
0396
0397                call linchk (lun,2)
0398
0399                write(lun,10) buffer(1 + j)
0400          10    format(/' ','CONTROLLER AT TR #',i<compress4 (buffer(1 + j))>,'.')
0401
0402                call ms780c (lun,buffer(2 + j))
0403                endif
0404
0405
0406                j = j + 4
0407
0408
0409         c
0410         c      MS780E
0411         c
0412                else if (lib$extzv(5,3,buffer(2 + j)) .eq. 3) then
0413
```

```
0414            if (emb$w_hd_entry .ne. '05'x) then
0415
0416            if (lib$extzv(28,1,buffer(4 + j)) .eq. 1) then
0417
0418            call memory_q (emb$l_hd_sid,buffer(1 + j),
0419        1   lib$extzv(24,3,buffer(4 + j)),lib$extzv(22,2,buffer(4+j)),
0420        1   decode_ecc(lib$extzv(0,7,buffer(4 + j)),buffer(2 + j)))
0421            Endif
0422
0423            If (lib$extzv(28,1,buffer(5 + j)) .eq. 1) then
0424
0425            call memory_q (emb$l_hd_sid,buffer(1 + j),
0426        1   (lib$extzv(24,3,buffer(5 + j))+8),lib$extzv(22,2,buffer(5+j)),
0427        1   decode_ecc(lib$extzv(0,7,buffer(5 + j)),buffer(2 + j)))
0428            Endif
0429
0430            If (
0431        1   lib$extzv(18,2,buffer(2 + j)) .ne. 0
0432        1   .or.
0433        1   lib$extzv(7,1,buffer(3 + j)) .eq. 1
0434        1   .or.
0435        1   lib$extzv(7,1,buffer(4 + j)) .eq. 1
0436        1   .or.
0437        1   lib$extzv(7,1,buffer(5 + j)) .eq. 1
0438        1   ) then
0439
0440            call memory_q (emb$l_hd_sid,buffer(1 + j),-1,-1,-1)
0441            endif
0442            endif
0443
0444    C
0445    C Full report output the TR# and call the MS780E routine to
0446    C decode/output the remainder of the entry.
0447    C
0448            if (option .eq. 'S') then
0449
0450            call linchk (lun,2)
0451            write(lun,10) buffer(1 + j)
0452
0453            call ms780e (lun,buffer(2 + j))
0454            endif
0455
0456
0457            j = j + 5
0458
0459    C
0460    C       MA780
0461    C
0462            else if (lib$extzv(5,3,buffer(2 + j)) .eq. 2) then
0463
0464            if (emb$w_hd_entry .ne. '05'x) then
0465
0466            if (lib$extzv (28,1,buffer(6 + j)) .eq. 1) then
0467
0468            call memory_q (emb$l_hd_sid,buffer(1 + j),
0469        1   lib$extzv(24,4,buffer(6 + j)),lib$extzv(23,1,buffer(6+j)),
0470        1   decode_ecc(lib$extzv(0,8,buffer(6 + j)),buffer(2 + j)))
```

```
0471              Endif
0472
0473              If (
0474          1    lib$extzv(26,6,buffer(3 + j)) .ne. 0
0475          1    .or.
0476          1    lib$extzv(14,2,buffer(4 + j)) .ne. 0
0477          1    .or.
0478          1    lib$extzv(28,1,buffer(4 + j)) .eq. 1
0479          1    .or.
0480          1    lib$extzv(30,2,buffer(4 + j)) .ne. 0
0481          1    .or.
0482          1    lib$extzv(31,1,buffer(6 + j)) .eq. 1
0483          1    ) then
0484
0485              call memory_q (emb$l_hd_sid,buffer(1 + j),-1,-1,-1)
0486              endif
0487              endif
0488
0489    C
0490    C Full report output the TR# and call the MA780 routine to
0491    C decode/output the remainder of the entry.
0492    C
0493              if (option .eq. 'S') then
0494
0495              call linchk (lun,2)
0496              write(lun,10) buffer(1 + j)
0497
0498              call ma780 (lun,buffer(2 + j))
0499              endif
0500
0501              j = j + 9
0502              endif
0503
0504    20        continue
0505
0506
0507    C
0508    C         11/750
0509    C
0510              else if (lib$extzv(24,8,emb$l_hd_sid) .eq. 2) then
0511
0512              If (option .eq. 'S'
0513          1    .OR.
0514          2    option .eq. 'R') then
0515
0516              error_array = lib$extzv(9,15,buffer(2))
0517
0518              l0011 = .false.
0519
0520              l0016 = .false.
0521
0522              if (jiand(buffer(4),'01000000'x) .ne. 0) then
0523
0524              l0016 = .true.
0525              else
0526
0527              l0011 = .true.
```

0287
0288
0289
0290
0291
0292
0293
0294
0295
0296
0297
0298
0299
0300
0301
0302
0303
0304
0305
0306
0307
0308
0309
0310
0311
0312
0313
0314
0315
0316
0317
0318
0319
0320
0321
0322
0323
0324
0325
0326
0327
0328
0329
0330
0331
0332
0333
0334
0335
0336
0337
0338
0339
0340
0341
0342
0343

```
0528              endif
0529
0530              do 30,i = 0,15,2
0531
0532              array_code = lib$extzv(i,2,buffer(4))
0533
0534              if (l0016) then
0535
0536              if (array_code .eq. 3) then
0537
0538              error_array = error_array - 512
0539
0540              if (lib$extzv(17,1,buffer(2)) .eq. 0) then
0541
0542              error_bank = 0
0543              else
0544
0545              error_bank = 1
0546              endif
0547
0548              else if (array_code .eq. 2) then
0549
0550              error_array = error_array - 2048
0551
0552              error_bank = lib$extzv(19,2,buffer(2))
0553              endif
0554
0555              else if (l0011) then
0556
0557              if (array_code .eq. 1) then
0558
0559              error_array = error_array - 256
0560
0561              error_bank = 0
0562
0563              else if (array_code .eq. 3) then
0564
0565              error_array = error_array - 512
0566
0567              if (lib$extzv(17,1,buffer(2)) .eq. 0) then
0568
0569              error_bank = 0
0570              else
0571
0572              error_bank = 1
0573              endif
0574              endif
0575              endif
0576
0577              if (error_array .le. 0) then
0578
0579              error_array = i/2
0580
0581              goto 40
0582              endif
0583
0584      30      continue
```

0344
0345
0346
0347
0348
0349
0350
0351
0352
0353
0354
0355
0356
0357
0358
0359
0360
0361
0362
0363
0364
0365
0366
0367
0368
0369
0370
0371
0372
0373
0374
0375
0376
0377
0378
0379
0380
0381
0382
0383
0384
0385
0386
0387
0388
0389
0390
0391
0392
0393
0394
0395
0396
0397
0398
0399
0400

```
0585
0586    40        if (lib$extzv(29,1,buffer(2)) .eq. 1) then
0587
0588              call memory_q (emb$l_hd_sid,lib$extzv(0,3,buffer(1)),
0589         1 error_array,error_bank,decode_ecc(lib$extzv(0,7,buffer(2))))
0590
0591              else if (lib$extzv(31,1,buffer(2)) .eq. 1) then
0592
0593              call memory_q (emb$l_hd_sid,lib$extzv(0,3,buffer(1)),
0594         1 error_array,error_bank,-1)
0595              endif
0596              endif
0597
0598              If (option .NE. 'R') then
0599
0600              call header (lun)
0601
0602              If (emb$w_hd_entry .eq. 8) then
0603
0604              call logger (lun,'FATAL MEMORY ERROR')
0605              else
0606
0607              call logger (lun,'MEMORY ERROR')
0608              endif
0609              endif
0610
0611              if (option .eq. 'S') then
0612
0613              call linchk (lun,2)
0614              write(lun,50) lib$extzv(0,3,buffer(1))
0615    50        format(/' ','CONTROLLER AT SLOT INDEX #',
0616         1 i<compress4 (lib$extzv(0,3,buffer(1)))>,'.')
0617
0618              call ms750 (lun,buffer(2))
0619              endif
0620
0621    C
0622    C        11/730
0623    C
0624              else if (lib$extzv(24,8,emb$l_hd_sid) .eq. 3) then
0625
0626              If (option .eq. 'S'
0627         1 .or.
0628         2 option .eq. 'R') then
0629
0630    c        11/730 syndrome bits are inverted so...
0631    c
0632
0633              error_bit = decode_ecc (lib$extzv(0,7,not(lib$extzv(0,7,buffer(0)))))
0634
0635              error_array = lib$extzv(9,15,buffer(0))
0636
0637              if (lib$extzv(24,1,buffer(2)) .eq. 1) then
0638
0639              page_count = 1024
0640              else
0641
```

```
0642              page_count = 256
0643              endif
0644
0645              do 60,i = 0,15
0646
0647              if (lib$extzv(i,1,buffer(2)) .eq. 1) then
0648
0649              error_array = error_array - page_count
0650
0651              if (error_array .le. 0) then
0652
0653              error_array = i/2
0654
0655              goto 65
0656              endif
0657              endif
0658
0659   60         continue
0660
0661   65         if (page_count .eq. 1024) then
0662
0663              error_bank = lib$extzv (19,2,buffer(0))
0664              else
0665
0666              error_bank = lib$extzv (17,1,buffer(0))
0667              endif
0668
0669              if (error_bit .eq. -1) then
0670
0671              call memory_q (emb$l_hd_sid,0,
0672             1 error_array,error_bank,-1)
0673              else
0674
0675              call memory_q (emb$l_hd_sid,0,
0676             1 error_array,error_bank,error_bit)
0677              endif
0678              endif
0679
0680              If (option .ne. 'R') then
0681
0682              call header (lun)
0683
0684              If (emb$w_hd_entry .eq. 8) then
0685
0686              call logger (lun,'FATAL MEMORY ERROR')
0687              else
0688
0689              call logger (lun,'MEMORY ERROR')
0690              endif
0691
0692              call ms730 (lun,buffer)
0693              endif
0694
0695
0696   c          UVAX1
0697   c
0698
```

```
0699              else if (lib$extzv(24,8,emb$l_hd_sid) .eq. 7) then
0700
0701              do 80,i = 1,16
0702
0703              if (lib$extzv(15,1,buffer(i)) .eq. 1) then
0704
0705              call memory_q (emb$l_hd_sid,0,i,-1,-1)
0706              endif
0707
0708    80        continue
0709
0710              if (option .ne. 'C'
0711         1 .and.
0712         1 option .ne. 'R') then
0713
0714              call header (lun)
0715
0716              call logger (lun,'FATAL MEMORY ERROR')
0717              endif
0718
0719              if (option .eq. 'S') then
0720
0721              call memory_register_uv1 (lun,buffer(0))
0722              endif
0723
0724    c
0725    c         The IF-THEN-ELSE should be expanded at this point to add
0726    c         additional CPU memory support.
0727    c
0728              endif
0729
0730              return
0731              end
```

PROGRAM SECTIONS

	Name	Bytes	Attributes
0	$CODE	2633	PIC CON REL LCL SHR EXE RD NOWRT LONG
1	$PDATA	218	PIC CON REL LCL SHR NOEXE RD NOWRT LONG
2	$LOCAL	928	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
3	EMB	512	PIC OVR REL GBL SHR NOEXE RD WRT LONG
4	SYECOM	44	PIC OVR REL GBL SHR NOEXE RD WRT LONG

Total Space Allocated   4335

ENTRY POINTS

Address	Type	Name
0-00000000		MEMORY

VARIABLES

Address	Type	Name	Address	Type	Name
2-00000018	I*4	ARRAY_CODE	3-00000010	I*4	CONTROLLER_COUNT
4-00000012	L*1	CP_11750	4-00000011	L*1	CP_11780
4-00000013	L*1	CP_1172Z	4-00000014	L*4	CRYPTK_FLAG
4-0000000D	I*4	DEV_CHAR	3-00000000	I*4	EMB$L_RD_SID
3-00000004	I*2	EMB$W_HD_ENTRY	3-0000000E	I*2	EMB$W_HD_ERRSEQ
4-0000001E	L*1	END_VALUE	4-0000001D	L*1	EOF_FLAG
2-00000008	I*4	ERROR_ARRAY	2-0000000C	I*4	ERROR_BANK
2-00000010	I*4	ERROR_BIT	4-00000004	L*4	FORMS
2-00000004	I*4	I	2-0000001C	I*4	J
2-00000000	L*1	L0011	2-00000001	L*1	L0016
4-0000000C	L*1	LINES	4-00000027	L*4	LSTLUN
AP-0000004@	L*1	LUN	4-0000001F	I*4	MAILBOX_CHANNEL
AP-0000008@	CHAR	OPTION	4-0000002B	CHAR	OPTIONS
2-00000014	I*4	PAGE_COUNT	4-00000008	L*4	PRINTER
4-00000000	I*4	RECCNT	4-00000023	I*4	RECORD_SIZE
4-00000019	L*1	VALID_CLASS	4-0000001A	L*1	VALID_CPU
4-0000001B	L*1	VALID_ENTRY	4-0000001C	L*1	VALID_TYPE
4-00000018	L*1	VOLUME_OUTPUT			

ARRAYS

Address	Type	Name	Bytes	Dimensions
3-00000010	I*4	BUFFER	484	(0:120)
3-00000000	L*1	EMB	512	(0:511)
3-00000006	I*4	EMB$Q_HD_TIME	8	(2)

LABELS

Address	Label	Address	Label	Address	Label	Address	Label	Address	Label	Address	Label
1-0000008E	10'	**	20	**	30	0-0000075C	40	1-000000B0	50'	**	60
0-000008FC	65'	**	80								

FUNCTIONS AND SUBROUTINES REFERENCED

Type	Name	Type	Name	Type	Name
I*4	COMPRESS4	I*4	DECODE_ECC		HEADER
I*4	LIB$EXTZV		LINCHK		LOGGER
	MA780		MEMORY_Q		MEMORY_REGISTER_UV1
	MS730		MS750		MS780C
	MS780E				

0002
0003
0004
0005
0006
0007
0008
0009
0010
0011
0012
0013
0014
0015
0016
0017
0018
0019
0020
0021
0022
0023
0024
0025
0026
0027
0028
0029
0030
0031
0032
0033
0034
0035
0036
0037
0038
0039
0040
0041
0042
0043
0044
0045
0046
0047
0048
0049
0050
0051
0052
0053
0054
0055
0056
0057
0058

0001

MS78

0059
0060
0061
0062
0063
0064
0065
0066
0067
0068
0069
0070
0071
0072
0073
0074
0075
0076
0077
0078
0079
0080
0081
0082
0083
0084
0085
0086
0087
0088
0089
0090
0091
0092
0093
0094
0095
0096
0097
0098
0099
0100
0101
0102
0103
0104
0105
0106
0107
0108
0109
0110
0111
0112
0113
0114
0115

```
0002                    Subroutine MA780 (lun,memory_registers)
0003
0004      c++
0005      c          This routine displays the ma780 memory registers.
0006      c          The format of the multi-port memory sub packet is as follows.
0007      c
0008      c          +----------------------------------+
0009      c          :     port configuration register  :
0010      c          +----------------------------------+
0011      c          :  port interface control register  :
0012      c          +----------------------------------+
0013      c          :  port controller status register  :
0014      c          +----------------------------------+
0015      c          :  port invalidation control reg    :
0016      c          +----------------------------------+
0017      c          :        array error register       :
0018      c          +----------------------------------+
0019      c          :  configuration status register 0  :
0020      c          +----------------------------------+
0021      c          :  configuration status register 1  :
0022      c          +----------------------------------+
0023      c          :     maintenance control register   :
0024      c          +----------------------------------+
0025      c--
0026
0027              Implicit        none
0028              byte            lun
0029
0030              integer*4       memory_registers
0031              integer*4       buffer(8)
0032              integer*4       port_configuration_register
0033              integer*4       port_interface_control_register
0034              integer*4       port_controller_status_register
0035              integer*4       port_invalidation_control_reg
0036              integer*4       array_error_register
0037              integer*4       configuration_status_register0
0038              integer*4       configuration_status_register1
0039              integer*4       maintenance_control_register
0040
0041              equivalence     (buffer(1),port_configuration_register)
0042              equivalence     (buffer(2),port_interface_control_register)
0043              equivalence     (buffer(3),port_controller_status_register)
0044              equivalence     (buffer(4),port_invalidation_control_reg)
0045              equivalence     (buffer(5),array_error_register)
0046              equivalence     (buffer(6),configuration_status_register0)
0047              equivalence     (buffer(7),configuration_status_register1)
0048              equivalence     (buffer(8),maintenance_control_register)
0049
0050              integer*4       ram_count
0051              integer*4       array_count
0052              integer*4       starting_address
0053              integer*4       error_syndrome
0054              integer*4       error_bit
0055              integer*4       error_array
0056              Integer*4       error_bank
0057              integer*4       array_init_status_bits
0058              integer*4       port_type_5its
```

```
0116
0117
0118
0119
0120
0121
0122
0123
0124
0125
0126
0127
0128
0129
0130
0131
0132
0133
0134
0135
0136
0137
0138
0139
0140
0141
0142
0143
0144
0145
0146
0147
0148
0149
0150
0151
0152
0153
0154
0155
0156
0157
0158
0159
0160
0161
0162
0163
0164
0165
0166
0167
0168
0169
0170
0171
0172
```

```
0059           equivalence      (ram_count,array_count,starting_address,error_syndrome,
0060          1 error_bit,error_array,array_init_status_bits,port_type_bits)
0061
0062
0063           integer*4        compress4
0064           integer*4        compressc
0065           Integer*4        I
0066           Integer*4        J
0067           integer*4        lib$extzv
0068           Integer*4        lib$locc
0069           integer*4        decode_ecc
0070           integer*4        adapter_tr
0071
0072           logical*1        diagnostic_mode
0073
0074           character*32     v1register2(0:1)
0075           data             v1register2(0)    /'MASTER INTERRUPT ENABLE*'/
0076           data             v1register2(1)    /'PORT INTERFACE INTERRUPT ENABLE*'/
0077
0078           character*30     v3register2(23:31)
0079           data             v3register2(23) /'MARK INTERLOCK IN PROGRESS*'/
0080           data             v3register2(24) /'MARK TIMEOUT*'/
0081           data             v3register2(25) /'MARK REQUESTER*'/
0082           data             v3register2(26) /'OUTPUT BUFFER OVERFLOW*'/
0083           data             v3register2(27) /'INVALIDATION ACK NOT RECEIVED*'/
0084           data             v3register2(28) /'OUTPUT BUFFER PARITY ERROR*'/
0085           data             v3register2(29) /'INVALIDATE LOST ON BDI*'/
0086           data             v3register2(30) /'BDI PARITY FAULT ON OUTPUT*'/
0087           data             v3register2(31) /'BDI PARITY FAULT ON INPUT*'/
0088
0089           character*23     v1register3(1)
0090           data             v1register3(1)    /'ERROR INTERRUPT ENABLE*'/
0091
0092           character*25     v2register3(6:8)
0093           data             v2register3(6)    /'SELF INVALIDATE ENABLE*'/
0094           data             v2register3(7)    /'INVALIDATION DISABLE*'/
0095           data             v2register3(8)    /'INHIBIT ADMI ARBITRATION*'/
0096
0097           character*25     v3register3(10:15)
0098           data             v3register3(10) /'INTERLOCK GRANT ACCEPTED*'/
0099           data             v3register3(11) /'INTERLOCK FLIP-FLOP*'/
0100           data             v3register3(12) /'ARRAY INIT IN PROGRESS*'/
0101   c
0102   c       Cell 13 of this array unused
0103   c
0104           data             v3register3(14) /'INVALIDATE DATA LOST*'/
0105           data             v3register3(15) /'INTERLOCK TIMEOUT*'/
0106
0107           character*30     v4register3(21)
0108           data             v4register3(21) /'NO C/A ON ADMI WHEN REQUESTED*'/
0109
0110           character*23     v5register3(22:25,0:1)
0111           data             v5register3(22,0)/'32-BIT OPERATION*'/
0112           data             v5register3(22,1)/'64-BIT OPERATION*'/
0113           data             v5register3(23,0)/'I/O SELECT*'/
0114           data             v5register3(23,1)/'ARRAY SELECT*'/
0115           data             v5register3(24,0)/'REQUESTER HAS NO CACHE*'/
```

MA780

K 3
16-Sep-1984 00:07:33   VAX-11 FORTRAN V3.4-56   Page 16
5-Sep-1984 14:01:18   DISK$VMSMASTER:[ERF.SRC]MEMORYS.FCR;1

MS78

LABE

1-
1-

FUNC

Ty

I

```
0116          data          v5register3(24,1)/'REQUESTER HAS CACHE*'/
0117          data          v5register3(25,0)/'ADMI READ*'/
0118          data          v5register3(25,1)/'ADMI WRITE*'/
0119
0120          character*32   v6register3(28:31)
0121          data          v6register3(28) /'MULTIPLE ADMI GRANT*'/
0122          data          v6register3(29) /'PORT TRANSMITTING DURING FAULT*'/
0123          data          v6register3(30) /'ADMI MULTIPLE TRANSMITTER FAULT*'/
0124          data          v6register3(31) /'ADMI COMMAND ABORT*'/
0125
0126          character*14   v1register4(31:31)
0127          data          v1register4(31) /'CACHED FORCED*'/
0128
0129          character*18   v1register5(28:31)
0130          data          v1register5(28) /'ERROR LOG REQUEST*'/
0131          data          v1register5(29) /'HIGH ERROR RATE*'/
0132          data          v1register5(30) /'CRD TAG*'/
0133          data          v1register5(31) /'MAP PARITY ERROR*'/
0134
0135          character*27   v1register6(0:1)
0136          data          v1register6(0)  /'NONCONTIGUOUS ARRAY ERROR*'/
0137          data          v1register6(1)  /'4K CHIP ARRAY ERROR*'/
0138
0139          character*28   v1register7(10:11)
0140          data          v1register7(10) /'MULTIPLE INTERLOCK ACCEPTED*'/
0141          data          v1register7(11) /'INVALIDATION MAP PRESENT*'/
0142
0143          character*27   array_init_status
0144          character*22   port_type
0145
0146
0147          call movc3 (%val(32),memory_registers,buffer)
0148
0149          diagnostic_mode = .false.
0150
0151          if (lib$extzv(4,3,port_interface_control_register) .ne. 0
0152        1 .or.
0153        1 lib$extzv(4,2,port_controller_status_register) .ne. 0
0154        1 .or.
0155        1 lib$extzv(13,1,port_controller_status_register) .ne. 0
0156        1 .or.
0157        1 lib$extzv(8,2,configuration_status_register1) .ne. 0
0158        1 .or.
0159        1 lib$extzv(8,6,maintenance_control_register) .ne. 0) then
0160
0161          diagnostic_mode = .true.
0162
0163          call linchk (lun,1)
0164
0165          write(lun,5) port_configuration_register
0166        5 format(/' ',t8,'PRTCFNG',t24,z8.8)
0167          endif
0168
0169          call ma780_rega (lun,port_configuration_register)
0170
0171          call linchk (lun,1)
0172
```

```
0173            write(lun,10) port_interface_control_register
0174      10    format(' ',t8,'PRTCR',t24,z8.8)
0175
0176            if (.not. diagnostic_mode) then
0177
0178            call output (lun,port_interface_control_register,v1register2,0,0,1,'0')
0179
0180            call linchk (lun,1)
0181
0182            ram_count = lib$extzv(16,4,port_interface_control_register)
0183
0184            write(lun,15) ram_count
0185      15    format(' ',t40,'RAM COUNT ',i<compress4 (ram_count)>,'.')
0186
0187            call output (lun,port_interface_control_register,v3register2,23,23,31,
0188           1 '0 )
0189            endif
0190
0191            if (lib$extzv(4,3,port_interface_control_register) .ne. 0) then
0192
0193            call linchk (lun,1)
0194
0195            write(lun,17)
0196      17    format(' ',t40,'DIAGNOSTIC MODE')
0197            endif
0198
0199            call linchk (lun,1)
0200
0201            write(lun,20) port_controller_status_register
0202      20    format(' ',t8,'PCSR',t24,z8.8)
0203
0204            if (lib$extzv(13,1,port_controller_status_register) .ne. 0
0205           1 .or.
0206           1 lib$extzv (4,2,port_controller_status_register) .ne. 0) then
0207
0208            call linchk (lun,1)
0209
0210            write(lun,17)
0211            endif
0212
0213            if (.not. diagnostic_mode) then
0214
0215            call output (lun,port_controller_status_register,v1register3,1,1,1,'0')
0216
0217            call output (lun,port_controller_status_register,v2register3,6,6,8,'0')
0218
0219            call output (lun,port_controller_status_register,v3register3,10,10,12,
0220           1 '0')
0221
0222            call output (lun,port_controller_status_register,v3register3,10,14,15,
0223           1 '0')
0224
0225            if (jiand(port_controller_status_register,'d0000000'x) .ne. 0) then
0226
0227            call output (lun,port_controller_status_register,v4register3,21,21,21,
0228           1 '0')
0229
```

0001

```
0230              call output (lun,port_controller_status_register,v5register3,22,22,25,    0002
0231         1 '2')                                                                          0003
0232                                                                                         0004
0233              I = LIB$EXTZV (26,2,port_controller_status_register)                       0005
0234                                                                                         0006
0235              Call LINCHK (lun,1)                                                         0007
0236              Write (lun,25) I                                                           0008
0237         25   Format (' ',T40,'"ADMI" PORT #',I<COMPRESS4 (I)>,'.')                      0009
0238                                                                                         0010
0239              call output (lun,port_controller_status_register,v6register3,28,28,31,     0011
0240         1 '0')                                                                          0012
0241              endif                                                                       0013
0242              endif                                                                       0014
0243                                                                                         0015
0244              call linchk (lun,1)                                                         0016
0245                                                                                         0017
0246              write(lun,30) port_invalidation_control_reg                                0018
0247         30   format(' ',t8,'IVDTCR',t24,z8.8)                                           0019
0248                                                                                         0020
0249              if (.not. diagnostic_mode) then                                            0021
0250                                                                                         0022
0251              do 50,i = 0,15                                                             0023
0252                                                                                         0024
0253              if (lib$extzv(i,1,port_invalidation_control_reg) .eq. 1) then              0025
0254                                                                                         0026
0255              call linchk (lun,1)                                                         0027
0256                                                                                         0028
0257              write(lun,40) i                                                            0029
0258         40   format(' ',t40,'INVALIDATE CACHE DEVICE ID = ',i<compress4 (i)>,'.')       0030
0259              endif                                                                       0031
0260                                                                                         0032
0261         50   continue                                                                    0033
0262                                                                                         0034
0263              call linchk (lun,1)                                                         0035
0264                                                                                         0036
0265              array_count = lib$extzv(16,3,port_invalidation_control_reg)                0037
0266                                                                                         0038
0267              if (array_count .eq. 0                                                     0039
0268         1 .and.                                                                          0040
0269         1 lib$extzv(0,2,configuration_status_register0) .ne. 0) then                    0041
0270                                                                                         0042
0271              write(lun,60) 'INVALID ARRAY CONFIGURATION'                                0043
0272         60   format(' ',t40,a)                                                          0044
0273              else                                                                        0045
0274                                                                                         0046
0275              write(lun,70) array_count + 1                                              0047
0276         70   format(' ',t40,i<compress4 (array_count + 1)>,                             0048
0277         1 ', ARRAY BOARD(S) PRESENT')                                                   0049
0278              endif                                                                       0050
0279                                                                                         0051
0280              starting_address = lib$extzv(20,11,port_invalidation_control_reg)*256      0052
0281                                                                                         0053
0282              call linchk (lun,1)                                                         0054
0283                                                                                         0055
0284              write(lun,80) starting_address                                             0056
0285         80   format(' ',t40,'MEMORY BASE ADDRESS = ',                                   0057
0286         1 i<compress4 (starting_address)>,'.K')                                         0058
```

```
0287              call output (lun,port_invalidation_control_reg,v1register4,31,31,31,
0288             1 '0')
0289              endif
0290
0291              call linchk (lun,1)
0292
0293              write(lun,90) array_error_register
0294        90    format(' ',t8,'AER',t24,z8.8)
0295
0296              if (.not. diagnostic_mode) then
0297
0298              if (lib$extzv (28,1,array_error_register)  eq. 1) then
0299
0300              error_syndrome = lib$extzv(0,8,array_error_register)
0301
0302              call linchk (lun,1)
0303
0304              write(lun,100) error_syndrome
0305       100    format(' ',t40,'ERROR SYNDROME = ',z2.2)
0306
0307              error_bit = decode_ecc (error_syndrome,port_configuration_register)
0308
0309              call linchk (lun,1)
0310
0311              if (error_bit .eq. -1) then
0312
0313              write(lun,110) 'RDS ERROR'
0314       110    format(' ',t40,a)
0315              else
0316
0317              write(lun,120) 'CRD ERROR, CORRECTED BIT #',error_bit,'.'
0318       120    format(' ',t40,a,i<compress4 (error_bit)>,a)
0319              endif
0320
0321              error_array = lib$extzv (24,4,array_error_register)
0322
0323              call linchk (lun,1)
0324
0325              write(lun,140) error_array
0326       140    format(' ',t40,'ARRAY #',i<compress4 (error_array)>,'. IN ERROR')
0327              endif
0328
0329              Error_bank = LIB$EXTZV(23,1,array_error_register)
0330
0331              Call LINCHK (lun,1)
0332              Write (lun,145) error_bank
0333       145    Format(' ',T40,
0334             1 'ARRAY BANK #',I<COMPRESS4 (error_bank)>,'. IN ERROR')
0335
0336              call output (lun,array_error_register,v1register5,28,28,31,'0')
0337              endif
0338
0339              call linchk (lun,1)
0340
0341              write(lun,170) configuration_status_register0
0342       170    format(' ',t8,'CSR0',t24,z8.8)
0343
```

0287
0288
0289
0290
0291
0292
0293
0294
0295
0296
0297
0298
0299
0300
0301
0302
0303
0304
0305
0306
0307
0308
0309
0310
0311
0312
0313
0314
0315
0316
0317
0318
0319
0320
0321
0322
0323
0324
0325
0326
0327
0328
0329
0330
0331
0332
0333
0334
0335
0336
0337
0338
0339
0340
0341
0342
0343

0059
0060
0061
0062
0063
0064
0065
0066
0067
0068
0069
0070
0071
0072
0073
0074
0075
0076
0077
0078
0079
0080
0081
0082
0083
0084
0085
0086
0087
0088
0089
0090
0091
0092
0093
0094
0095
0096
0097
0098
0099
0100
0101
0102
0103
0104
0105
0106
0107
0108
0109
0110
0111
0112
0113
0114
0115

```
0344
0345              if (.not. diagnostic_mode) then
0346
0347              call output (lun,configuration_status_register0,v1register6,0,0,1,'0')
0348
0349              array_init_status_bits = lib$extzv(2,2,configuration_status_register0)
0350
0351              if (array_init_status_bits .eq. 0) then
0352
0353              array_init_status = 'INITIALIZATION IN PROGRESS*'
0354
0355              else if (array_init_status_bits .eq. 2) then
0356
0357              array_init_status = 'MEMORY CONTAINS VALID DATA*'
0358
0359              else if (array_init_status_bits .eq. 3) then
0360
0361              array_init_status = 'INITIALIZATION COMPLETE*'
0362              endif
0363
0364              call linchk (lun,1)
0365
0366              write(lun,180) array_init_status(:(lib$locc('*',array_init_status)-1))
0367     180      format(' ',t40,a)
0368
0369              do 200,i = 4,7
0370
0371              if (lib$extzv(i,1,configuration_status_register0) .eq. 1) then
0372
0373              call linchk (lun,1)
0374
0375              write(lun,190) 'PORT #',i - 4,'. POWERED DOWN'
0376     190      format(' ',t40,a,i<compress4 (i - 4)>,a)
0377              endif
0378
0379     200      continue
0380
0381              do 220,i = 8,11
0382
0383              if (lib$extzv(i,1,configuration_status_register0) .eq. 1) then
0384
0385              call linchk (lun,1)
0386
0387              write(lun,210) 'ERROR INTERRUPT FROM PORT #',i - 8,'.'
0388     210      format(' ',t40,a,i<compress4 (i - 8)>,a)
0389              endif
0390
0391     220      continue
0392
0393              do 240,i = 12,15
0394
0395              if (lib$extzv(i,1,configuration_status_register0) .eq. 1) then
0396
0397              call linchk (lun,1)
0398
0399              write(lun,230) 'PORT #',i - 12,'. OFFLINE'
0400     230      format(' ',t40,a,i<compress4 (i - 12)>,a)
```

0116
0117
0118
0119
0120
0121
0122
0123
0124
0125
0126
0127
0128
0129
0130
0131
0132
0133
0134
0135
0136
0137
0138
0139
0140
0141
0142
0143
0144
0145
0146
0147
0148
0149
0150
0151
0152
0153
0154
0155
0156
0157
0158
0159
0160
0161
0162
0163
0164
0165
0166
0167
0168
0169
0170
0171
0172

```
0401            endif
0402
0403    240     continue
0404            endif
0405
0406            call linchk (lun,1)
0407
0408            write(lun,250) configuration_status_register1
0409    250     format(' ',t8,'CSR1',t24,z8.8)
0410
0411            if (.not. diagnostic_mode) then
0412
0413            do 270,i = 0,6,2
0414
0415            port_type_bits = lib$extzv(i,2,configuration_status_register1)
0416
0417            if (port_type_bits .eq. 0) then
0418
0419            port_type = '. NOT PRESENT*'
0420
0421            else if (port_type_bits .eq. 2) then
0422
0423            port_type = '. CONNECTED TO AN SBI*'
0424            endif
0425
0426            if (port_type_bits .ne. 1
0427           1 .or.
0428           1 port_type_bits .ne. 3) then
0429
0430            call linchk (lun,1)
0431
0432            write(lun,260) i/2,port_type
0433    260     format(' ',t40,'PORT #',i<compress4 (i/2)>,a<compressc (port_type)>)
0434            endif
0435
0436    270     continue
0437
0438            call output (lun,configuration_status_register1,v1register7,10,10,11,
0439           1 '0')
0440
0441            do 290,i = 12,15
0442
0443            if (lib$extzv(i,1,configuration_status_register1) .eq. 1) then
0444
0445            call linchk (lun,1)
0446
0447            write(lun,280) 'PORT #',i - 12,'. INVALIDATION ACK RECEIVED'
0448    280     format(' ',t40,a,i<compress4 (i - 12)>,a)
0449            endif
0450
0451    290     continue
0452            endif
0453
0454            if (lib$extzv(8,2,configuration_status_register1) .ne. 0) then
0455
0456            call linchk (lun,1)
0457
```

```
0458              write(lun,17)
0459              endif
0460
0461              call linchk (lun,1)
0462
0463              write(lun,295) maintenance_control_register
0464     295      format(' ',t8,'MAT',t24,z8.8)
0465
0466              if (lib$extzv(8,6,maintenance_control_register) .ne. 0) then
0467
0468              call linchk (lun,1)
0469
0470              write(lun,17)
0471              endif
0472
0473              return
0474              end
```

PROGRAM SECTIONS

	Name	Bytes	Attributes								
0	$CODE	2631	PIC CON REL LCL		SHR	EXE		RD	NOWRT	LONG
1	$PDATA	919	PIC CON REL LCL		SHR	NOEXE		RD	NOWRT	LONG
2	$LOCAL	3188	PIC CON REL LCL	NOSHR	NOEXE			RD	WRT	LONG

    Total Space Allocated   6738

ENTRY POINTS

Address	Type	Name
0-00000000		MA780

VARIABLES

Address	Type	Name	Address	Type	Name
2-0000071C	I*4	ADAPTER_TP	2-00000000	I*4	ARRAY_COUNT
2-00000014	I*4	ARRAY_ERROR_REGISTER	2-000006DD	CHAR	ARRAY_INIT_STATUS
2-00000000	I*4	ARRAY_INIT_STATUS_BITS	2-00000018	I*4	CONFIGURATION_STATUS_REGISTER0
2-0000001C	I*4	CONFIGURATION_STATUS_REGISTER1	2-000006DC	L*1	DIAGNOSTIC_MODE
2-00000000	I*4	ERROR_ARRAY	2-00000710	I*4	ERROR_BANK
2-00000000	I*4	ERROR_BIT	2-00000000	I*4	ERROR_SYNDROME
2-00000714	I*4	I	2-00000718	I*4	J
AP-00000004@	L*1	LUN	2-00000020	I*4	MAINTENANCE_CONTROL_REGISTER
AP-00000008@	I*4	MEMORY_REGISTERS	2-00000004	I*4	PORT_CONFIGURATION_REGISTER
2-0000000C	I*4	PORT_CONTROLLER_STATUS_REGISTER	2-00000008	I*4	PORT_INTERFACE_CONTROL_REGISTER
2-00000010	I*4	PORT_INVALIDATION_CONTROL_REG	2-000006F8	CHAR	PORT_TYPE
2-00000000	I*4	PORT_TYPE_BITS	2-00000000	I*4	RAM_COUNT
2-00000000	I*4	STARTING_ADDRESS			

MS78

PROG

0
1
2

ENTR

0-

VARI

2-
2-
2-
2-
AP-
2-
2-

ARP.A

2-(
2-(
2-(
2-(
2-(
2-

LABE

1-
1-

ARRAYS

Address	Type	Name	Bytes	Dimensions
2-00000004	I*4	BUFFER	32	(8)
2-00000024	CHAR	V1REGISTER2	64	(0:1)
2-00000172	CHAR	V1REGISTER3	23	(1)
2-00000618	CHAR	V1REGISTER4	14	(31:31)
2-00000626	CHAR	V1REGISTER5	72	(28:31)
2-0000066E	CHAR	V1REGISTER6	54	(0:1)
2-000006A4	CHAR	V1REGISTER7	56	(10:11)
2-000001B9	CHAR	V2REGISTER3	75	(6:8)
2-00000064	CHAR	V3REGISTER2	270	(23:31)
2-000001D4	CHAR	V3REGISTER3	150	(10:15)
2-0000026A	CHAR	V4REGISTER3	630	(21)
2-000004E0	CHAR	V5REGISTER3	184	(22:25, 0:1)
2-00000598	CHAR	V6REGISTER3	128	(28:31)

LABELS

Address	Label	Address	Label	Address	Label	Address	Label	Address	Label	Address	Label
1-000000F4	5'	1-00000109	10'	1-0000011B	15'	1-00000136	17'	1-0000014D	20'	1-0000015E	25'
1-0000017C	30'	1-0000018F	40'	**	50	1-000001BD	60'	1-000001C4	70'	1-000001EA	80'
1-00000212	90'	1-00000222	100'	1-0000023E	110'	1-00000245	120'	1-00000253	140'	1-00000274	145'
1-0000029A	170'	1-000002AB	180'	1-000002B2	190'	**	200	1-000002C0	210'	**	220
1-000002CE	230'	**	240	1-000002DC	250'	1-000002ED	260'	**	270	1-00000307	280'
**	290	1-00000315	295'								

FUNCTIONS AND SUBROUTINES REFERENCED

Type	Name	Type	Name	Type	Name	Type	Name	Type	Name	Type	Name
I*4	COMPRESS4	I*4	COMPRESSC	I*4	DECODE_ECC	I*4	LIB$EXTZV	I*4	LIB$LOCC		LINCHK
	MA780_REGA		MOVC3		OUTPUT						

0001

```
0002            Subroutine MS780C (lun,memory_registers)
0003
0004    c++
0005    c       This routine displays the error log packet for the ms780c
0006    c       memory controller.  The format of the packet is as follows.
0007    c
0008    r       +-----------------------------------+
0009    c       :          memory register A         :
0010    c       +-----------------------------------+
0011    c       :          memory register B         :
0012    c       +-----------------------------------+
0013    c       :          memory register C         :
0014    c       +-----------------------------------+
0015    c--
0016
0017
0018            Implicit        none
0019
0020            byte            lun
0021
0022            integer*4       memory_registers
0023            integer*4       adapter_tr
0024            integer*4       buffer(3)
0025            integer*4       memory_register_a
0026            integer*4       memory_register_b
0027            integer*4       memory_register_c
0028
0029            equivalence     (buffer(1),memory_register_a)
0030            equivalence     (buffer(2),memory_register_b)
0031            equivalence     (buffer(3),memory_register_c)
0032
0033            character*27    memory_init_status(0:3)
0034            data            memory_init_status(0)    /'INITIALIZATION IN PROGRESS*'/
0035            data            memory_init_status(1)    /'MEMORY CONTAINS VALID DATA*'/
0036            data            memory_init_status(2)    /'INVALID STATE*'/
0037            data            memory_init_status(3)    /'INITIALIZATION COMPLETE*'/
0038
0039            character*27    v2memory_registerb(14:14)
0040            data            v2memory_registerb(14)   /'START ADDRESS WRITE ENABLE*'/
0041
0042            character*18    v1memory_registerc(28:30)
0043            data            v1memory_registerc(28)   /'ERROR LOG REQUEST*'/
0044            data            v1memory_registerc(29)   /'HIGH ERROR RATE*'/
0045            data            v1memory_registerc(30)   /'INHIBIT CRD TAG*'/
0046
0047            integer*4       lib$extzv
0048            Integer*4       I
0049            integer*4       decode_ecc
0050            integer*4       compress4
0051            integer*4       compressc
0052            integer*4       init_status
0053            integer*4       starting_address
0054            integer*4       error_syndrome
0055            integer*4       error_bit
0056            integer*4       error_bank_address
0057            integer*4       error_array
0058
```

```
0059              logical*1         diagnostic_mode
0060              Logical*1         four_k
0061              Logical*1         sixteen_k
0062
0063
0064
0065              call movc3 (%val(12),memory_registers,buffer)
0066
0067              diagnostic_mode = .false.
0068
0069              if (lib$extzv (8,2,memory_register_b) .ne. 0) diagnostic_mode = .true.
0070
0071              if (.not. diagnostic_mode) then
0072
0073              call ms780c_rega (lun,memory_register_a)
0074              else
0075
0076              call linchk (lun,1)
0077
0078              write(lun,5) memory_register_a
0079        5     format(' ',t8,'CSRA',t24,z8.8)
0080              endif
0081
0082              Four_k = .false.
0083              Sixteen_k = .false.
0084
0085              If (LIB$EXTZV(3,2,memory_register_a) .EQ. 1) four_k = .true.
0086              If (LIB$EXTZV(3,2,memory_register_a) .EQ. 2) sixteen_k = .true.
0087
0088              call linchk (lun,1)
0089
0090              write(lun,10) 'CSRB',memory_register_b
0091        10    format(' ',t8,a,t24,z8.8)
0092
0093              if (diagnostic_mode) then
0094
0095              call linchk (lun,1)
0096
0097              write(lun,12) 'DIAGNOSTIC MODE'
0098        12    format(' ',t40,a)
0099              endif
0100
0101              if (.not. diagnostic_mode) then
0102
0103              init_status = lib$extzv(12,2,memory_register_b)
0104
0105              call linchk (lun,1)
0106
0107              write(lun,30) memory_init_status(init_status)
0108        30    format(' ',t40,a<compressc (memory_init_status(init_status))>)
0109
0110              call output (lun,memory_register_b,v2memory_registerb,14,14,14,'0')
0111
0112              starting_address = lib$extzv(15,13,memory_register_b)*64
0113
0114              call linchk (lun,1)
0115
```

```
0116          write(lun,35) starting_address
0117    35    format(' ',t40,'MEMORY BASE ADDRESS = ',
0118        1 i<compress4 (starting_address)>,'.K')
0119
0120          call linchk (lun,2)
0121
0122          write(lun,40) (lib$extzv(i,2,memory_register_b),i = 28,30,2)
0123    40    format(' ',t40,'FILE INPUT POINTER ',
0124        1 i<compress4 (lib$extzv(i,c,memory_register_b))>,'.',/,
0125        1 t40,'FILE OUTPUT POINTER ',
0126        1 i<compress4 (lib$extzv(i,2,memory_register_b))>,'.')
0127          endif
0128
0129          call linchk (lun,1)
0130
0131          write(lun,45) 'CSRC',memory_register_c
0132    45    format(' ',t8,a,t24,z8.8)
0133
0134          if (.not. diagnostic_mode) then
0135
0136          if (lib$extzv(28,1,memory_register_c) .eq. 1) then
0137
0138          error_syndrome = lib$extzv(0,8,memory_register_c)
0139
0140          call linchk (lun,1)
0141
0142          write(lun,15) error_syndrome
0143    15    format(' ',t40,'ERROR SYNDROME = ',z2.2)
0144
0145          error_bit = decode_ecc (error_syndrome,memory_register_a)
0146
0147          call linchk (lun,1)
0148
0149          if (error_bit .eq. -1) then
0150
0151          write(lun,20) 'RDS ERROR'
0152    20    format(' ',t40,a)
0153          else
0154
0155          write(lun,25) 'CRD ERROR, CORRECTED BIT #',error_bit,'.'
0156    25    format(' ',t40,a,i<compress4 (error_bit)>,a)
0157          endif
0158
0159          If (four_k) then
0160          Error_bank_address = LIB$EXTZV(21,1,memory_register_c)
0161
0162          Else if (sixteen_k) then
0163          Error_bank_address = LIB$EXTZV(23,1,memory_register_c)
0164
0165          Endif
0166
0167          call linchk (lun,1)
0168
0169          write(lun,47) error_bank_address
0170    47    format(' ',t40,'ARRAY BANK #',
0171        1 i<compress4 (error_bank_address)>,'. IN ERROR')
0172
```

0116
0117
0118
0119
0120
0121
0122
0123
0124
0125
0126
0127
0128
0129
0130
0131
0132
0133
0134
0135
0136
0137
0138
0139
0140
0141
0142
0143
0144
0145
0146
0147
0148
0149
0150
0151
0152
0153
0154
0155
0156
0157
0158
0159
0160
0161
0162
0163
0164
0165
0166
0167
0168
0169
0170
0171
0172

```
0173              error_array = lib$extzv(24,4,memory_register_c)
0174
0175              call linchk (lun,1)
0176
0177              write(lun,50) error_array
0178     50       format(' ',t40,'ARRAY #',i<compress4 (error_array)>,'. IN ERROR')
0179              endif
0180
0181              call output (lun,memory_register_c,v1memory_registerc,28,28,30,'0')
0182              endif
0183
0184              return
0185              end
```

PROGRAM SECTIONS

	Name	Bytes	Attributes
0	$CODE	1100	PIC CON REL LCL SHR EXE RD NOWRT LONG
1	$PDATA	411	PIC CON REL LCL SHR NOEXE RD NOWRT LONG
2	$LOCAL	716	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
	Total Space Allocated	2227	

ENTRY POINTS

Address	Type	Name
0-0000 00		MS780C

VARIABLES

Address	Type	Name	Address	Type	Name
2-000000CC	I*4	ADAPTER_TR	2-000000C9	L*1	DIAGNOSTIC_MODE
2-000000E8	I*4	ERROR_ARRAY	2-000000E4	I*4	ERROR_BANK_ADDRESS
2-000000E0	I*4	ERROR_BIT	2-000000DC	I*4	ERROR_SYNDROME
2-000000CA	L*1	FOUR_K	2-000000D0	I*4	I
2-000000D4	I*4	INIT_STATUS	AP-00000004@	L*1	LUN
AP-00000008@	I*4	MEMORY_REGISTERS	2-00000000	I*4	MEMORY_REGISTER_A
2-00000004	I*4	MEMORY_REGISTER_B	2-00000008	I*4	MEMORY_REGISTER_C
2-000000CB	L*1	SIXTEEN_K	2-000000D8	I*4	STARTING_ADDRESS

ARRAYS

Address	Type	Name	Bytes	Dimensions
2-00000000	I*4	BUFFER	12	(3)
2-0000000C	CHAR	MEMORY_INIT_STATUS	108	(0:3)
2-00000093	CHAR	V1MEMORY_REGISTERC	54	(28:30)
2-00000078	CHAR	V2MEMORY_REGISTERB	27	(14:14)

MS750

0173
0174
0175
0176
0177
0178
0179
0180
0181
0182
0183
0184
0185
0186
0187
0188
0189
0190
0191
0192
0193
0194
0195
0196
0197
0198
0199
0200
0201
0202
0203
0204
0205
0206
0207
0208
0209
0210
0211
0212
0213
0214
0215
0216
0217
0218
0219
0220
0221
0222
0223
0224
0225
0226
0227
0228
0229

LABELS

Address	Label	Address	Label	Address	Label	Address	Label	Address	Label	Address	Label
1-00000079	5'	1-0000008A	10'	1-00000096	12'	1-00000123	15'	1-0000013F	20'	1-00000146	25'
1-0000009D	30'	1-000000A9	35'	1-000000D1	40'	1-00000117	45'	1-00000154	47'	1-0000017A	50'

FUNCTIONS AND SUBROUTINES REFERENCED

Type	Name	Type	Name	Type	Name	Type	Name	Type	Name	Type	Name
I*4	COMPRESS4	I*4	COMPRESSC	I*4	DECODE_ECC	I*4	LIB$EXTZV		LINCHK		MOVC3
	MS780C_REGA		OUTPUT								

MS750

0230
0231
0232
0233
0234
0235
0236
0237
0238
0239
0240
0241
0242
0243
0244
0245
0246
0247
0248
0249
0250
0251
0252
0253
0254
0255
0256
0257
0258
0259
0260
0261
0262
0263
0264
0265
0266
0267
0268
0269
0270
0271

0001

MS750

PROG

0
1
2

ENTR

0-

VARI

2-
2-
2-
2-
2-
AP-
2-
2-

ARRA

2-
2-
2-
2-

LABE

1-
1-
1-

```
0002          Subroutine MS780E (lun,memory_registers)
0003
0004    c++
0005    c        This routine displays the error log packet for the ms780e
0006    c        memory controller.  The format of the packet is as follows.
0007    c
0008    c        +--------------------------------------+
0009    c        :          memory register A           :
0010    c        +--------------------------------------+
0011    c        :          memory register B           :
0012    c        +--------------------------------------+
0013    c        :          memory register C           :
0014    c        +--------------------------------------+
0015    c        :          memory register D           :
0016    c        +--------------------------------------+
0017    c--
0018
0019          Implicit        none
0020
0021          byte            lun
0022
0023          integer*4       memory_registers
0024          integer*4       buffer(4)
0025          integer*4       memory_register_a
0026          integer*4       memory_register_b
0027          integer*4       memory_register('c'x:'d'x)
0028
0029          equivalence     (buffer(1),memory_register_a)
0030          equivalence     (buffer(2),memory_register_b)
0031          equivalence     (buffer(3),memory_register)
0032
0033          logical*1       diagnostic_mode
0034
0035          integer*4       decode_ecc
0036          integer*4       compress4
0037          integer*4       compressc
0038          Integer*4       lib$extzv
0039          Integer*4       I
0040
0041          character*27    memory_init_status(0:3)
0042          data            memory_init_status(0)    /'INITIALIZATION IN PROGRESS*'/
0043          data            memory_init_status(1)    /'MEMORY CONTAINS VALID DATA*'/
0044          data            memory_init_status(2)    /'INVALID STATE*'/
0045          data            memory_init_status(3)    /'INITIALIZATION COMPLETE*'/
0046
0047          character*33    v1memory_registerb(7:7)
0048          data            v1memory_registerb(7)
0049        1 /'SBI INTERFACE WRITE PARITY ERROR*'/
0050
0051          character*27    v2memory_registerb(14:14)
0052          data            v2memory_registerb(14)  /'START ADDRESS WRITE ENABLE*'/
0053
0054          character*29    v1memory_register(7:7)
0055          data            v1memory_register(7)
0056        1 /'MICRO-SEQUENCER PARITY ERROR*'/
0057
0058          character*18    v2memory_register(28:30)
```

```
0059          data            v2memory_register(28)   /'ERROR LOG REQUEST*'/
0060          data            v2memory_register(29)   /'HIGH ERROR RATE*'/
0061          data            v2memory_register(30)   /'INHIBIT CRD TAG*'/
0062
0063          integer*4       init_status
0064          integer*4       starting_address
0065          integer*4       error_syndrome
0066          integer*4       error_bit
0067          integer*4       error_bank_address
0068          integer*4       error_array
0069          integer*4       adapter_tr
0070
0071
0072          call movc3 (%val(16),memory_registers,buffer)
0073
0074          diagnostic_mode = .false.
0075
0076          if (
0077        1 lib$extzv(7,3,memory_register_b) .ne. 0
0078        1 .or.
0079        1 lib$extzv(11,1,memory_register_b) .eq. 1
0080        1 .or.
0081        1 lib$extzv(31,1,memory_register('c'x)) .eq. 1
0082        1 .or.
0083        1 lib$extzv(31,1,memory_register('d'x)) .eq. 1
0084        1 ) then
0085
0086          diagnostic_mode = .true.
0087          endif
0088
0089          if (.not. diagnostic_mode) then
0090
0091          call ms780e_rega (lun,memory_register_a)
0092          else
0093
0094          call linchk (lun,1)
0095
0096          write(lun,5) memory_register_a
0097    5     format(' ',t8,'CSRA',t24,z8.8)
0098          endif
0099
0100          call linchk (lun,1)
0101
0102          write(lun,10) 'CSRB',memory_register_b
0103    10    format(' ',t8,a,t24,z8.8)
0104
0105          if (diagnostic_mode) then
0106
0107          call linchk (lun,1)
0108
0109          write(lun,12) 'DIAGNOSTIC MODE'
0110    12    format(' ',t40,a)
0111          endif
0112
0113          if (.not. diagnostic_mode) then
0114
0115          init_status = lib$extzv(12,2,memory_register_b)
```

0001

```
0116
0117              call linchk (lun,1)
0118
0119              write(lun,15) memory_init_status(init_status)
0120      15      format(' ',t40,a<compressc (memory_init_status(init_status))>)
0121
0122              call output (lun,memory_register_b,v1memory_registerb,7,7,7,'0')
0123
0124              call output (lun,memory_register_b,v2memory_registerb,14,14,14,'0')
0125
0126              starting_address = lib$extzv(19,9,memory_register_b)
0127
0128              call linchk (lun,1)
0129
0130              write(lun,20) starting_address
0131      20      format(' ',t40,'MEMORY BASE ADDRESS = ',
0132             1 i<compress4 (starting_address)>,'.M')
0133              endif
0134
0135              do 55,i = 'c'x,'d'x
0136
0137              call linchk (lun,1)
0138
0139              if (i .eq. 'c'x) then
0140
0141              write(lun,25) 'C',memory_register(i)
0142      25      format(' ',t8,'CSR',a,t24,z8.8)
0143
0144              else if (i .eq.'d'x) then
0145
0146              write(lun,25) 'D',memory_register(i)
0147              endif
0148
0149              if (.not. diagnostic_mode) then
0150
0151              if (lib$extzv(28,1,memory_register(i)) .eq. 1) then
0152
0153              error_syndrome = lib$extzv(0,7,memory_register(i))
0154
0155              call linchk (lun,1)
0156
0157              write(lun,30) error_syndrome
0158      30      format(' ',t40,'ERROR SYNDROME = ',z2.2)
0159
0160              error_bit = decode_ecc (error_syndrome,memory_register_a)
0161
0162              call linchk (lun,1)
0163
0164              if (error_bit .eq. -1) then
0165
0166              write(lun,35) 'RDS ERROR'
0167      35      format(' ',t40,a)
0168              else
0169
0170              write(lun,40) 'CRD ERROR, CORRECTED BIT #',error_bit,'.'
0171      40      format(' ',t40,a,i<compress4 (error_bit)>,a)
0172              endif
```

0002
0003
0004
0005
0006
0007
0008
0009
0010
0011
0012
0013
0014
0015
0016
0017
0018
0019
0020
0021
0022
0023
0024
0025
0026
0027
0028
0029
0030
0031
0032
0033
0034
0035
0036
0037
0038
0039
0040
0041
0042
0043
0044
0045
0046
0047
0048
0049
0050
0051
0052
0053
0054
0055
0056
0057
0058

```
0173
0174              call output (lun,memory_register(i),v1memory_register,7,7,7,'0')
0175
0176              error_bank_address = lib$extzv(22,2,memory_register(i))
0177
0178              call linchk (lun,1)
0179
0180              write(lun,45) error_bank_address
0181        45    format(' ',t40,'ARRAY BANK #',
0182              1 i<compress4 (error_bank_address)>,'. IN ERROR')
0183
0184              error_array = lib$extzv(24,3,memory_register(i))
0185
0186              If (i .EQ. 'd'x) then
0187              Error_array = error_array + 8
0188
0189              Endif
0190
0191              call linchk (lun,1)
0192
0193              write(lun,50) error_array
0194        50    format(' ',t40,'ARRAY #',i<compress4 (error_array)>,'. IN ERROR')
0195              endif
0196
0197              call output (lun,memory_register(i),v2memory_register,28,28,30,'0')
0198              endif
0199
0200        55    continue
0201
0202              return
0203              end
```

PROGRAM SECTIONS

	Name	Bytes	Attributes
0	$CODE	1098	PIC CON REL LCL SHR EXE RD NOWRT LONG
1	$PDATA	344	PIC CON REL LCL SHR NOEXE RD NOWRT LONG
2	$LOCAL	888	PIC CON REL LCL NOSHR NOEXE RD WRT LONG

Total Space Allocated   2330

ENTRY POINTS

Address	Type	Name
0-00000000		MS780E

VARIABLES

Address	Type	Name	Address	Type	Name
2-00000128	I*4	ADAPTER_TR	2-0000010B	L*1	DIAGNOSTIC_MODE
2-00000124	I*4	ERROR_ARRAY	2-00000120	I*4	ERROR_BANK_ADDRESS
2-0000011C	I*4	ERROR_BIT	2-00000118	I*4	ERROR_SYNDROME
2-0000010C	I*4	I	2-00000110	I*4	INIT_STATUS
AP-0000004a	L*1	LUN	AP-0000008a	I*4	MEMORY_REGISTERS
2-00000000	I*4	MEMORY_REGISTER_A	2-00000004	I*4	MEMORY_REGISTER_B
2-00000114	I*4	STARTING_ADDRESS			

ARRAYS

Address	Type	Name	Bytes	Dimensions
2-00000000	I*4	BUFFER	16	(4)
2-00000010	CHAR	MEMORY_INIT_STATUS	108	(0:3)
2-00000008	I*3	MEMORY_REGISTER	8	(12:13)
2-000000B8	CHAR	V1MEMORY_REGISTER	29	(7:7)
2-0000007C	CHAR	V1MEMORY_REGISTERB	33	(7:7)
2-000000D5	CHAR	V2MEMORY_REGISTER	54	(28:30)
2-0000009D	CHAR	V2MEMORY_REGISTERB	27	(14:14)

LABELS

Address	Label	Address	Label	Address	Label	Address	Label	Address	Label	Address	Label
1-00000077	5'	1-00000088	10'	1-00000094	12'	1-0000009B	15'	1-000000A7	20'	1-000000CF	25'
1-000000E0	30'	1-000000FC	35'	1-00000103	40'	1-00000111	45'	1-00000137	50'	**	55

MS73
0116
0117
0118
0119
0120
0121
0122
0123
0124
0125
0126
0127
0128
0129
0130
0131
0132
0133
0134
0135
0136
0137
0138
0139
0140
0141
0142
0143
0144
0145
0146
0147
0148
0149
0150
0151
0152
0153
0154
0155
0156
0157
0158
0159
0160
0161
0162
0163
0164
0165
0166
0167
0168
0169
0170
0171
0172

FUNCTIONS AND SUBROUTINES REFERENCED

Type	Name	Type	Name	Type	Name	Type	Name	Type	Name	Type	Name
I*4	COMPRESS4	I*4	COMPRESSC	I*4	DECODE_ECC	I*4	LIBSEXTZV		LINCHK		MOVC3
	MS780E_REGA		OUTPUT								

MS73

0173
0174
0175
0176
0177
0178
0179
0180
0181
0182
0183
0184
0185
0186
0187

PROG

0
1
2

ENTR

0-

VARI

2-
2-
2-
AP-
2-
AP-

ARRA

2-
2-
2-
2-

0001

```
0002            Subroutine MS750 (lun,memory_registers)
0003
0004    c++
0005    c
0006    c       Functional description:
0007    c
0008    c       This module displays ms750 memory error packets.  The format is as
0009    c       follows.
0010    c
0011    c       +-------------------------------+
0012    c       :          memory register 0    :
0013    c       +-------------------------------+
0014    c       :          memory register 1    :
0015    c       +-------------------------------+
0016    c       :          memory register 2    :
0017    c       +-------------------------------+
0018    c--
0019
0020            Implicit        none
0021
0022            byte            lun
0023
0024            integer*4       memory_registers
0025            integer*4       slot_index
0026            integer*4       buffer(3)
0027            integer*4       memory_register_0
0028            integer*4       memory_register_1
0029            integer*4       memory_register_2
0030
0031            equivalence     (buffer(1),memory_register_0)
0032            equivalence     (buffer(2),memory_register_1)
0033            equivalence     (buffer(3),memory_register_2)
0034
0035            character*35    v1memory_register0(29:31)
0036            data            v1memory_register0(29)   /'CORRECTED ERROR FLAG*'/
0037            data            v1memory_register0(30)
0038    1 /'UNCORRECTED ERROR INFORMATION LOST*'/
0039            data            v1memory_register0(31)   /'UNCORRECTED ERROR FLAG*'/
0040
0041            character*34    v1memory_register1(28:28)
0042            data            v1memory_register1(28)
0043    1 /'ENABLE REPORTING CORRECTED ERRORS*'/
0044
0045            character*23    v1memory_register2(16:16)
0046            data            v1memory_register2(16)   /'BATTERY BACKUP FAILURE*'/
0047
0048            integer*4       compress4
0049            integer*4       lib$extzv
0050            Integer*4       I
0051            integer*4       decode_ecc
0052            integer*4       error_bit
0053            integer*4       error_array
0054            Integer*4       error_bank
0055            integer*4       arrays_present
0056            integer*4       start_address
0057
0058            equivalence     (error_bit,error_array,arrays_present,
```

```
0059              1 start_address)
0060
0061              integer*4        array_code
0062
0063              logical*1        diagnostic_mode
0064              logical*1        L0011
0065              logical*1        L0016
0066
0067
0068              call movc3 (%val(12),memory_registers,buffer)
0069
0070              diagnostic_mode = .false.
0071
0072              if (lib$extzv(25,3,memory_register_1) .ne. 0) diagnostic_mode = .true.
0073
0074              call linchk (lun,2)
0075
0076              write(lun,10) memory_register_0
0077     10       format(/' ',t8,'CSR0',t24,z8.8)
0078
0079              if (.not. diagnostic_mode) then
0080
0081              call linchk (lun,1)
0082
0083              write(lun,15) lib$extzv(0,7,memory_register_0)
0084     15       format(' ',t40,'ERROR SYNDROME = ',z2.2)
0085
0086              if (lib$extzv(29,1,memory_register_0) .eq. 1) then
0087
0088              error_bit = decode_ecc (lib$extzv(0,7,memory_register_0))
0089
0090              call linchk (lun,1)
0091
0092              if (error_bit .eq. -1) then
0093
0094              write(lun,20) '"ECC" CODE, UNCORRECTED ERROR'
0095              else
0096
0097              write(lun,20) 'CORRECTED ERROR, BIT #',error_bit,'.'
0098     20       format(' ',t40,a,:i<compress4 (error_bit)>,:a)
0099              endif
0100              endif
0101
0102              if (lib$extzv(30,1,memory_register_0) .eq. 0) then
0103
0104              error_array = lib$extzv(9,15,memory_register_0)
0105
0106              L0011 = .false.
0107
0108              L0016 = .false.
0109
0110              if (lib$extzv(24,1,memory_register_2) .eq. 1) then
0111
0112              L0016 = .true.
0113              else
0114
0115              L0011 = .true.
```

```
0116              endif
0117
0118              do 25,i = 0,15,2
0119
0120              array_code = lib$extzv(i,2,memory_register_2)
0121
0122              if (L0016) then
0123
0124              if (array_code .eq. 3) then
0125
0126              error_array = error_array - 512
0127
0128              If (LIB$EXTZV(17,1,memory_register_0) .EQ. 0) then
0129              Error_bank = 0
0130
0131              Else
0132              Error_bank = 1
0133
0134              Endif
0135
0136              else if (array_code .eq. 2) then
0137
0138              error_array = error_array - 2048
0139              Error_bank = LIB$EXTZV(19,2,memory_register_0)
0140
0141              endif
0142
0143              else if (L0011) then
0144
0145              if (array_code .eq. 1) then
0146
0147              error_array = error_array - 256
0148              Error_bank = 0
0149
0150              else if (array_code .eq. 3) then
0151
0152              error_array = error_array - 512
0153
0154              If (LIB$EXTZV(17,1,memory_register_0) .EQ. 0) then
0155              Error_bank = 0
0156
0157              Else
0158              Error_bank = 1
0159
0160              Endif
0161              endif
0162              endif
0163
0164              if (error_array .le. 0) then
0165
0166              error_array = i/2
0167
0168              goto 26
0169              endif
0170
0171     25       continue
0172
```

0060
0061
0062
0063
0064
0065
0066
0067
0068
0069
0070
0071
0072
0073
0074
0075
0076
0077
0078

PROG

0
1
2

ENTR

0-

VARI

2-
2-

ARRA

AP-
2-
3-
2-

```
0173     26        call linchk (lun,1)
0174
0175               write (lun,28) error_bank
0176     28        Format(' ',t40,'ARRAY BANK #',
0177               1 i<COMPRESS4 (error_bank)>,'. IN ERROR')
0178
0179               Call LINCHK (lun,1)
0180
0181               write(lun,30) error_array
0182     30        format(' ',t40,'ARRAY #',i<compress4 (error_array)>,'. IN ERROR')
0183               endif
0184
0185               call output (lun,memory_register_0,v1memory_register0,29,29,31,'0')
0186               endif
0187
0188               call linchk (lun,1)
0189
0190               write(lun,35) memory_register_1
0191     35        format(' ',t8,'CSR1',t24,z8.8)
0192
0193               if (lib$extzv(25,3,memory_register_1) .eq. 0) then
0194
0195               call output (lun,memory_register_1,v1memory_register1,28,28,28,'0')
0196               else
0197
0198               call linchk (lun,1)
0199
0200               write(lun,40) 'DIAGNOSTIC MODE'
0201     40        format(' ',t40,a)
0202               endif
0203
0204               call linchk (lun,1)
0205
0206               write(lun,45) memory_register_2
0207     45        format(' ',t8,'CSR2',t24,z8.8)
0208
0209               if (.not. diagnostic_mode) then
0210
0211               arrays_present = 0
0212
0213               do 47,i = 0,15,2
0214
0215               array_code = lib$extzv(i,2,memory_register_2)
0216
0217               if (L0016) then
0218
0219               if (array_code .eq. 3) then
0220
0221               arrays_present = arrays_present + 2
0222
0223               else if (array_code .eq. 2) then
0224
0225               arrays_present = arrays_present + 8
0226               endif
0227
0228               else if (L0011) then
0229
```

```
0230              if (array_code .eq. 1) then
0231
0232              arrays_present = arrays_present + 1
0233
0234              else if (array_code .eq. 3) then
0235
0236              arrays_presen. = arrays_present + 2
0237              endif
0238              endif
0239
0240      47      continue
0241
0242              call linchk (lun,1)
0243
0244              write(lun,50) arrays_present*128
0245      50      format(' ',t40,'MEMORY SIZE = '
0246              1 i<compress4 (arrays_present*128)>,'.K')
0247
0248              call output (lun,memory_register_2,v1memory_register2,16,16,16,'0')
0249
0250              start_address = lib$extzv (17,7,memory_register_2)
0251
0252              call linchk (lun,1)
0253
0254              write(lun,55) start_address*128
0255      55      format(' ',t40,'MEMORY BASE ADDRESS = ',
0256              1 i<compress4 (start_address*128)>,'.K')
0257              endif
0258
0259              call linchk (lun,1)
0260
0261              if (l0016) then
0262
0263              write(lun,60) 'L0016'
0264      60      format(' ',t40,'CONTROLLER IS ',a)
0265              else
0266
0267              write(lun,60) 'L0011'
0268              endif
0269
0270              return
0271              end
```

0001

PROGRAM SECTIONS

     Name   Bytes   Attributes

   0 $CODE   1271   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
   1 $PDATA   412   PIC CON REL LCL   SHR NOEXE   RD NOWRT LONG
   2 $LOCAL   680   PIC CON REL LCL NOSHR NOEXE   RD   WRT LONG

     Total Space Allocated   2363

ENTRY POINTS

     Address Type Name

   0-00000000   MS750

VARIABLES

     Address Type Name   Address Type Name

   2-00000000 I*4 ARRAYS_PRESENT   2-000000C4 I*4 ARRAY_CODE
   2-000000B2 L*1 DIAGNOSTIC_MODE   2-00000000 I*4 ERROR_ARRAY
   2-000000C0 I*4 ERROR_BANK   2-00000000 I*4 ERROR_BIT
   2-000000BC I*4 I   2-000000B3 L*1 L0011
   2-000000B4 L*1 L0016   AP-0000004a L*1 LUN
  AP-0000008a I*4 MEMORY_REGISTERS   2-00000004 I*4 MEMORY_REGISTER_0
   2-00000008 I*4 MEMORY_REGISTER_1   2-0000000C I*4 MEMORY_REGISTER_2
   2-000000B8 I*4 SLOT_INDEX   2-00000000 I*4 START_ADDRESS

ARRAYS

     Address Type Name   Bytes Dimensions

   2-00000004 I*4 BUFFER   12 (3)
   2-00000010 CHAR V1MEMORY_REGISTER0   105 (29:31)
   2-00000079 CHAR V1MEMORY_REGISTER1   34 (28:28)
   2-0000009B CHAR V1MEMORY_REGISTER2   23 (16:16)

LABELS

    Address   Label   Address   Label   Address   Label   Address   Label   Address   Label   Address   Label

   1-0000008F 10'   1-000000A1 15'   1-000000BD 20'   **   25   0-0000022E 26   1-000000CD 28'
   1-000000F3 30'   1-00000114 35'   1-00000125 40'   1-0000012C 45'   **   47   1-0000013D 50'
   1-0000015D 55'   1-00000185 60'

0002
0003
0004
0005
0006
0007
0008
0009
0010
0011
0012
0013
0014
0015
0016
0017
0018
0019
0020
0021
0022
0023
0024
0025
0026
0027
0028
0029
0030
0031
0032
0033
0034
0035
0036
0037
0038
0039
0040
0041
0042
0043
0044
0045
0046
0047
0048
0049
0050
0051
0052
0053
0054
0055
0056
0057
0058

MS750   M 5
16-Sep-1984 00:07:33   VAX-11 FORTRAN V3.4-56   Page 44
5-Sep-1984 14:01:18   DISK$VMSMASTER:[ERF.SRC]MEMORYS.FOR;1

FUNCTIONS AND SUBROUTINES REFERENCED

Type	Name	Type	Name	Type	Name	Type	Name	Type	Name	Type	Name
I*4	COMPRESS4	I*4	DECODE_ECC	I*4	LIB$EXTZV		LINCHK		MOVC3		OUTPUT

0059
0060
0061
0062
0063
0064
0065
0066
0067
0068
0069
0070
0071
0072
0073
0074
0075
0076
0077
0078
0079
0080
0081
0082
0083
0084
0085
0086
0087
0088
0089
0090
0091
0092
0093
0094
0095
0096
0097
0098
0099
0100
0101
0102
0103
0104
0105
0106
0107
0108
0109
0110
0111
0112
0113
0114
0115

0001

MEMO

0116
0117
0118
0119
0120
0121
0122
0123
0124
0125
0126
0127
0128
0129
0130
0131
0132
0133
0134
0135
0136
0137
0138
0139
0140
0141
0142
0143
0144
0145
0146
0147
0148
0149
0150
0151
0152
0153
0154
0155
0156
0157
0158
0159
0160
0161
0162
0163
0164
0165
0166
0167
0168
0169
0170
0171
0172

```
0002            Subroutine MS730 (lun,memory_register_0)
0003
0004     c++
0005     c       Functional description:
0006     c
0007     c       This routine displays the memory registers for the 11/730.  The
0008     c       format of the buffer is as follows.
0009     c
0010     c       +----------------------------------+
0011     c       :           memory csr0            :
0012     c       +----------------------------------+
0013     c       :           memory csr1            :
0014     c       +----------------------------------+
0015     c       :           memory csr2            :
0016     c       +----------------------------------+
0017     c--
0018
0019            Implicit        none
0020
0021            byte            lun
0022
0023            integer*4       memory_register_0
0024            integer*4       buffer(3)
0025            integer*4       memory_csr0
0026            integer*4       memory_csr1
0027            integer*4       memory_csr2
0028
0029            equivalence     (buffer(1),memory_csr0)
0030            equivalence     (buffer(2),memory_csr1)
0031            equivalence     (buffer(3),memory_csr2)
0032
0033            logical*1       diagnostic_mode
0034
0035            integer*4       compress4
0036            integer*4       decode_ecc
0037            integer*4       error_bit
0038            integer*4       error_array
0039            Integer*4       error_bank
0040            integer*4       kilo_bytes_present
0041            Integer*4       lib$extzv
0042            Integer*4       I
0043
0044            character*23    v1memory_register_1(27:28)
0045            data            v1memory_register_1(27) /'MEMORY MAPPING ENABLE*'/
0046            data            v1memory_register_1(28) /'ENABLE "CRD" REPORTING*'/
0047
0048            character*12    v2memory_register_1(30:30)
0049            data            v2memory_register_1(30) /'"CRD" ERROR*'/
0050
0051            character*17    v1memory_register_2(24:24,0:1)
0052            data            v1memory_register_2(24,0) /'16K RAMS PRESENT*'/
0053            data            v1memory_register_2(24,1) /'64K RAMS PRESENT*'/
0054
0055
0056
0057            call movc3 (%val(12),memory_register_0,buffer)
0058
```

MEMO

0173
0174
0175
0176
0177
0178
0179
0180
0181
0182
0183
0184
0185
0186
0187
0188
0189
0190
0191
0192
0193
0194
0195
0196
0197
0198
0199
0200
0201
0202
0203
0204
0205
0206
0207
0208
0209
0210
0211
0212
0213
0214
0215
0216
0217
0218
0219
0220
0221
0222
0223
0224
0225
0226
0227
0228
0229

```
0059              diagnostic_mode = .false.
0060
0061              if (iand(memory_csr1,'26000000'x) .ne. 0) diagnostic_mode = .true.
0062
0063              call linchk (lun,2)
0064
0065              write(lun,10) memory_csr0
0066        10    format(/' ',t8,'CSR0',t24,z8.8)
0067
0068              if (.not. diagnostic_mode) then
0069
0070       c
0071       c      11/730 syndrome bits are inverted so...
0072       c
0073
0074              call linchk (lun,1)
0075
0076              write(lun,15) lib$extzv(0,7,not(lib$extzv(0,7,memory_csr0)))
0077        15    format(' ',t40,'ERROR SYNDROME = ',z2.2)
0078
0079              error_bit = decode_ecc (lib$extzv(0,7,not(lib$extzv(0,7,memory_csr0))))
0080
0081              call linchk (lun,1)
0082
0083              if (error_bit .eq. -1) then
0084
0085              write(lun,20) '"ECC" CODE, UNCORRECTED ERROR'
0086              else
0087
0088              write(lun,20) 'CORRECTED ERROR, BIT #',error_bit,'.'
0089        20    format(' ',t40,a,:i<compress4 (error_bit)>,:a)
0090              endif
0091
0092              error_array = lib$extzv(9,15,memory_csr0)
0093
0094              If (LIB$EXTZV(24,1,memory_csr2) .EQ. 1) then
0095              Error_bank = LIB$EXTZV(19,2,memory_csr0)
0096
0097              Else
0098              Error_bank = LIB$EXTZV(17,1,memory_csr0)
0099
0100              Endif
0101
0102              Call LINCHK (lun,1)
0103              Write (lun,22) error_bank
0104        22    Format(' ',T40,'ARRAY BANK #',
0105             1 I<COMPRESS4 (error_bank)>,'. IN ERROR')
0106
0107              do 25,i = 0,15
0108
0109              if (lib$extzv(i,1,memory_csr2) .eq. 1) then
0110
0111              if (lib$extzv(24,1,memory_csr2) .eq. 1) then
0112
0113              error_array = error_array - 1024
0114              else
0115
```

```
0116            error_array = error_array - 256
0117            endif
0118            endif
0119
0120            if (error_array .le. 0) then
0121
0122            error_array = i/2
0123
0124            goto 27
0125            endif
0126
0127     25     continue
0128
0129     27     call linchk (lun,1)
0130
0131            write(lun,30) error_array
0132     30     format(' ',t40,'ARRAY #',i<compress4 (error_array)>,'. IN ERROR')
0133            endif
0134
0135            call linchk (lun,1)
0136
0137            write(lun,35) memory_csr1
0138     35     format(' ',t8,'CSR1',t24,z8.8)
0139
0140            if (.not. diagnostic_mode) then
0141
0142            call output (lun,memory_csr1,v1memory_register_1,27,27,28,'0')
0143
0144            call output (lun,memory_csr1,v2memory_register_1,30,30,30,'0')
0145            else
0146
0147            call linchk (lun,1)
0148
0149            write(lun,40) 'DIAGNOSTIC MODE'
0150     40     format(' ',t40,a)
0151            endif
0152
0153            call linchk (lun,1)
0154
0155            write(lun,45) memory_csr2
0156     45     format(' ',t8,'CSR2',t24,z8.8)
0157
0158            if (.not. diagnostic_mode) then
0159
0160            kilo_bytes_present = 0
0161
0162            do 50,i = 0,15
0163
0164            if (lib$extzv(i,1,memory_csr2) .eq. 1) then
0165
0166            if (lib$extzv(24,1,memory_csr2) .eq. 1) then
0167
0168            kilo_bytes_present = kilo_bytes_present + 512
0169            else
0170
0171            kilo_bytes_present = kilo_bytes_present + 128
0172            endif
```

028`
0288
028`
0290
029`
029`
029`
029`
029`
029`
029`
029`
0300
030`
030`
030`
030`
030`
030`
030`
0308
0308
0310
031`
031`
031`
0316
031`
0318
031`
0320
032`
032`
032`
0324
032`
032`
032`
032`
032`
0330
033`
033`
033`
033`
033`
033`
033`
0340
034`
034`
034`

```
0173            endif
0174
0175    50      continue
0176
0177            call linchk (lun,1)
0178
0179            write(lun,55) kilo_bytes_present
0180    55      format(' ',t40,'MEMORY SIZE = ',i<compress4 (kilo_bytes_present)>,
0181           1 ',.K')
0182
0183            call output (lun,memory_csr2,v1memory_register_2,24,24,24,'2')
0184            endif
0185
0186            return
0187            end
```

PROGRAM SECTIONS

Name	Bytes	Attributes
0 $CODE	925	PIC CON REL LCL SHR EXE RD NOWRT LONG
1 $PDATA	325	PIC CON REL LCL SHR NOEXE RD NOWRT LONG
2 $LOCAL	572	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
Total Space Allocated	1822	

ENTRY POINTS

Address	Type	Name
0-00000000		MS730

VARIABLES

Address	Type	Name	Address	Type	Name
2-00000068	L*1	DIAGNOSTIC_MODE	2-00000070	I*4	ERROR_ARRAY
2-00000074	I*4	ERROR_BANK	2-0000006C	I*4	ERROR_BIT
2-0000007C	I*4	I	2-00000078	I*4	KILO_BYTES_PRESENT
AP-00000004@	L*1	LUN	2-00000000	I*4	MEMORY_CSR0
2-00000004	I*4	MEMORY_CSR1	2-00000008	I*4	MEMORY_CSR2
AP-00000008@	I*4	MEMORY_REGISTER_0			

ARRAYS

Address	Type	Name	Bytes	Dimensions
2-00000000	I*4	BUFFER	12	(3)
2-0000000C	CHAR	V1MEMORY_REGISTER_1	46	(27:28)
2-00000046	CHAR	V1MEMORY_REGISTER_2	34	(24:24, 0:1)
2-0000003A	CHAR	V2MEMORY_REGISTER_1	12	(30:30)

LABELS

Address	Label	Address	Label	Address	Label	Address	Label	Address	Label	Address	Label
1-00000077	10'	1-00000089	15'	1-000000A5	20'	1-000000B5	22'	**	25	0-000001FE	27.
1-000000DB	30'	1-000000FC	35'	1-0000010D	40'	1-00000114	45'	**	50	1-00000125	55'

FUNCTIONS AND SUBROUTINES REFERENCED

Type	Name	Type	Name	Type	Name	Type	Name	Type	Name	Type	Name
I*4	COMPRESS4	I*4	DECODE_ECC	I*4	LIB$EXTZV		LINCHK		MOVC3		OUTPUT

0001
0002

MEMO

2:
2:
2:
2:
2:
2:
AP:
2:
2:
2:
2:
AP:
AP:
AP:

ARRA

2:
2:
2:
2:
2:

LABE

0-

FUN(

Ty

```
0003          Subroutine MEMORY_REGISTER_UV1 (lun,memory_registers)
0004
0005     C++
0006     C        This routine displays the UVAX1 memory registers.
0007     C        The format of the memory sub packet is as follows:
0008     C
0009     C        +-----------------------------------+
0010     C        :              CSR count            :
0011     C        +-----------------------------------+
0012     C        :              CSR or zero          :
0013     C        +-----------------------------------+
0014     C        .                                   .
0015     C        .                                   .
0016     C        .                                   .
0017     C        +-----------------------------------+
0018     C        :              CSR or zero          :
0019     C        +-----------------------------------+
0020     C--
0021
0022          Implicit        none
0023
0024          byte            lun
0025
0026          integer*4       memory_registers(0:16)
0027          integer*4       error_address1
0028          integer*4       error_address2
0029          integer*4       loop
0030          integer*4       compress4
0031          integer*4       lib$extzv
0032          integer*4       lib$insv
0033
0034          character*20    v1csr(0:0)
0035          data            v1csr(0)        /'PARITY ERROR ENABLE*'/
0036
0037          character*16    v2csr(2:2)
0038          data            v2csr(2)        /'DIAGNOSTIC MODE*'/
0039
0040          character*25    v3csr(14:15)
0041          data            v3csr(14)       /'EXTENDED CSR READ ENABLE*'/
0042          data            v3csr(15)       /'PARITY ERROR*'/
0043
0044          call linchk (lun,1)
0045          write (lun,1)
0046     1    format (' ')
0047
0048          do 15,loop = 1,16
0049
0050          if (lib$extzv(15,1,memory_registers(loop)) .eq. 1) then
0051
0052          call linchk (lun,1)
0053
0054          write(lun,5) loop,memory_registers(loop)
0055     5    format(' ',T8,'CSR #',i<compress4 (loop)>,t24,z8.8)
0056
0057          call output (lun,memory_registers(loop),v1csr,0,0,0,'0')
0058          call output (lun,memory_registers(loop),v2csr,2,2,2,'0')
0059
```

000

```
0060              error_address1 = lib$extzv (5,7,memory_registers(loop))
0061              error_address1 = lib$insv (error_address1,11,7,error_address1)
0062
0063              error_address2 = lib$extzv (5,4,memory_registers(loop))
0064              error_address1 = lib$insv (error_address2,18,4,error_address1)
0065
0066              call linchk (lun,1)
0067
0068              write(lun,10) error_address1
0069      10      format(' ',t40,'PARITY ERROR ADDRESS, ',
0070             1 i<compress4 (error_address1)>,'.K')
0071
0072              call output (lun,memory_registers(loop),v3csr,14,14,15,'0')
0073              endif
0074
0075      15      continue
0076
0077              Return
0078              End
```

PROGRAM SECTIONS

Name	Bytes	Attributes
0 $CODE	383	PIC CON REL LCL SHR EXE RD NOWRT LONG
1 $PDATA	110	PIC CON REL LCL SHR NOEXE RD NOWRT LONG
2 $LOCAL	436	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
Total Space Allocated	929	

ENTRY POINTS

Address	Type	Name
0-00000000		MEMORY_REGISTER_UV1

VARIABLES

Address	Type	Name	Address	Type	Name
2-00000058	I*4	ERROR_ADDRESS1	2-0000005C	I*4	ERROR_ADDRESS2
2-00000060	I*4	LOOP	AP-00000004@	L*1	LUN

ARRAYS

Address	Type	Name	Bytes	Dimensions
AP-00000008@	I*4	MEMORY_REGISTERS	68	(0:16)
2-00000000	CHAR	V1CSR	20	(0:0)
2-00000014	CHAR	V2CSR	16	(2:2)
2-00000024	CHAR	V3CSR	50	(14:15)

LABELS

   Address   Label   Address   Label   Address   Label   Address   Label

   1-0000002A 1'   1-0000002E 5'   1-00000046 10'   **   15

FUNCTIONS AND SUBROUTINES REFERENCED

   Type   Name   Type   Name   Type   Name   Type   Name   Type   Name

   I*4   COMPRESS4   I*4   LIB$EXTZV   I*4   LIB$INSV   LINCHK   OUTPUT

0001

```
0002  C++
0003  C          Functional description:
0004  C
0005  C          This module maintains a list which is used to produce a display
0006  C          that shows where and how many memory errors have occured.  The format
0007  C          of the list is shown below.
0008  C
0009  C          +---------------------------------------+
0010  C          !               flink1                  !
0011  C          +---------------------------------------+
0012  C          !               blink1                  !
0013  C          +---------------------------------------+
0014  C          !             logging sid               !
0015  C          +---------------------------------------+
0016  C          !        root node memory flink         !
0017  C          +---------------------------------------+
0018  C          !        root memory node blink         !
0019  C          +---------------------------------------+
0020  C          !        memory node entry count        !
0021  C          +---------------------------------------+
0022  C
0023  C
0024  C          +---------------------------------------+
0025  C          !               flink2                  !
0026  C          +---------------------------------------+
0027  C          !               blink2                  !
0028  C          +---------------------------------------+
0029  C          !             memory node               !
0030  C          +---------------------------------------+
0031  C          !            root array flink           !
0032  C          +---------------------------------------+
0033  C          !            root array blink           !
0034  C          +---------------------------------------+
0035  C          !           array entry count           !
0036  C          +---------------------------------------+
0037  C
0038  C
0039  C          +---------------------------------------+
0040  C          !               flink3                  !
0041  C          +---------------------------------------+
0042  C          !               blink3                  !
0043  C          +---------------------------------------+
0044  C          !                array                  !
0045  C          +---------------------------------------+
0046  C          !         root array bank flink         !
0047  C          +---------------------------------------+
0048  C          !         root array bank blink         !
0049  C          +---------------------------------------+
0050  C          !        array bank entry count         !
0051  C          +---------------------------------------+
0052  C
0053  C
0054  C          +---------------------------------------+
0055  C          !               flink4                  !
0056  C          +---------------------------------------+
0057  C          !               blink4                  !
0058  C          +---------------------------------------+
```

```
0059    C               :          array bank           :
0060    C               +------------------------------+
0061    C               :     root array bit flink      :
0062    C               +------------------------------+
0063    C               :     root array bit blink      :
0064    C               +------------------------------+
0065    C               :     array page entry count    :
0066    C               +------------------------------+
0067    C
0068    C
0069    C               +------------------------------+
0070    C               :           flink5             :
0071    C               +------------------------------+
0072    C               :           blink5             :
0073    C               +------------------------------+
0074    C               :          array bit            :
0075    C               +------------------------------+
0076    C               :          error count         :
0077    C               +------------------------------+
0078    C--
0079
0080            Subroutine MEMORY_Q (search_sid,search_memory_node,
0081           1 search_array,search_array_bank,search_array_bit)
0082
0083            Implicit        none
0084
0085            byte            lun
0086
0087            integer*4       buffer0(2)
0088            integer*4       buffer1(6)
0089            integer*4       buffer2(6)
0090            integer*4       buffer3(6)
0091            integer*4       buffer4(6)
0092            integer*4       buffer5(4)
0093            integer*4       root_logging_sid_flink
0094            integer*4       root_logging_sid_blink
0095
0096            equivalence     (buffer0(1),root_logging_sid_flink)
0097            equivalence     (buffer0(2),root_logging_sid_blink)
0098
0099            integer*4       flink1
0100            integer*4       blink1
0101            integer*4       logging_sid
0102            integer*4       root_memory_node_flink
0103            integer*4       root_memory_node_blink
0104            integer*4       memory_node_entry_count
0105
0106            equivalence     (buffer1(1),flink1)
0107            equivalence     (buffer1(2),blink1)
0108            equivalence     (buffer1(3),logging_sid)
0109            equivalence     (buffer1(4),root_memory_node_flink)
0110            equivalence     (buffer1(5),root_memory_node_blink)
0111            equivalence     (buffer1(6),memory_node_entry_count)
0112
0113            integer*4       flink2
0114            integer*4       blink2
0115            integer*4       memory_node
```

```
0116              integer*4       root_array_flink
0117              integer*4       root_array_blink
0118              integer*4       array_entry_count
0119
0120              equivalence     (buffer2(1),flink2)
0121              equivalence     (buffer2(2),blink2)
0122              equivalence     (buffer2(3),memory_node)
0123              equivalence     (buffer2(4),root_array_flink)
0124              equivalence     (buffer2(5),root_array_blink)
0125              equivalence     (buffer2(6),array_entry_count)
0126
0127              integer*4       flink3
0128              integer*4       blink3
0129              integer*4       array
0130              integer*4       root_array_bank_flink
0131              integer*4       root_array_bank_blink
0132              integer*4       array_bank_entry_count
0133
0134              equivalence     (buffer3(1),flink3)
0135              equivalence     (buffer3(2),blink3)
0136              equivalence     (buffer3(3),array)
0137              equivalence     (buffer3(4),root_array_bank_flink)
0138              equivalence     (buffer3(5),root_array_bank_blink)
0139              equivalence     (buffer3(6),array_bank_entry_count)
0140
0141              integer*4       flink4
0142              integer*4       blink4
0143              integer*4       array_bank
0144              integer*4       root_array_bit_flink
0145              integer*4       root_array_bit_blink
0146              integer*4       array_bit_entry_count
0147
0148              equivalence     (buffer4(1),flink4)
0149              equivalence     (buffer4(2),blink4)
0150              equivalence     (buffer4(3),array_bank)
0151              equivalence     (buffer4(4),root_array_bit_flink)
0152              equivalence     (buffer4(5),root_array_bit_blink)
0153              equivalence     (buffer4(6),array_bit_entry_count)
0154
0155              integer*4       flink5
0156              integer*4       blink5
0157              integer*4       array_bit
0158              integer*4       error_count
0159
0160              equivalence     (buffer5(1),flink5)
0161              equivalence     (buffer5(2),blink5)
0162              equivalence     (buffer5(3),array_bit)
0163              equivalence     (buffer5(4),error_count)
0164
0165              integer*4       insert_blink
0166              integer*4       logging_sid_entry_count
0167              integer*4       logging_sid_entry_address
0168              integer*4       memory_node_entry_address
0169              integer*4       array_entry_address
0170              integer*4       array_bank_entry_address
0171              integer*4       array_bit_entry_address
0172              integer*4       search_sid
```

```
0173              integer*4      search_memory_node
0174              integer*4      search_array
0175              integer*4      search_array_bank
0176              integer*4      search_array_bit
0177              integer*4      compress4
0178              Integer*4      I
0179              Integer*4      J
0180              Integer*4      K
0181              Integer*4      L
0182              Integer*4      M
0183              Integer*4      lib$extzv
0184              Integer*4      Root_flink
0185              Integer*4      Sid_count
0186              Integer*4      Node_count
0187              Integer*4      Array_count
0188              Integer*4      Bank_count
0189              Integer*4      Bit_count
0190
0191              logical*1      lib$get_vm
0192
0193
0194              logging_sid_entry_address = root_logging_sid_flink
0195
0196              do 100,i = 1,logging_sid_entry_count
0197
0198              call movc3 (%val(24),%val(logging_sid_entry_address),buffer1)
0199
0200              if (search_sid .eq. logging_sid) then
0201
0202      5        memory_node_entry_address = root_memory_node_flink
0203
0204              do 90,j = 1,memory_node_entry_count
0205
0206              call movc3 (%val(24),%val(memory_node_entry_address),buffer2)
0207
0208              if (search_memory_node .eq. memory_node) then
0209
0210     10        array_entry_address = root_array_flink
0211
0212              do 80,k = 1,array_entry_count
0213
0214              call movc3 (%val(24),%val(array_entry_address),buffer3)
0215
0216              if (search_array .eq. array) then
0217
0218     15        array_bank_entry_address = root_array_bank_flink
0219
0220              do 70,l = 1,array_bank_entry_count
0221
0222              call movc3 (%val(24),%val(array_bank_entry_address),buffer4)
0223
0224              if (search_array_bank .eq. array_bank) then
0225
0226     20        array_bit_entry_address = root_array_bit_flink
0227
0228              do 60,m = 1,array_bit_entry_count
0229
```

```
0230                call movc3 (%val(16),%val(array_bit_entry_address),buffer5)
0231
0232                if (search_array_bit .eq. array_bit) then
0233
0234        25      error_count = error_count + 1
0235
0236                call movl (error_count,%val(array_bit_entry_address + 12))
0237
0238                return
0239                endif
0240
0241                array_bit_entry_address = flink5
0242
0243        60      continue
0244
0245                call movc5 (%val(0),,%val(0),%val(16),buffer5)
0246
0247                if (lib$get_vm(((16+7)/8)*8,array_bit_entry_address)) then
0248
0249                call insque (%val(array_bit_entry_address),
0250               1 %val(root_array_bit_blink))
0251
0252                array_bit = search_array_bit
0253
0254                call movq (array_bit,%val(array_bit_entry_address + 8))
0255
0256                array_bit_entry_count = array_bit_entry_count + 1
0257
0258                call movl (array_bit_entry_count,%val(array_bank_entry_address + 20))
0259
0260                goto 25
0261                endif
0262
0263                return
0264                endif
0265
0266                insert_blink = blink4
0267
0268                if (array_bank .gt. search_array_bank) goto 75
0269
0270                array_bank_entry_address = flink4
0271
0272        70      continue
0273
0274                insert_blink = root_array_bank_blink
0275
0276        75      call movc5 (%val(0),,%val(0),%val(24),buffer4)
0277
0278                if (lib$get_vm(((24+7)/8)*8,array_bank_entry_address)) then
0279
0280                call insque (%val(array_bank_entry_address),%val(insert_blink))
0281
0282                array_bank = search_array_bank
0283
0284                root_array_bit_flink = array_bank_entry_address + 12
0285
0286                root_array_bit_blink = root_array_bit_flink
```

```
0287
0288            call movc3 (%val(16),array_bank,%val(array_bank_entry_address + 8))
0289
0290            array_bank_entry_count = array_bank_entry_count + 1
0291
0292            call movl (array_bank_entry_count,%val(array_entry_address + 20))
0293
0294            goto 20
0295            endif
0296
0297            return
0298            endif
0299
0300            insert_blink = blink3
0301
0302            if (array .gt. search_array) goto 85
0303
0304            array_entry_address = flink3
0305
0306    80      continue
0307
0308            insert_blink = root_array_blink
0309
0310    85      call movc5 (%val(0),,%val(0),%val(24),buffer3)
0311
0312            if (lib$get_vm(((24+7)/8)*8,array_entry_address)) then
0313
0314            call insque (%val(array_entry_address),%val(insert_blink))
0315
0316            array = search_array
0317
0318            root_array_bank_flink = array_entry_address + 12
0319
0320            root_array_bank_blink = root_array_bank_flink
0321
0322            call movc3 (%val(16),array,%val(array_entry_address + 8))
0323
0324            array_entry_count = array_entry_count + 1
0325
0326            call movl (array_entry_count,%val(memory_node_entry_address + 20))
0327
0328            goto 15
0329            endif
0330
0331            return
0332            endif
0333
0334            insert_blink = blink2
0335
0336            if (memory_node .gt. search_memory_node) goto 95
0337
0338            memory_node_entry_address = flink2
0339
0340    90      continue
0341
0342            insert_blink = root_memory_node_blink
0343
```

```
0344    95      call movc5 (%val(0),,%val(0),%val(24),buffer2)
0345
0346            if (lib$get_vm(((24+7)/8)*8,memory_node_entry_address)) then
0347
0348            call insque (%val(memory_node_entry_address),%val(insert_blink))
0349
0350            memory_node = search_memory_node
0351
0352            root_array_flink = memory_node_entry_address + 12
0353
0354            root_array_blink = root_array_flink
0355
0356            call movc3 (%val(16),memory_node,%val(memory_node_entry_address + 8))
0357
0358            memory_node_entry_count = memory_node_entry_count + 1
0359
0360            call movl (memory_node_entry_count,
0361          1 %val(logging_sid_entry_address + 20))
0362
0363            goto 10
0364            endif
0365
0366            return
0367            endif
0368
0369            logging_sid_entry_address = flink1
0370
0371    100     continue
0372
0373            call movc5 (%val(0),,%val(0),%val(24),buffer1)
0374
0375            if (logging_sid_entry_count .eq. 0) then
0376
0377            root_logging_sid_flink = %loc(root_logging_sid_flink)
0378
0379            root_logging_sid_blink = root_logging_sid_flink
0380            endif
0381
0382            if (lib$get_vm(((24+7)/8)*8,logging_sid_entry_address)) then
0383
0384            call insque (%val(logging_sid_entry_address),
0385          1 %val(root_logging_sid_blink))
0386
0387            logging_sid = search_sid
0388
0389            root_memory_node_flink = logging_sid_entry_address + 12
0390
0391            root_memory_node_blink = root_memory_node_flink
0392
0393            call movc3 (%val(16),logging_sid,%val(logging_sid_entry_address + 8))
0394
0395            logging_sid_entry_count = logging_sid_entry_count + 1
0396
0397            goto 5
0398            endif
0399
0400            return
```

```
0401
0402        C
0403        C This routine returns the root flink and the number of entries for the
0404        C memory information queue.
0405        C
0406              Entry GET_MEMORY_Q_INFO (root_flink,sid_count,node_count,array_count,
0407             1 bank_count,bit_count)
0408
0409              Root_flink = root_logging_sid_flink
0410              Sid_count = logging_sid_entry_count
0411              Node_count = memory_node_entry_count
0412              Array_count = array_entry_count
0413              Bank_count = array_bank_entry_count
0414              Bit_count = array_bit_entry_count
0415
0416              Return
0417
0418              End
```

PROGRAM SECTIONS

	Name	Bytes	Attributes
0	$CODE	925	PIC CON REL LCL SHR EXE RD NOWRT LONG
1	$PDATA	8	PIC CON REL LCL SHR NOEXE RD NOWRT LONG
2	$LOCAL	608	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
	Total Space Allocated	1541	

ENTRY POINTS

Address	Type	Name	Address	Type	Name
0-00000375		GET_MEMORY_Q_INFO	0-00000000		MEMORY_Q

VARIABLES

Address	Type	Name	Address	Type	Name
2-00000030	I*4	ARRAY	2-00000018	I*4	ARRAY_BANK
2-00000090	I*4	ARRAY_BANK_ENTRY_ADDRESS	2-0000003C	I*4	ARRAY_BANK_ENTRY_COUNT
2-00000008	I*4	ARRAY_BIT	2-00000094	I*4	ARRAY_BIT_ENTRY_ADDRESS
2-00000024	I*4	ARRAY_BIT_ENTRY_COUNT	AP-0000010a	I*4	ARRAY_COUNT
2-0000008C	I*4	ARRAY_ENTRY_ADDRESS	2-00000054	I*4	ARRAY_ENTRY_COUNT
AP-0000014a	I*4	BANK_COUNT	AP-0000018a	I*4	BIT_COUNT
2-0000005C	I*4	BLINK1	2-00000044	I*4	BLINK2
2-0000002C	I*4	BLINK3	2-00000014	I*4	BLINK4
2-00000004	I*4	BLINK5	2-00000098	I*4	COMPRESS4
2-0000000C	I*4	ERROR_COUNT	2-00000058	I*4	FLINK1
2-00000040	I*4	FLINK2	2-00000028	I*4	FLINK3
2-00000010	I*4	FLINK4	2-00000000	I*4	FLINK5
2-0000009C	I*4	I	2-0000007C	I*4	INSERT_BLINK

```
2-000000A0  I*4  J                            2-000000A4  I*4  K
2-000000A8  I*4  L                            2-000000BC  I*4  LIB$EXTZV
2-00000060  I*4  LOGGING_SID                  2-00000084  I*4  LOGGING_SID_ENTRY_ADDRESS
2-00000080  I*4  LOGGING_SID_ENTRY_COUNT      2-00000078  L*1  LUN
2-000000AC  I*4  M                            2-00000048  I*4  MEMORY_NODE
2-00000088  I*4  MEMORY_NODE_ENTRY_ADDRESS    2-0000006C  I*4  MEMORY_NODE_ENTRY_COUNT
AP-0000000C@ I*4  NODE_COUNT                   2-00000038  I*4  ROOT_ARRAY_BANK_BLINK
2-00000034  I*4  ROOT_ARRAY_BANK_FLINK        2-00000020  I*4  ROOT_ARRAY_BIT_BLINK
2-0000001C  I*4  ROOT_ARRAY_BIT_FLINK         2-00000050  I*4  ROOT_ARRAY_BLINK
2-0000004C  I*4  ROOT_ARRAY_FLINK            AP-0000004@ I*4  ROOT_FLINK
2-00000074  I*4  ROOT_LOGGING_SID_BLINK       2-00000070  I*4  ROOT_LOGGING_SID_FLINK
2-00000068  I*4  ROOT_MEMORY_NODE_BLINK       2-00000064  I*4  ROOT_MEMORY_NODE_FLINK
AP-0000000C@ I*4  SEARCH_ARRAY                AP-000000010@ I*4  SEARCH_ARRAY_BANK
AP-000000014@ I*4  SEARCH_ARRAY_BIT           AP-000000008@ I*4  SEARCH_MEMORY_NODE
AP-000000004@ I*4  SEARCH_SID                 AP-000000008@ I*4  SID_COUNT
```

ARRAYS

```
    Address  Type  Name              Bytes  Dimensions

    2-00000070  I*4  BUFFER0            8    (2)
    2-00000058  I*4  BUFFER1           24    (6)
    2-00000040  I*4  BUFFER2           24    (6)
    2-00000028  I*4  BUFFER3           24    (6)
    2-00000010  I*4  BUFFER4           24    (6)
    2-00000000  I*4  BUFFER5           16    (4)
```

LABELS

Address	Label	Address	Label	Address	Label	Address	Label	Address	Label	Address	Label
0-C000003A	5	C-0000006B	10	0-0000009C	15	0-000000CD	20	0-000000F9	25	**	60
**	70	0-00000194	75	**	80	0-00000218	85	**	90	0-0000029C	95
**	100										

FUNCTIONS AND SUBROUTINES REFERENCED

Type	Name	Type	Name	Type	Name	Type	Name	Type	Name	Type	Name
	INSQUE	L*1	LIB$GET_VM		MOVC3		MOVC5		MOVL		MOVQ

0001

```
0002    C++
0003    C           Functional description:
0004    C
0005    C           This entry point is used to display the memory error occurrance list
0006    C           built by memory_q.
0007    C--
0008
0009                Subroutine MEMORY_DISPLAY (lun)
0010
0011                Implicit        none
0012
0013                integer*4       buffer0(2)
0014                integer*4       buffer1(6)
0015                integer*4       buffer2(6)
0016                integer*4       buffer3(6)
0017                integer*4       buffer4(6)
0018                integer*4       buffer5(4)
0019                integer*4       root_logging_sid_flink
0020                integer*4       root_logging_sid_blink
0021
0022                equivalence     (buffer0(1),root_logging_sid_flink)
0023                equivalence     (buffer0(2),root_logging_sid_blink)
0024
0025                integer*4       flink1
0026                integer*4       blink1
0027                integer*4       logging_sid
0028                integer*4       root_memory_node_flink
0029                integer*4       root_memory_node_blink
0030                integer*4       memory_node_entry_count
0031
0032                equivalence     (buffer1(1),flink1)
0033                equivalence     (buffer1(2),blink1)
0034                equivalence     (buffer1(3),logging_sid)
0035                equivalence     (buffer1(4),root_memory_node_flink)
0036                equivalence     (buffer1(5),root_memory_node_blink)
0037                equivalence     (buffer1(6),memory_node_entry_count)
0038
0039                integer*4       flink2
0040                integer*4       blink2
0041                integer*4       memory_node
0042                integer*4       root_array_flink
0043                integer*4       root_array_blink
0044                integer*4       array_entry_count
0045
0046                equivalence     (buffer2(1),flink2)
0047                equivalence     (buffer2(2),blink2)
0048                equivalence     (buffer2(3),memory_node)
0049                equivalence     (buffer2(4),root_array_flink)
0050                equivalence     (buffer2(5),root_array_blink)
0051                equivalence     (buffer2(6),array_entry_count)
0052
0053                integer*4       flink3
0054                integer*4       blink3
0055                integer*4       array
0056                integer*4       root_array_bank_flink
0057                integer*4       root_array_bank_blink
0058                integer*4       array_bank_entry_count
```

```
0059
0060          equivalence      (buffer3(1),flink3)
0061          equivalence      (buffer3(2),blink3)
0062          equivalence      (buffer3(3),array)
0063          equivalence      (buffer3(4),root_array_bank_flink)
0064          equivalence      (buffer3(5),root_array_bank_blink)
0065          equivalence      (buffer3(6),array_bank_entry_count)
0066
0067          integer*4        flink4
0068          integer*4        blink4
0069          integer*4        array_bank
0070          integer*4        root_array_bit_flink
0071          integer*4        root_array_bit_blink
0072          integer*4        array_bit_entry_count
0073
0074          equivalence      (buffer4(1),flink4)
0075          equivalence      (buffer4(2),blink4)
0076          equivalence      (buffer4(3),array_bank)
0077          equivalence      (buffer4(4),root_array_bit_flink)
0078          equivalence      (buffer4(5),root_array_bit_blink)
0079          equivalence      (buffer4(6),array_bit_entry_count)
0080
0081          integer*4        flink5
0082          integer*4        blink5
0083          integer*4        array_bit
0084          integer*4        error_count
0085
0086          equivalence      (buffer5(1),flink5)
0087          equivalence      (buffer5(2),blink5)
0088          equivalence      (buffer5(3),array_bit)
0089          equivalence      (buffer5(4),error_count)
0090
0091          integer*4        insert_blink
0092          integer*4        logging_sid_entry_count
0093          integer*4        logging_sid_entry_address
0094          integer*4        memory_node_entry_address
0095          integer*4        array_entry_address
0096          integer*4        array_bank_entry_address
0097          integer*4        array_bit_entry_address
0098          integer*4        search_sid
0099          integer*4        search_memory_node
0100          integer*4        search_array
0101          integer*4        search_array_bank
0102          integer*4        search_array_bit
0103          Integer*4        I
0104          Integer*4        J
0105          Integer*4        K
0106          Integer*4        L
0107          Integer*4        M
0108          Integer*4        lib$extzv
0109          integer*4        compress4
0110
0111          byte             lun
0112
0113    C
0114    C Get the root flink and neccessary entry counts fort the memory information
0115    C queue.
```

```
0116    C
0117            Call GET_MEMORY_Q_INFO (root_logging_sid_flink,logging_sid_entry_count,
0118          1 memory_node_entry_count,array_entry_count,array_bank_entry_count,
0119          2 array_bit_entry_count)
0120
0121            logging_sid_entry_address = root_logging_sid_flink
0122
0123            do 200,i = 1,logging_sid_entry_count
0124
0125            call movc3 (%val(24),%val(logging_sid_entry_address),buffer1)
0126
0127            call frctof (lun)
0128
0129            call linchk (lun,2)
0130
0131            write(lun,110) logging_sid
0132    110     format(/' ','SUMMARY OF MEMORY ERRORS LOGGED BY SID ',z8.8)
0133
0134            memory_node_entry_address = root_memory_node_flink
0135
0136            do 190,j = 1,memory_node_entry_count
0137
0138            call movc3 (%val(24),%val(memory_node_entry_address),buffer2)
0139
0140            call linchk (lun,5)
0141
0142            if (lib$extzv(24,8,logging_sid) .eq. 1) then
0143
0144            write(lun,115) 'TR #',memory_node
0145    115     format(/' ','CONTROLLER AT ',a,i<compress4 (memory_node)>,'.')
0146
0147            else if (lib$extzv(24,8,logging_sid) .eq. 2) then
0148
0149            write(lun,115) 'SLOT INDEX #',memory_node
0150            endif
0151
0152            write(lun,117) 'ARRAY#','BIT#','BANK','CORRECTED','FATAL',
0153          1 'ERRORS','ERRORS'
0154    117     format(/' ',t8,a,t16,a,t24,a,t35,a,t50,a,/,
0155          1 t37,a,t50,a)
0156
0157            array_entry_address = root_array_flink
0158
0159            do 180,k = 1,array_entry_count
0160
0161            call movc3 (%val(24),%val(array_entry_address),buffer3)
0162
0163            array_bank_entry_address = root_array_bank_flink
0164
0165            do 170,l = 1,array_bank_entry_count
0166
0167            call movc3 (%val(24),%val(array_bank_entry_address),buffer4)
0168
0169            array_bit_entry_address = root_array_bit_flink
0170
0171            do 160,m = 1,array_bit_entry_count
0172
```

033
033
033
033
033
033
033
033
033
034
034
034
034
034
034
034
034
035
035
035
035
035
035
035
035
035
035
036
036
036
036
036
036
036
036
036
037
037
037
037
037
037
037
037
038
038
038
038
038
038

```
0173            call movc3 (%val(16),%val(array_bit_entry_address),buffer5)
0174
0175            call linchk (lun,2)
0176
0177            if (
0178          1 array .ne. -1
0179          1 .and.
0180          1 array_bank .ne. -1
0181          1 .and.
0182          1 array_bit .ne. -1
0183          1 ) then
0184
0185            write(lun,120) array,array_bit,array_bank,error_count
0186    120     format(/' ',t10,i2.2,'.',t17,i2.2,'.',t25,i2.2,'.',t31,i10.1,'.')
0187
0188            else if (
0189          1 array .ne. -1
0190          1 .and.
0191          1 array_bank .ne. -1
0192          1 .and.
0193          1 array_bit .eq. -1
0194          1 ) then
0195
0196            write(lun,125) array,array_bank,error_count
0197    125     format(/' ',t10,i2.2,'.',t25,i2.2,'.',t44,i10.1,'.')
0198
0199            else if (
0200          1 array .NE. -1
0201          1 .AND.
0202          1 array_bank .EQ. -1
0203          1 .AND.
0204          1 array_bit .EQ. -1
0205          1 ) then
0206
0207            Write (lun,127) array,error_count
0208    127     Format (' ',T10,I2.2,'.',T17,'N/A',T25,'N/A',T44,I10.1,'.')
0209            Else
0210
0211            write(lun,130) error_count
0212    130     format(/' ',t44,i10.T,'.')
0213            endif
0214
0215    155     array_bit_entry_address = flink5
0216
0217    160     continue
0218
0219    165     array_bank_entry_address = flink4
0220
0221    170     continue
0222
0223    175     array_entry_address = flink3
0224
0225    180     continue
0226
0227    185     memory_node_entry_address = flink2
0228
0229    190     continue
```

```
0230
0231      195      logging_sid_entry_address = flink1
0232
0233      200      continue
0234
0235               return
0236
0237               end
```

PROGRAM SECTIONS

Name	Bytes	Attributes
0 $CODE	916	PIC CON REL LCL SHR EXE RD NOWRT LONG
1 $PDATA	286	PIC CON REL LCL SHR NOEXE RD NOWRT LONG
2 $LOCAL	424	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
Total Space Allocated	1626	

ENTRY POINTS

Address	Type	Name
0-00000000		MEMORY_DISPLAY

VARIABLES

Address	Type	Name	Address	Type	Name
2-00000030	I*4	ARRAY	2-00000018	I*4	ARRAY_BANK
2-0000008C	I*4	ARRAY_BANK_ENTRY_ADDRESS	2-0000003C	I*4	ARRAY_BANK_ENTRY_COUNT
2-00000008	I*4	ARRAY_BIT	2-00000090	I*4	ARRAY_BIT_ENTRY_ADDRESS
2-00000024	I*4	ARRAY_BIT_ENTRY_COUNT	2-00000088	I*4	ARRAY_ENTRY_ADDRESS
2-00000054	I*4	ARRAY_ENTRY_COUNT	2-0000005C	I*4	BLINK1
2-00000044	I*4	BLINK2	2-0000002C	I*4	BLINK3
2-00000014	I*4	BLINK4	2-00000004	I*4	BLINK5
2-0000000C	I*4	ERROR_COUNT	2-00000058	I*4	FLINK1
2-00000040	I*4	FLINK2	2-00000028	I*4	FLINK3
2-00000010	I*4	FLINK4	2-00000000	I*4	FLINK5
2-000000A8	I*4	I	2-00000078	I*4	INSERT_BLINK
2-000000AC	I*4	J	2-000000B0	I*4	K
2-000000B4	I*4	L	2-00000060	I*4	LOGGING_SID
2-00000080	I*4	LOGGING_SID_ENTRY_ADDRESS	2-0000007C	I*4	LOGGING_SID_ENTRY_COUNT
AP-00000048	L*1	LUN	2-000000B8	I*4	M
2-00000048	I*4	MEMORY_NODE	2-00000084	I*4	MEMORY_NODE_ENTRY_ADDRESS
2-0000006C	I*4	MEMORY_NODE_ENTRY_COUNT	2-00000038	I*4	ROOT_ARRAY_BANK_BLINK
2-00000034	I*4	ROOT_ARRAY_BANK_FLINK	2-00000020	I*4	ROOT_ARRAY_BIT_BLINK
2-0000001C	I*4	ROOT_ARRAY_BIT_FLINK	2-00000050	I*4	ROOT_ARRAY_BLINK
2-0000004C	I*4	ROOT_ARRAY_FLINK	2-00000074	I*4	ROOT_LOGGING_SID_BLINK
2-00000070	I*4	ROOT_LOGGING_SID_FLINK	2-00000068	I*4	ROOT_MEMORY_NODE_BLINK
2-00000064	I*4	ROOT_MEMORY_NODE_FLINK	2-0000009C	I*4	SEARCH_ARRAY
2-000000A0	I*4	SEARCH_ARRAY_BANK	2-000000A4	I*4	SEARCH_ARRAY_BIT

044
044
044
044
044
045
045
045
045
045
045
045
045
046
046
046
046
046
046
046
046
047
047
047
047
047
047
047
048
048
048
048
048
048
048
048
048
049
049
049
049
049
049
049
049
049
050

2-00000098 I*4 SEARCH_MEMORY_NODE   2-00000094 I*4 SEARCH_SID

ARRAYS

   Address   Type Name   Bytes Dimensions

2-00000070 I*4 BUFFER0   8 (2)
2-00000058 I*4 BUFFER1   24 (6)
2-00000040 I*4 BUFFER2   24 (6)
2-00000028 I*4 BUFFER3   24 (6)
2-00000010 I*4 BUFFER4   24 (6)
2-00000000 I*4 BUFFER5   16 (4)

LABELS

   Address   Label   Address   Label   Address   Label   Address   Label   Address   Label   Address   Label

1-00000042 110'   1-00000073 115'   1-00000092 117'   1-000000AD 120'   1-000000D2 125'   1-000000EF 127'
1-00000111 130'   **   155   **   160   **   165   **   170   **   175
   **   180   **   185   **   190   **   195   **   200

FUNCTIONS AND SUBROUTINES REFERENCED

   Type Name   Type Name   Type Name

   I*4 COMPRESS4   FRCTOF   GET_MEMORY_Q_INFO
   I*4 LIB$EXTZV   LINCHK   MOVC3

COMMAND QUALIFIERS

   FORTRAN /LIS=LIS$:MEMORYS/OBJ=OBJ$:MEMORYS MSRC$:MEMORYS

   /CHECK=(NOBOUNDS,OVERFLOW,NOUNDERFLOW)
   /DEBUG=(NOSYMBOLS,TRACEBACK)
   /STANDARD=(NOSYNTAX,NOSOURCE_FORM)
   /SHOW=(NOPREPROCESSOR,NOINCLUDE,MAP)
   /F77 /NOG_FLOATING /I4 /OPTIMIZE /WARNINGS /NOD_LINES /NOCROSS_REFERENCE /NOMACHINE_CODE /CONTINUATIONS=19

COMPILATION STATISTICS

   Run Time: 34.31 seconds
   Elapsed Time: 68.12 seconds
   Page Faults: 470
   Dynamic Memory: 236 pages

MESSAGE
LIS

ML11.
LIS

MSCP
LIS

MFTAPE
LIS

MOUNT
LIS

MEMORYS
LIS

MOUXX
LIS

MCHK_DISP
LIS