K 6

```
EEEEEEEEEEEEEEE  RRRRRRRRRRRR     FFFFFFFFFFFFFFF
EEEEEEEEEEEEEEE  RRRRRRRRRRRR     FFFFFFFFFFFFFFF
EEEEEEEEEEEEEEE  RRRRRRRRRRRR     FFFFFFFFFFFFFFF
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
EEEEEEEEEEEEEEE  RRR         RRR  FFF
EEEEEEEEEEEEEEE  RRR         RRR  FFF
EEEEEEEEEEEEEEE  RRR         RRR  FFF
```

```
MM      MM  BBBBBBBB      AAAAAA
MM      MM  BBBBBBBB      AAAAAA
MMMM  MMMM  BB      BB  AA      AA
MMMM  MMMM  BB      BB  AA      AA
MM  MM  MM  BB      BB  AA      AA
MM  MM  MM  BB      BB  AA      AA
MM      MM  BBBBBBBB    AA      AA
MM      MM  BBBBBBBB    AA      AA
MM      MM  BB      BB  AAAAAAAAAA
MM      MM  BB      BB  AAAAAAAAAA
MM      MM  BB      BB  AA      AA      ....
MM      MM  BB      BB  AA      AA      ....
MM      MM  BBBBBBBB    AA      AA      ....
MM      MM  BBBBBBBB    AA      AA      ....

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
```

RH

PR

EN

VA

AR

A

LA

FU

```
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
0028    c       Author  Brian Porter                    Creation Date   11-NOV-1981
0029
0030
0031    c++
0032    c       Functional description:
0033    c
0034    c       This is a collection of routines that display the VAX-11 family
0035    c       of MBA adapter registers.  It is called by various massbus device
0036    c       modules.
0037    c
0038    c       Modified by:
0039    C
0040    C       V03-002 SAR0146         Sharon A. Reynolds,     5-Oct-1983
0041    C               Added an SYE update that makes the register heralds
0042    C               generic for 11/785 and 11/7XX support.
0043    C
0044    C       V03-001 SAR0080         Sharon A. Reynolds,     20-Jun-1983
0045    C               Changed the carriage control in the 'format' statements
0046    C               for use with ERF.
0047    C
0048    c--
0049    c--
0050
0051
0052
0053            subroutine mba_control_registers (lun,number_of_registers,
0054            1 adapter_registers,selected_mapping_register)
0055
0056
0057
```

```
0058            include 'src$:msghdr.for /nolist'
0117
0118
0119
0120            byte            lun
0121
0122            integer*4       number_of_registers
0123
0124            integer*4       adapter_registers(number_of_registers)
0125
0126            integer*4       selected_mapping_register
0127
0128            integer*4       compress4
0129
0130
0131
0132
0133            if (
0134            1 lib$extzv(24,8,emb$l_hd_sid) .eq. 255
0135            1 .or.
0136            1 lib$extzv(24,8,emb$l_hd_sid) .eq. 1
0137            1 ) then
0138
0139            call rh780_control_registers (lun,adapter_registers,
0140            1 selected_mapping_register)
0141
0142            else if (lib$extzv(24,8,emb$l_hd_sid) .eq. 2) then
0143
0144            call rh750_control_registers (lun,adapter_registers,
0145            1 selected_mapping_register)
0146
0147    c
0148    c       for future MBA support the ELSE-IF-THEN should be expanded
0149    c       at this point.
0150    c
0151
0152            else
0153
0154            call linchk (lun,(number_of_registers + 1))
0155
0156            do 20,i = 1,number_of_registers
0157
0158            write(lun,10) ''RH'' REGISTER #',i,adapter_registers(i)
0159    10      format(' ',t8,a,i<compress4 (i)>,t24,z8.8)
0160
0161    20      continue
0162
0163            selected_mapping_register = -1
0164            endif
0165
0166            return
0167
0168            end
```

PROGRAM SECTIONS

	Name	Bytes	Attributes
0	$CODE	244	PIC CON REL LCL SHR EXE RD NOWRT LONG
1	$PDATA	41	PIC CON REL LCL SHR NOEXE RD NOWRT LONG
2	$LOCAL	104	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
3	EMB	512	PIC OVR REL GBL SHR NOEXE RD WRT LONG
	Total Space Allocated	901	

ENTRY POINTS

Address	Type	Name
0-00000000		MBA_CONTROL_REGISTERS

VARIABLES

Address	Type	Name	Address	Type	Name
3-00000000	I*4	EMB$L_HD_SID	3-00000004	I*2	EMB$W_HD_ENTRY
3-0000000E	I*2	EMB$W_HD_ERRSEQ	2-00000000	I*4	I
AP-00000004@	L*1	LUN	AP-00000008@	I*4	NUMBER_OF_REGISTERS
AP-00000010@	I*4	SELECTED_MAPPING_REGISTER			

ARRAYS

Address	Type	Name	Bytes	Dimensions
AP-0000000C@	I*4	ADAPTER_REGISTERS	**	(*)
3-00000000	L*1	EMB	512	(0:511)
3-00000006	I*4	EMB$Q_HD_TIME	8	(2)

LABELS

Address	Label	Address	Label
1-00000017	10'	**	20

FUNCTIONS AND SUBROUTINES REFERENCED

Type	Name	Type	Name	Type	Name
I*4	COMPRESS4	I*4	LIB$EXTZV		LINCHK
	RH750_CONTROL_REGISTERS		RH780_CONTROL_REGISTERS		

```
0001
0002
0003
0004            subroutine rh780_control_registers (lun,adapter_registers,
0005            1 selected_mapping_register)
0006
0007
0008    c++
0009    c       This routine displays the RH780 adapter registers.  It expects
0010    c       the registers in the following order.
0011    c
0012    c       +-------------------------------+
0013    c       |    configuration register     |
0014    c       +-------------------------------+
0015    c       |       control register        |
0016    c       +-------------------------------+
0017    c       |        status register        |
0018    c       +-------------------------------+
0019    c       |   virtual address register    |
0020    c       +-------------------------------+
0021    c       |      byte count register      |
0022    c       +-------------------------------+
0023    c--
0024
0025
0026
0027
0028            byte            lun
0029
0030            integer*4       adapter_registers(5)
0031
0032            integer*4       selected_mapping_register
0033
0034            integer*4       compress4
0035
0036            logical*1       diagnostic_mode
0037
0038            integer*4       byte_offset
0039
0040            integer*4       selected_map_register
0041
0042            integer*4       sbi_byte_count
0043
0044            integer*4       massbus_byte_count
0045
0046            character*17    virh780_control_register(0:2)
0047
0048            data    virh780_control_register(0)     /'INITIALIZATION*'/
0049
0050            data    virh780_control_register(1)     /'ABORT*'/
0051
0052            data    virh780_control_register(2)     /'INTERRUPT ENABLE*'/
0053
0054            character*28    virh780_status_register(0:13)
0055
0056            data    virh780_status_register(0)  /'READ DATA TIMEOUT*'/
0057
```

```
0058            data    v1rh780_status_register(1)  /'INTERFACE SEQUENCE TIMEOUT*'/
0059
0060            data    v1rh780_status_register(2)  /'READ DATA SUBSTITUTE*'/
0061
0062            data    v1rh780_status_register(3)  /'ERROR CONFIRMATION*'/
0063
0064            data    v1rh780_status_register(4)  /'INVALID MAP*'/
0065
0066            data    v1rh780_status_register(5)  /'PAGE FRAME MAP PARITY ERROR*'/
0067
0068            data    v1rh780_status_register(6)  /''MASSBUS'' DATA PARITY ERROR*'/
0069
0070            data    v1rh780_status_register(7) /''MASSBUS'' EXCEPTION*'/
0071
0072            data    v1rh780_status_register(8)  /'MISS TRANSFER ERROR*'/
0073
0074            data    v1rh780_status_register(9)  /'WRITE CHECK LOWER ERROR*'/
0075
0076            data    v1rh780_status_register(10) /'WRITE CHECK UPPER ERROR*'/
0077
0078            data    v1rh780_status_register(11) /'DATA LATE*'/
0079
0080            data    v1rh780_status_register(12) /'DATA TRANSFER ABORTED*'/
0081
0082            data    v1rh780_status_register(13) /'DATA TRANSFER COMPLETED*'/
0083
0084
0085
0086            diagnostic_mode = .false.
0087
0088            if (lib$extzv(3,1,adapter_registers(2)) .eq. 1)
0089            1 diagnostic_mode = .true.
0090
0091            if (.not. diagnostic_mode) then
0092
0093            call rh780_configuration_register (lun,adapter_registers(1))
0094            else
0095
0096            call linchk (lun,2)
0097
0098            write(lun,10) adapter_registers(1)
0099    10      format(/' ',t8,''RH'' CSR',t24,z8.8)
0100            endif
0101
0102            call linchk (lun,1)
0103
0104            write(lun,15) adapter_registers(2)
0105    15      format(' ',t8,''RH'' CR',t24,z8.8)
0106
0107            if (.not. diagnostic_mode) then
0108
0109            call output (lun,adapter_registers(2),v1rh780_control_register,
0110            1 0,0,2,'0')
0111            else
0112
0113            call linchk (lun,1)
0114
```

```
0115            write(lun,20) 'DIAGNOSTIC MODE'
0116    20      format(' ',t40,a)
0117            endif
0118
0119            call linchk (lun,1)
0120
0121            write(lun,25) adapter_registers(3)
0122    25      format(' ',t8,''RH'' SR',t24,z8.8)
0123
0124            if (.not. diagnostic_mode) then
0125
0126            call output (lun,adapter_registers(3),v1rh780_status_register,
0127            1 0,0,13,'0')
0128
0129            call rh780_status_register16_31 (lun,adapter_registers(3))
0130            endif
0131
0132            call linchk (lun,1)
0133
0134            write(lun,30) adapter_registers(4)
0135    30      format(' ',t8,''RH'' VAR',t24,z8.8)
0136
0137            if (.not. diagnostic_mode) then
0138
0139            byte_offset = lib$extzv(0,9,adapter_registers(4))
0140
0141            selected_map_register = lib$extzv(9,8,adapter_registers(4))
0142
0143            call linchk (lun,2)
0144
0145            if (byte_offset .eq. 0) then
0146
0147            write(lun,20) 'PAGE ALIGNED'
0148            else
0149
0150            write(lun,35) byte_offset
0151    35      format(' ',t40,i<compress4 (byte_offset)>,'. BYTE, PAGE OFFSET')
0152            endif
0153
0154            write(lun,40) selected_map_register
0155    40      format(' ',t40,'MAPPING REGISTER #',
0156            1 i<compress4 (selected_map_register)>,'. SELECTED')
0157            endif
0158
0159            call linchk (lun,1)
0160
0161            write(lun,45) adapter_registers(5)
0162    45      format(' ',t8,''RH'' BCR',t24,z8.8)
0163
0164            if (.not. diagnostic_mode) then
0165
0166            sbi_byte_count = lib$extv(0,16,adapter_registers(5))
0167
0168            sbi_byte_count = max(0,sbi_byte_count) - min(0,sbi_byte_count)
0169
0170            if (sbi_byte_count .ne. 0) then
0171
```

```
0172            call linchk (lun,1)
0173
0174            write(lun,50) sbi_byte_count
0175    50      format(' ',t40,''SBI'' BYTE COUNT, ',i<compress4 (sbi_byte_count)>,
0176            1 '.')
0177            endif
0178
0179            massbus_byte_count = lib$extv(16,16,adapter_registers(5))
0180
0181            massbus_byte_count = max(0,massbus_byte_count) -
0182            1 min(0,massbus_byte_count)
0183
0184            if (massbus_byte_count .ne. 0) then
0185
0186            call linchk (lun,1)
0187
0188            write(lun,55) massbus_byte_count
0189    55      format(' ',t40,''MASSBUS'' BYTE COUNT, '
0190            1 i<compress4 (massbus_byte_count)>,'.')
0191            endif
0192            endif
0193
0194            selected_mapping_register = selected_map_register
0195
0196            return
0197
0198            end
```

PROGRAM SECTIONS

	Name	Bytes	Attributes
0	$CODE	844	PIC CON REL LCL SHR EXE RD NOWRT LONG
1	$PDATA	323	PIC CON REL LCL SHR NOEXE RD NOWRT LONG
2	$LOCAL	780	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
	Total Space Allocated	1947	

ENTRY POINTS

Address	Type	Name
0-00000000		RH780_CONTROL_REGISTERS

VARIABLES

Address	Type	Name	Address	Type	Name
2-000001BC	I*4	BYTE_OFFSET	2-000001BB	L*1	DIAGNOSTIC_MODE
AP-00000004@	L*1	LUN	2-000001C8	I*4	MASSBUS_BYTE_COUNT
2-000001C4	I*4	SBI_BYTE_COUNT	AP-0000000C@	I*4	SELECTED_MAPPING_REGISTER
2-000001C0	I*4	SELECTED_MAP_REGISTER			

ARRAYS

Address	Type	Name	Bytes	Dimensions
AP-00000008@	I*4	ADAPTER_REGISTERS	20	(5)
2-00000000	CHAR	V1RH780_CONTROL_REGISTER	51	(0:2)
2-00000033	CHAR	V1RH780_STATUS_REGISTER	392	(0:13)

LABELS

Address	Label	Address	Label	Address	Label	Address	Label	Address	Label	Address	Label
1-0000003D	10'	1-00000053	15'	1-00000067	20'	1-0000006E	25'	1-00000082	30'	1-00000097	35'
1-000000B8	40'	1-000000E4	45'	1-000000F9	50'	1-0000011C	55'				

FUNCTIONS AND SUBROUTINES REFERENCED

Type	Name	Type	Name	Type	Name
I*4	COMPRESS4	I*4	LIB$EXTV	I*4	LIB$EXTZV
	LINCHK		OUTPUT		RH780_CONFIGURATION_REGISTER
	RH780_STATUS_REGISTER16_31				

```
0001
0002
0003
0004            subroutine rh750_control_registers (lun,adapter_registers,
0005            1 selected_mapping_register)
0006
0007
0008
0009    c++
0010    c       This routine displays the RH750 adapter registers.  It expects
0011    c       the registers in the following order.
0012    c
0013    c       +-------------------------------+
0014    c       |       garbage longword        |
0015    c       +-------------------------------+
0016    c       |       control register        |
0017    c       +-------------------------------+
0018    c       |        status register        |
0019    c       +-------------------------------+
0020    c       |   virtual address register    |
0021    c       +-------------------------------+
0022    c       |      byte count register      |
0023    c       +-------------------------------+
0024    c--
0025
0026
0027
0028
0029            byte            lun
0030
0031            integer*4       adapter_registers(5)
0032
0033            integer*4       selected_mapping_register
0034
0035            byte            diagnostic_mode
0036
0037            integer*4       byte_offset
0038
0039            integer*4       selected_map_register
0040
0041            integer*4       cmi_byte_count
0042
0043            integer*4       massbus_byte_count
0044
0045            integer*4       compress4
0046
0047            character*17    v1rh750_control_register(0:2)
0048
0049            data    v1rh750_control_register(0)  /'INITIALIZATION*'/
0050
0051            data    v1rh750_control_register(1)  /'ABORT*'/
0052
0053            data    v1rh750_control_register(2)  /'INTERRUPT ENABLE*'/
0054
0055            character*25    v2rh750_control_register(4:4)
0056
0057            data    v2rh750_control_register(4)  /'"IGNORE BYTE COUNT" MODE*'/
```

```
0058
0059            character*19    v1rh750_status_register(1:1)
0060
0061            data    v1rh750_status_register(1)  /'NO RESPONSE STATUS*'/
0062
0063            character*28    v2rh750_status_register(3:14)
0064
0065            data    v2rh750_status_register(3)  /'ERROR STATUS*'/
0066
0067            data    v2rh750_status_register(4)  /'INVALID MAP*'/
0068
0069            data    v2rh750_status_register(5)  /'PAGE FRAME MAP PARITY ERROR*'/
0070
0071            data    v2rh750_status_register(6)  /''MASSBUS'' DATA PARITY ERROR*'/
0072
0073            data    v2rh750_status_register(7)  /''MASSBUS'' EXCEPTION*'/
0074
0075            data    v2rh750_status_register(8)  /'MISS TRANSFER ERROR*'/
0076
0077            data    v2rh750_status_register(9)  /'WRITE CHECK LOWER ERROR*'/
0078
0079            data    v2rh750_status_register(10) /'WRITE CHECK UPPER ERROR*'/
0080
0081            data    v2rh750_status_register(11) /'DATA LATE*'/
0082
0083            data    v2rh750_status_register(12) /'DATA TRANSFER ABORTED*'/
0084
0085            data    v2rh750_status_register(13) /'DATA TRANSFER COMPLETED*'/
0086
0087            data    v2rh750_status_register(14) /'SILO PARITY ERROR*'/
0088
0089
0090
0091
0092            diagnostic_mode = .false.
0093
0094            if (lib$extzv(3,1,adapter_registers(2)) .eq. 1)
0095            1 diagnostic_mode = .true.
0096
0097            call linchk (lun,2)
0098
0099            write(lun,15) adapter_registers(2)
0100    15      format(/' ',t8,''RH'' CR',t24,z8.8)
0101
0102            if (.not. diagnostic_mode) then
0103
0104            call output (lun,adapter_registers(2),v1rh750_control_register,
0105            1 0,0,2,'0')
0106
0107            call output (lun,adapter_registers(2),v2rh750_control_register,
0108            1 4,4,4,'0')
0109            else
0110
0111            call linchk (lun,1)
0112
0113            write(lun,20) 'DIAGNOSTIC MODE'
0114    20      format(' ',t40,a)
```

```
0115            endif
0116
0117            call linchk (lun,1)
0118
0119            write(lun,25) adapter_registers(3)
0120    25      format(' ',t8,''RH'' SR',t24,z8.8)
0121
0122            if (.not. diagnostic_mode) then
0123
0124            call output (lun,adapter_registers(3),v1rh750_status_register,
0125           1 1,1,1,'0')
0126
0127            call output (lun,adapter_registers(3),v2rh750_status_register,
0128           1 3,3,14,'0')
0129
0130            call rh750_status_register16_31 (lun,adapter_registers(3))
0131            endif
0132
0133            call linchk (lun,1)
0134
0135            write(lun,30) adapter_registers(4)
0136    30      format(' ',t8,''RH'' VAR',t24,z8.8)
0137
0138            if (.not. diagnostic_mode) then
0139
0140            byte_offset = lib$extzv(0,9,adapter_registers(4))
0141
0142            selected_map_register = lib$extzv(9,8,adapter_registers(4))
0143
0144            call linchk (lun,2)
0145
0146            if (byte_offset .eq. 0) then
0147
0148            write(lun,20) 'PAGE ALIGNED'
0149            else
0150
0151            write(lun,35) byte_offset
0152    35      format(' ',t40,i<compress4 (byte_offset)>,'. BYTE, PAGE OFFSET')
0153            endif
0154
0155            write(lun,40) selected_map_register
0156    40      format(' ',t40,'MAPPING REGISTER #',
0157           1 i<compress4 (selected_map_register)>,'. SELECTED')
0158            endif
0159
0160            call linchk (lun,1)
0161
0162            write(lun,45) adapter_registers(5)
0163    45      format(' ',t8,''RH'' BCR',t24,z8.8)
0164
0165            if (.not. diagnostic_mode) then
0166
0167            cmi_byte_count = lib$extv(0,16,adapter_registers(5))
0168
0169            cmi_byte_count = max(0,cmi_byte_count) - min(0,cmi_byte_count)
0170
0171            if (cmi_byte_count .ne. 0) then
```

```
0172
0173            call linchk (lun,1)
0174
0175            write(lun,50) cmi_byte_count
0176    50      format(' ',t40,'"CMI" BYTE COUNT, ',i<compress4 (cmi_byte_count)>,
0177            1 '.')
0178            endif
0179
0180            massbus_byte_count = lib$extv(16,16,adapter_registers(5))
0181
0182            massbus_byte_count = max(0,massbus_byte_count) -
0183            1 min(0,massbus_byte_count)
0184
0185            if (massbus_byte_count .ne. 0) then
0186
0187            call linchk (lun,1)
0188
0189            write(lun,55) massbus_byte_count
0190    55      format(' ',t40,'"MASSBUS" BYTE COUNT, ',
0191            1 i<compress4 (massbus_byte_count)>,'.')
0192            endif
0193            endif
0194
0195            selected_mapping_register = selected_map_register
0196
0197            return
0198
0199            end
```

PROGRAM SECTIONS

	Name	Bytes	Attributes
0	$CODE	819	PIC CON REL LCL SHR EXE RD NOWRT LONG
1	$PDATA	306	PIC CON REL LCL SHR NOEXE RD NOWRT LONG
2	$LOCAL	896	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
	Total Space Allocated	2021	

ENTRY POINTS

Address	Type	Name
0-00000000		RH750_CONTROL_REGISTERS

VARIABLES

Address	Type	Name	Address	Type	Name
2-000001B0	I*4	BYTE_OFFSET	2-000001B8	I*4	CMI_BYTE_COUNT
2-000001AF	L*1	DIAGNOSTIC_MODE	AP-00000004@	L*1	LUN
2-000001BC	I*4	MASSBUS_BYTE_COUNT	AP-0000000C@	I*4	SELECTED_MAPPING_REGISTER
2-000001B4	I*4	SELECTED_MAP_REGISTER			

ARRAYS

Address	Type	Name	Bytes	Dimensions
AP-00000008@	I*4	ADAPTER_REGISTERS	20	(5)
2-00000000	CHAR	V1RH750_CONTROL_REGISTER	51	(0:2)
2-0000004C	CHAR	V1RH750_STATUS_REGISTER	19	(1)
2-00000033	CHAR	V2RH750_CONTROL_REGISTER	25	(4:4)
2-0000005F	CHAR	V2RH750_STATUS_REGISTER	336	(3:14)

LABELS

Address	Label	Address	Label	Address	Label	Address	Label	Address	Label	Address	Label
1-00000041	15'	1-00000056	20'	1-0000005D	25'	1-00000071	30'	1-00000086	35'	1-000000A7	40'
1-000000D3	45'	1-000000E8	50'	1-0000010B	55'						

FUNCTIONS AND SUBROUTINES REFERENCED

Type	Name	Type	Name	Type	Name
I*4	COMPRESS4	I*4	LIB$EXTV	I*4	LIB$EXTZV
	LINCHK		OUTPUT		RH750_STATUS_REGISTER16_31

```
0001
0002
0003
0004            subroutine mba_status_register16_31 (lun,register1,register2,flag)
0005
0006
0007
0008    c++
0009    c       'flag' is used in the following way.  If flag is equal to 0
0010    c       then the status represented in 'register1' is output.  If flag
0011    c       is equal to 1 then the status represented by the difference of
0012    c       additionally set bits in 'register1' and 'register2'.
0013    c--
0014
0015
0016            include 'src$:msghdr.for /nolist'
0076
0077
0078
0079            byte            lun
0080
0081            integer*4       register1
0082
0083            integer*4       register2
0084
0085            byte            flag
0086
0087            integer*4       pseudo_status_register
0088
0089            integer*4       status_register1_bits
0090
0091            integer*4       status_register2_bits
0092
0093
0094
0095            if (flag .eq. 0) then
0096
0097            pseudo_status_register = iand(register1,'ffff0000'x)
0098
0099            else if (flag .eq. 1) then
0100
0101            status_register1_bits = iand(register1,'ffff0000'x)
0102
0103            status_register2_bits = iand(register2,'ffff0000'x)
0104
0105            pseudo_status_register =
0106            1 iand(not(status_register1_bits),status_register2_bits)
0107            endif
0108
0109            if (pseudo_status_register .ne. 0) then
0110
0111            if (
0112            1 lib$extzv(24,8,emb$l_hd_sid) .eq. 255
0113            1 .or.
0114
0115
```

```
0116            1 lib$extzv(24,8,emb$l_hd_sid) .eq. 1
0117            1 ) then
0118
0119            call rh780_status_register16_31 (lun,pseudo_status_register)
0120
0121            else if (lib$extzv(24,8,emb$l_hd_sid) .eq. 2) then
0122
0123            call rh750_status_register16_31 (lun,pseudo_status_register)
0124
0125    c
0126    c       for future MBA support the ELSE-IF-THEN should be expanded
0127    c       at this point.
0128    c
0129
0130            endif
0131            endif
0132
0133            return
0134
0135            end
```

PROGRAM SECTIONS

	Name	Bytes	Attributes
0	$CODE	132	PIC CON REL LCL SHR EXE RD NOWRT LONG
1	$PDATA	8	PIC CON REL LCL SHR NOEXE RD NOWRT LONG
2	$LOCAL	40	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
3	EMB	512	PIC OVR REL GBL SHR NOEXE RD WRT LONG
	Total Space Allocated	692	

ENTRY POINTS

Address	Type	Name
0-00000000		MBA_STATUS_REGISTER16_31

VARIABLES

Address	Type	Name	Address	Type	Name
3-00000000	I*4	EMB$L_HD_SID	3-00000004	I*2	EMB$W_HD_ENTRY
3-0000000E	I*2	EMB$W_HD_ERRSEQ	AP-00000010@	L*1	FLAG
AP-00000004@	L*1	LUN	2-00000000	I*4	PSEUDO_STATUS_REGISTER
AP-00000008@	I*4	REGISTER1	AP-0000000C@	I*4	REGISTER2
2-00000004	I*4	STATUS_REGISTER1_BITS	2-00000008	I*4	STATUS_REGISTER2_BITS

ARRAYS

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

FUNCTIONS AND SUBROUTINES REFERENCED

Type	Name	Type	Name	Type	Name
I*4	LIB$EXTZV		RH750_STATUS_REGISTER16_31		RH780_STATUS_REGISTER16_31

```
0001
0002
0003
0004
0005          subroutine rh780_status_register16_31 (lun,status_register)
0006
0007
0008
0009
0010          byte            lun
0011
0012          integer*4       status_register
0013
0014          character*31    v1rh780_status_register(16:19)
0015
0016          data    v1rh780_status_register(16) /'ATTENTION*'/
0017
0018          data    v1rh780_status_register(17) /''MASSBUS'' CONTROL PARITY ERROR*'/
0019
0020          data    v1rh780_status_register(18) /'NON-EXISTING DRIVE*'/
0021
0022          data    v1rh780_status_register(19) /'PROGRAMMING ERROR*'/
0023
0024          character*25    v2rh780_status_register(29:31)
0025
0026          data    v2rh780_status_register(29) /'CORRECTED READ DATA*'/
0027
0028          data    v2rh780_status_register(30) /'NO RESPONSE CONFIRMATION*'/
0029
0030          data    v2rh780_status_register(31) /'DATA TRANSFER BUSY*'/
0031
0032
0033
0034          call output (lun,status_register,v1rh780_status_register,16,16,19,'0')
0035
0036          call output (lun,status_register,v2rh780_status_register,29,29,31,'0')
0037
0038          return
0039
0040          end
```

PROGRAM SECTIONS

	Name	Bytes	Attributes
0	$CODE	46	PIC CON REL LCL SHR EXE RD NOWRT LONG
1	$PDATA	18	PIC CON REL LCL SHR NOEXE RD NOWRT LONG
2	$LOCAL	336	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
	Total Space Allocated	400	

ENTRY POINTS

Address	Type	Name
0-00000000		RH780_STATUS_REGISTER16_31

VARIABLES

Address	Type	Name	Address	Type	Name
AP-00000004@	L*1	LUN	AP-00000008@	I*4	STATUS_REGISTER

ARRAYS

Address	Type	Name	Bytes	Dimensions
2-00000000	CHAR	V1RH780_STATUS_REGISTER	124	(16:19)
2-0000007C	CHAR	V2RH780_STATUS_REGISTER	75	(29:31)

FUNCTIONS AND SUBROUTINES REFERENCED

Type	Name
	OUTPUT

MB

01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01

```
0001
0002
0003
0004          subroutine rh750_status_register16_31 (lun,status_register)
0005
0006
0007
0008          byte            lun
0009
0010          integer*4       status_register
0011
0012          character*31    v1rh750_status_register(16:19)
0013
0014          data    v1rh750_status_register(16) /'ATTENTION*'/
0015
0016          data    v1rh750_status_register(17) /'"MASSBUS" CONTROL PARITY ERROR*'/
0017
0018          data    v1rh750_status_register(18) /'NON-EXISTENT DRIVE*'/
0019
0020          data    v1rh750_status_register(19) /'PROGRAMMING ERROR*'/
0021
0022          character*17    v2rh750_status_register(23:23)
0023
0024          data    v2rh750_status_register(23) /'CONTROL BUS HUNG*'/
0025
0026          character*20    v3rh750_status_register(29:29)
0027
0028          data    v3rh750_status_register(29) /'CORRECTED READ DATA*'/
0029
0030          character*19    v4rh750_status_register(31:31)
0031
0032          data    v4rh750_status_register(31) /'DATA TRANSFER BUSY*'/
0033
0034
0035
0036
0037          call output (lun,status_register,v1rh750_status_register,16,16,19,'0')
0038
0039          call output (lun,status_register,v2rh750_status_register,23,23,23,'0')
0040
0041          call output (lun,status_register,v3rh750_status_register,29,29,29,'0')
0042
0043          call output (lun,status_register,v4rh750_status_register,31,31,31,'0')
0044
0045          return
0046
0047          end
```

PROGRAM SECTIONS

	Name	Bytes	Attributes
0	$CODE	94	PIC CON REL LCL SHR EXE RD NOWRT LONG
1	$PDATA	22	PIC CON REL LCL SHR NOEXE RD NOWRT LONG
2	$LOCAL	444	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
	Total Space Allocated	560	

ENTRY POINTS

Address	Type	Name
0-00000000		RH750_STATUS_REGISTER16_31

VARIABLES

Address	Type	Name	Address	Type	Name
AP-00000004@	L*1	LUN	AP-00000008@	I*4	STATUS_REGISTER

ARRAYS

Address	Type	Name	Bytes	Dimensions
2-00000000	CHAR	V1RH750_STATUS_REGISTER	124	(16:19)
2-0000007C	CHAR	V2RH750_STATUS_REGISTER	17	(23:23)
2-0000008D	CHAR	V3RH750_STATUS_REGISTER	20	(29:29)
2-000000A1	CHAR	V4RH750_STATUS_REGISTER	19	(31:31)

FUNCTIONS AND SUBROUTINES REFERENCED

Type	Name
	OUTPUT

00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
00
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01
01

```
0001
0002
0003
0004
0005            subroutine mba_mapping_register (lun,mapping_register_number,
0006            1 mapping_register_image)
0007
0008
0009
0010
0011            include 'src$:msghdr.for /nolist'
0070
0071
0072
0073
0074            byte            lun
0075
0076            integer*4       mapping_register_number
0077
0078            integer*4       mapping_register_image
0079
0080            integer*4       compress4
0081
0082
0083
0084
0085            if (
0086            1 lib$extzv(24,8,emb$l_hd_sid) .eq. 255
0087            1 .or.
0088            1 lib$extzv(24,8,emb$l_hd_sid) .eq. 1
0089            1 ) then
0090
0091            call rh780_mapping_register (lun,mapping_register_number,
0092            1 mapping_register_image)
0093
0094            else if (lib$extzv(24,8,emb$l_hd_sid) .eq. 2) then
0095
0096            call rh750_mapping_register (lun,mapping_register_number,
0097            1 mapping_register_image)
0098
0099        c
0100        c       for future MBA support the ELSE-IF-THEN should be expanded
0101        c       at this point.
0102        c
0103
0104            else
0105
0106            call linchk (lun,1)
0107
0108            if (mapping_register_number .ne. -1) then
0109
0110            write(lun,10) mapping_register_number,mapping_register_image
0111        10  format(' ',t8,''RH'' MPR #',i<compress4 (mapping_register_number)>,
0112            1 '.',t24,z8.8)
0113            else
0114
0115            write(lun,15) mapping_register_image
```

```
0116     15       format(' ',t8,''RH'' MPR #???',t24,z8.8)
0117              endif
0118              endif
0119
0120              return
0121
0122              end
```

PROGRAM SECTIONS

	Name	Bytes	Attributes
0	$CODE	235	PIC CON REL LCL SHR EXE RD NOWRT LONG
1	$PDATA	70	PIC CON REL LCL SHR NOEXE RD NOWRT LONG
2	$LOCAL	56	PIC CON REL LCL NOSHR NOEXE RD WRT LONG
3	EMB	512	PIC OVR REL GBL SHR NOEXE RD WRT LONG
	Total Space Allocated	873	

ENTRY POINTS

Address	Type	Name
0-00000000		MBA_MAPPING_REGISTER

VARIABLES

Address	Type	Name	Address	Type	Name
3-00000000	I*4	EMB$L_HD_SID	3-00000004	I*2	EMB$W_HD_ENTRY
3-0000000E	I*2	EMB$W_HD_ERRSEQ	AP-00000004@	L*1	LUN
AP-0000000C@	I*4	MAPPING_REGISTER_IMAGE	2-00000000@	I*4	MAPPING_REGISTER_NUMBER

ARRAYS

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

LABELS

Address	Label	Address	Label
1-00000000C	10'	1-0000002C	15'

FUNCTIONS AND SUBROUTINES REFERENCED

Type	Name	Type	Name	Type	Name
I*4	COMPRESS4	I*4	LIB$EXTZV		LINCHK
	RH750_MAPPING_REGISTER		RH780_MAPPING_REGISTER		

```
0001
0002
0003
0004
0005            subroutine rh780_mapping_register (lun,mapping_register_number,
0006            1 mapping_register_image)
0007
0008
0009
0010            byte            lun
0011
0012            integer*4       mapping_register_number
0013
0014            integer*4       mapping_register_image
0015
0016            integer*4       compress4
0017
0018            integer*4       compressf
0019
0020            integer*4       pfn
0021
0022            real*4          transfer_address
0023
0024
0025
0026
0027            call linchk (lun,1)
0028
0029            if (mapping_register_number .ne. -1) then
0030
0031            write(lun,10) mapping_register_number,mapping_register_image
0032    10      format(' ',t8,''RH'' MPR #',i<compress4 (mapping_register_number)>,
0033            1 '.',t24,z8.8)
0034            else
0035
0036            write(lun,15) mapping_register_image
0037    15      format(' ',t8,''RH'' MPR #???',t24,z8.8)
0038            endif
0039
0040            if (lib$extzv(31,1,mapping_register_image) .eq. 1) then
0041
0042            call linchk (lun,2)
0043
0044            write(lun,20) 'VALID'
0045    20      format(' ',t40,a)
0046
0047            pfn = lib$extzv(0,21,mapping_register_image)
0048
0049            transfer_address = real(pfn)/2
0050
0051            write(lun,25) transfer_address
0052    25      format(' ',t40,'TRANSFER PAGE, ',
0053            1 f<compressf (transfer_address,1)>.1,'. K')
0054            endif
0055
0056            return
0057
```

```
0058          end


PROGRAM SECTIONS

    Name                              Bytes   Attributes

  0 $CODE                               279   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  1 $PDATA                              125   PIC CON REL LCL   SHR NOEXE   RD NOWRT LONG
  2 $LOCAL                               96   PIC CON REL LCL NOSHR NOEXE   RD   WRT LONG

    Total Space Allocated               500

ENTRY POINTS

    Address  Type  Name

  0-00000000       RH780_MAPPING_REGISTER

VARIABLES

    Address  Type  Name                          Address  Type  Name

 AP-00000004@ L*1  LUN                        AP-0000000C@ I*4  MAPPING_REGISTER_IMAGE
  2-00000008@ I*4  MAPPING_REGISTER_NUMBER     2-00000000  I*4  PFN
  2-00000004  R*4  TRANSFER_ADDRESS

LABELS
    Address   Label       Address   Label       Address   Label       Address   Label
  1-00000019  10'       1-00000039  15'       1-00000053  20'       1-0000005A  25'

FUNCTIONS AND SUBROUTINES REFERENCED
  Type  Name            Type  Name            Type  Name            Type  Name
   I*4  COMPRESS4        I*4  COMPRESSF        I*4  LIB$EXTZV             LINCHK
```

```
0001
0002
0003
0004
0005          subroutine rh750_mapping_register (lun,mapping_register_number,
0006          1 mapping_register_image)
0007
0008
0009
0010
0011          byte            '..
0012
0013          integer*4       mapping_register_number
0014
0015          integer*4       mapping_register_image
0016
0017          integer*4       compress4
0018
0019          integer*4       compressf
0020
0021          integer*4       pfn
0022
0023          real*4          transfer_address
0024
0025
0026
0027
0028          call linchk (lun,1)
0029
0030          if (mapping_register_number .ne. -1) then
0031
0032          write(lun,10) mapping_register_number,mapping_register_image
0033    10    format(' ',t8,''RH'' MPR #',i<compress4 (mapping_register_number)>,
0034          1 '.',t24,z8.8)
0035          else
0036
0037          write(lun,15) mapping_register_image
0038    15    format(' ',t8,''RH'' MPR #???',t24,z8.8)
0039          endif
0040
0041          if (lib$extzv(31,1,mapping_register_image) .eq. 1) then
0042
0043          call linchk (lun,2)
0044
0045          write(lun,20) 'VALID'
0046    20    format(' ',t40,a)
0047
0048          pfn = lib$extzv(0,16,mapping_register_image)
0049
0050          transfer_address = real(pfn)/2
0051
0052          write(lun,25) transfer_address
0053    25    format(' ',t40,'TRANSFER PAGE, '
0054          1 f<compressf (transfer_address,1)>.1,'. K')
0055          endif
0056
0057          return
```

```
0058
0059            end


PROGRAM SECTIONS

    Name                                Bytes   Attributes

  0 $CODE                                 279   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  1 $PDATA                                125   PIC CON REL LCL   SHR NOEXE   RD NOWRT LONG
  2 $LOCAL                                 96   PIC CON REL LCL NOSHR NOEXE   RD   WRT LONG

    Total Space Allocated                 500

ENTRY POINTS

    Address  Type  Name

  0-00000000       RH750_MAPPING_REGISTER

VARIABLES

    Address  Type  Name                            Address  Type  Name

 AP-00000004@ L*1  LUN                          AP-0000000C@ I*4  MAPPING_REGISTER_IMAGE
  2-00000008@ I*4  MAPPING_REGISTER_NUMBER       2-00000000  I*4  PFN
  2-00000004  R*4  TRANSFER_ADDRESS

LABELS
    Address   Label      Address   Label      Address   Label      Address   Label
  1-00000019  10'       1-00000039  15'       1-00000053  20'       1-0000005A  25'

FUNCTIONS AND SUBROUTINES REFERENCED
  Type  Name            Type  Name            Type  Name            Type  Name
   I*4  COMPRESS4        I*4  COMPRESSF        I*4  LIB$EXTZV             LINCHK

COMMAND QUALIFIERS
  FORTRAN /LIS=LIS$:MBA/OBJ=OBJ$:MBA MSRC$:MBA
  /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:	10.34 seconds
Elapsed Time:	27.29 seconds
Page Faults:	205
Dynamic Memory:	183 pages

0150 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

LPA11K
LIS

MCHECKS
LIS

LCPRINTER
LIS

MBA
LIS

LOGMSCP
LIS

LINCHK
LIS

MBAINT
LIS

LABEL
LIS

LOGGER
LIS

MASSTAPE
LIS

MASSDISK
LIS