

PRODUCT ID: ZZ-ECKAM-3.2
PRODUCT TITLE: MS750 MEMORY DIAGNOSTIC
DECO/DEPO: 3.2
DATE: 23-MAR-1987
DEPARTMENT: PSE DIAGNOSTICS AND MICROCODE

THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
CORPORATION.

DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.

PRODUCT ID: ZZ-ECKAM-3.2
PRODUCT TITLE: MS750 MEMORY DIAGNOSTIC
DECO/DEPO: 3.2
DATE: 23-MAR-1987
DEPARTMENT: PSE DIAGNOSTICS AND MICROCODE

COPYRIGHT (C) 1980,1986

DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS 01754

THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY ON A SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE, OR ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH SYSTEM AND TO ONE WHO AGREES TO THESE LICENSE TERMS. TITLE TO AND OWNERSHIP OF THE SOFTWARE SHALL AT ALL TIMES REMAIN IN DEC.

THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.

DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.

IDENTIFICATION

PRODUCT ID: ZZ-ECKAM-3.2
PRODUCT TITLE: ECKAM31 MS750 MEMORY SUBSYSTEM DIAGNOSTIC
DECO/DEPO: 3.2
DATE: 20-JUNE-1986
AUTHOR: RAND GRAY, DON MONROE, PAUL HAKALA
DEPARTMENT: BASE SYSTEMS DIAGNOSTIC ENGINEERING

COPYRIGHT (C) 1980, 1981, 1982, 1986
DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS 01754

THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY ON A SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE, OR ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH SYSTEM AND TO ONE WHO AGREES TO THESE LICENSE TERMS. TITLE TO AND OWNERSHIP OF THE SOFTWARE SHALL AT ALL TIMES REMAIN IN DEC.

THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.

DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.

TABLE
OF
CONTENTS

1	ABSTRACT	3
2	HARDWARE REQUIREMENTS	4
3	SOFTWARE REQUIREMENTS	4
4	PREREQUISITES	4
5	LOADING AND STARTING	4
6	PROGRAM DESCRIPTION	6
6.1	Test 1: Memory Map Verification Test	7
6.2	Test 2: Data Bus Test	8
6.3	Test 3 Row Select Bus Test	9
6.4	Test 4: Address Bus Test	11
6.5	Test 5: ECC Logic Test Functional Description:	11
6.6	Test 6: CSRO Test	13
6.7	Test 7: Bootstrap Rom Test	14
6.8	Test 8: CPU Lost Error Test	14
6.9	Test 9: CPU XB Lost Error Test	14
6.10	Test 10: Moving Inversions Test	15
6.11	Test 11: Moving Inversion With Manual Array Select	16
6.12	Test 12: Memory Quick Verify Test	17
7	MAINTENANCE HISTORY	17

1 ABSTRACT

This program detects and isolates VAX 11/750 memory subsystem failures. It is a macro level program which runs under the VAX-11 Diagnostic Supervisor. It is intended for use only on a VAX 11/750 system. The diagnostic consists of the following tests:

1. Memory Map Verification Test
2. Data Bus Test
3. Row Select Bus Test
4. Address Bus Test
5. ECC Logic Test
6. CSRO Test
7. Bootstrap Rom Test
8. CPU Lost Error Test
9. CPU XB Error Bit Test
10. Moving Inversions (MOVI) Test
11. MOVI with Manual Array Select
12. Memory Quick Verify Test

This program is tailored strictly for the VAX 11/750 memory subsystem, which consists of memory controller modules (L0011,L0016,L0022) and from 1 to 8 memory array modules. The array modules may be either 256kb, 1024kb, or 4Mb.

The 128kb is not supported in this version of ECKAM. The program will run with the L0011, L0016, or L0022 controller. This version of ECKAM supports the 4Mb and 1024kb arrays when the L0022 controller is present. When a mixed configuration is used, the 4Mb arrays must be inserted consecutively starting in the first slot next to the controller. If used with the L0016, both 256kb and 1024kb arrays can be used. If used with the L0011, only 256kb arrays can be used. If a mix configuration is used, 1024kb arrays MUST be inserted first starting with the first slot next to the controller. It should also be noted that array testing (tests 10, 11, and 12) begins testing 1 page (row boundary aligned) above the space allocated by the Diagnostic Supervisor. If in doubt of the reliability of this portion of memory, a manual swap of the array cards, to a higher slot should be made. The first printout of the array under test, (test 10, 11, 12) will tell you where testing had begun.

NOTE

256kb arrays = M8728
1024kb arrays = M8750
4Mb arrays = M7199

2 HARDWARE REQUIREMENTS

VAX-11/750 Processor
512kb of memory
Console terminal
Load device

3 SOFTWARE REQUIREMENTS

The VAX-11/750 Diagnostic Supervisor must be loaded.

4 PREREQUISITES

EVKAA Hardcore Instruction Test and EVKAB/EVKAC Cluster
Exerciser must run successfully.

Successful operation of this program requires that the lowest
256kb of memory is usable. That is to say, that there are no
solid, uncorrectable errors in the lowest 256kb.

5 LOADING AND STARTING

Load the Diagnostic Supervisor by typing the following command
on the con-sole:

>>> B DDA0 ! Boot the Diagnostic Supervisor from the TU58

or:

>>> B/10 XXYn ! To boot the Diagnostic Supervisor from the selected
device.

The Supervisor takes a few seconds to initialize, and then
outputs the prompt

DIAGNOSTIC SUPERVISOR. ZZ-ECSAA-V6.X-YYY DATE

DS>

on the console. The program may be run in various configurations. To start the program in its default configuration, simply type

```
DS> SET T[RACE] <CR>
DS> RU[N] ECKAM <CR>
```

The square brackets mean that the enclosed characters are optional. Thus, SET T may be typed instead of SET TRACE if it is desired. By setting the trace flag, the title of each test is typed on the console as each one starts.

The following is a typical example of what will be displayed on the console in a normal, error-free pass:

```
.. PROGRAM: ECKAM-REV. X.YY MS750 MEMORY DIAGNOSTIC, REV X.YY, 12 TESTS.
```

```
*****
WARNING: DIAGNOSTIC DEFAULTS TO AN EXHAUSTIVE RUN.
FOR A QUICK VERIFY OF THE MEMORY,TYPE CTRL C AND
ABO. WHEN RETURNED TO DS PROMPT, SET FLAG QUICK
AND RESTART THE PROGRAM.
NOTE ALSO, ONLY MEMORY ABOVE THE DIAGNOSTIC
SUPERVISOR IS TESTED. ARRAY + ROW UNDER TEST
PRINTOUTS IN TESTS 10, 11, AND 12 WILL INDICATE
WHAT'S BEING TESTED.
*****
```

```
HIGHEST LONGWORD ADDRESS IS 0007FFFC(X)
TEST 1: MEMORY MAP VERIFICATION TEST
MEMORY MAP VALID: 0000000F(X) FOR: L0011 CONTROLLER
SLOT[0] IS A 256KB ARRAY
SLOT[1] IS A 256KB ARRAY
SLOT[2] IS EMPTY
SLOT[3] IS EMPTY
SLOT[4] IS EMPTY
SLOT[5] IS EMPTY
SLOT[6] IS EMPTY
SLOT[7] IS EMPTY
TEST 2: DATA BUS TEST
TEST 3: ROW SELECT BUS TEST
TEST 4: ADDRESS BUS TEST
TEST 5: ECC LOGIC TEST
TEST 6: CSRO TEST
TEST 7: BOOTSTRAP ROM TEST
TEST 8: CPU LOST ERROR TEST
TEST 9: CPU XB ERROR BIT TEST
TEST 10: MOVING INVERSIONS TEST
ARRAY[1] SINGLE BIT ERRORS:
  ROW 0 - 0
  ROW 1 - 0
  ROW 2 - 0
  ROW 3 - 0
```

THERE WERE NO CORRECTABLE ERRORS
TEST 12: MEMORY QUICK VERIFY TEST [aborted]
.. END OF RUN. 0 ERRORS DETECTED. PASS COUNT: 1.

Test 1 gives a configuration of the memory subsystem. The memory subsystem hardware should agree with this printout. Test 10 lists statistics of single bit errors for each 64kb, 256kb or 1024kb row of each array under test. Any double bit error will be reported explicitly. To manually select array cards to test, after the program is loaded type

DS> ST/SE[CTION]:[MAN]UAL <CR>

The program will then ask for the low array board number and the high array board number to test. If any single bit errors are detected, invoking the SUMMARY section will print the failing page address and bit number of the first 255 errors. If more than 256 single bit errors are detected, a message will be typed indicating that the error log is full. At this point the test will abort the program.

DS> SUM[MARY]

DS> SET FLAG QUICK <CR> Initiates the Quick Verify section. All tests will run with the exception of the Moving Inversions (test 10). Test 12 (Memory Quick Verify Test) will run instead. This test was designed to detect any GROSS memory errors.

RUN TIME: Test 10 Moving Inversions will take approximately 10 minutes per 256kb array, 45 minutes per 1024kb array, and 3 hours per 4Mb array. This test was design to detect any intermittent memory errors.

Test 12 Memory Quick Verify Test will take approximately 15 seconds per 256kb array, 1 minute per 1024kb array, and 3 hours per 4Mb array.

6 PROGRAM DESCRIPTION

To understand the test algorithms in this program, it is necessary to understand the organization of a memory made up of semiconductor RAM chips. It is recommended that the user read the hardware manual describing the VAX 11/750 memory subsystem for details of the subsystem organization. However, a brief discussion of the VAX 11/750 subsystem will be given here.

The 256kb module (M8728), the 1024kb array (M8750), and the 4Mb array (M7199) are both organized in four rows of 39 RAM chips. Each RAM chip is, respectively, 16k(256kb array), 64k(1024k), or 256k(4M) by 1 bit in size. Thus, there are a total of 156 RAM ICs per array module.

Each row consists of one longword (four bytes of data) and 7 checkbits for the ECC logic. 19 bits are necessary to address any given 256kb array, 21 bits for 1024kb arrays, and 23 bits for 4Mb arrays. The lower 14 bits give the address inside any row for 256kb array, the lower 16 for 1024kb arrays, and the lower 18 for 4Mb arrays. The upper 5 bits provide the row selection. Row size would be 64kb or 10000(hex) for 256kb arrays, 256kb or 40000(hex) for 1024kb arrays, or 1024kb (100000 hex) for 4Mb arrays. There are 32 possible rows in a maximum configuration sub-system (four rows per board times eight boards).

The 14/16 bit address bus is routed to each chip in parallel, as is the 32 bit data bus and 7 bit ECC bus. The row select lines are routed in parallel to each chip, but go to different pins on different chips.

Therefore, there are three distinct buses: the data bus, the row select bus, and the address bus. Typically, a failure on one of these three buses will appear as though a great many RAMs are failing. It is therefore desirable to test these buses before testing any individual RAM chips. Unfortunately, a single failing RAM could cause a test of one of the buses to fail, providing misleading results. This diagnostic employs algorithms which ignore individual RAM faults when testing the buses so that faults on array boards can be isolated reliably and automatically.

6.1 Test 1: Memory Map Verification Test

In this test the memory map (bits <15:00> of Control and Status Register 2) is verified. The memory map consists of 8 2-bit cells, each of which contains a code representing the contents of each of the eight array slots of the memory subsystem backplane. The map code differs for L0016 controllers, because half-populated (128kb) boards are illegal due to design differences of the M8750 array. The map code for the L0022 differs because it does not support 256kb arrays. This test will determine what type of controller is present and take the appropriate action. The test also checks the max longword address found by reading memory, to the max longword address found as indicated by the map in CSR2. If they disagree a hard error will occur. The code for the L0011, L0016, and L0022 is as follows.

L0011:

- 1 - half populated (128kb), is no longer supported
- 2 - register error
- 3 - 256kb array
- 0 - empty

L0016:

1 - register error
 2 - 1024kb array
 3 - 256kb array
 0 - empty

L0022:

0 = empty
 1 = 4Mb array
 2 = 1Mb array
 3 = register error

For example, a L0016 with a 1024kb array in slot 0, and a 256kb array in slots 1 and 2, and the rest of the slots empty would be indicated by the following map:

CSR 2

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0

Test algorithm:

The register is scanned 2 bits at a time, starting at cell 0. An error is caused by any 3 following a 0 or 1, or any 2 following any 0, or 3.

The contents of the map are always printed whether there were any errors or not.

6.2 Test 2: Data Bus Test

In this test the data bus is checked for bits stuck at one, stuck at zero, and shorted together. The test ignores individual RAM faults. The reason for this is that at this time we are only interested in the the data bus. Stuck or shorted RAMs will be detected in a later test. This partitioning of the tests is necessary to provide automatic fault isolation.

Test algorithm:

1. Write a pattern of all zeros to the longword 0 of each row.

2. Read bit zero of each written longword and store its value.
3. Write a pattern of all ones to longword 0 of each row.
4. Read bit zero of each written longword and store its value. Check that bit 0 of the written longword is able to toggle (change state from 0 to 1). If bit 0 never toggles, then bit 0 of the data bus is stuck.
5. Repeat steps 1 through 4 for bits 1 through 31 of the data bus.

NOTE

Note: This test uses the smallest of the rows (64k boundaries) for algorithms. test

The algorithm for this test is based on an idea by Srini, from his paper, "Fault Location in a Semiconductor Random-Access Memory Unit," IEEE TRANSACTIONS on COMPUTERS, Vol. C-27, No. 4, April, 1978.

6.3 Test 3 Row Select Bus Test

In this test the row select (bank select) is checked for stuck at zero or one faults. The memory array boards are organized in rows of 39 16k by 1 bit Ram Ic's. These Rams are either 16kb, 64kb, or 256kb depending upon the array type. Each row (bank) is either 64kb (10000 hex), 256kb (40000 hex), or 1024 (100000 hex) in density. The address selection circuit of the memory controller applies INT BUS MA <15:14> to the row select circuit of the memory array. The row select circuit decodes INT BUS MA <15:14>, thus generating the row address strobes RAS SEL <A:D> L which selects one of the four banks of rams. If there is a stuck or shorted bit in the row select it would show up as a dual addressing syndrome. For instance, consider the case of row select 0 stuck at zero. Writing a unique pattern to address 0, row 0 (the 24 bit CMI address would be 000000), and then writing a second unique pattern to address 0, row 1 (CMI address 010000) would overwrite the first pattern. One way of detecting a stuck or shorted bit in this bus would be to write unique patterns in the same 14/16 bit address of each row, and then verify that all of the unique patterns were intact. A convenient set of patterns would be the sequence 0, 1, 2, 3, ...

Assume in the following table for a 256kb array that row select bus bit 2 is stuck at zero:

Row Select Bus (binary)	CMI Address (hex)	Data Written (decimal)	Data Read (decimal)
00000	000000	0	2
00001	010000	1	3
00010	020000	2	2
00011	030000	3	3
00100	040000	4	6
00101	050000	5	7
00110	060000	6	6
00111	070000	7	7
11110	1E0000	30	30
11111	1F0000	31	31

What is occurring here is that whenever any address with row select bit 2 asserted is accessed, the address actually accessed is that address with bit 2 at zero. Therefore, when a 2 was written into location 20000 (hex), it was really written into location 000000 instead, thus overwriting the 0 that was originally there. And when a 3 was written into location 30000 (hex), it was really written to location 10000 (hex), thus overwriting the 1 that had previously been written there (and so on).

The trouble with using the simple sequence 0, 1, 2, ... is that a faulty RAM chip might cause the results of the test to be misleading. Therefore, to make the test insensitive to RAM faults, the values 0, 1, 2, ... are encoded into an error-correcting code. In this test (and in the following test) a Reed-Muller (32,6) code is employed. This code provides for a Hamming distance of 7 between codewords. Thus, the test tolerates a maximum of 6 faulty RAMs without invalidating the test results.

Test algorithm:

1. Write a unique code word in each row (bank) of all present arrays under test. The code word is the boolean product of the row number and a Reed-Muller code word of (32,6). The assignment of the codeword to a row (bank) is arbitrary.
2. Read each word written in step 1, and decode them. A stuck or shorted row select bit will be indicated if the decoded word does not match its indexed value.

6.4 Test 4: Address Bus Test

In this test the address bus is checked for bits stuck at one, stuck at zero, and shorted together. Each row of an array board consists of 39 16k by 1 bit for 256kb arrays, 64k by 1 for 1024kb arrays, or 256k by 1 for 4Mb arrays, of semiconductor RAMs. A fourteen/sixteen bit address bus runs in parallel to all of the RAMs in a row. To ascertain there are no problems with the address bus, a scheme similar to that of Test 3 is employed.

Test algorithm:

1. Write a unique pattern into each location of the address sequence 0, 1, 2, 4, 8, ..., 16384 for 16k rams, 65536 for 64k rams, or 262144 for 256k rams of each row of all present arrays. The patterns are Reed-Muller (32,6) codewords representing the values in the sequence 0, 1, 2, 3, ...
2. Read each word written in step 1 and decode. A stuck or shorted address bit will result in a mismatch of the decoded word and its expected value.

This test tolerates a maximum of 6 RAM failures per row without invalidating the test results.

6.5 Test 5: ECC Logic Test Functional Description:

In this test the ECC logic is functionally verified.

Test algorithm:

1. Check first that the diagnostic bits ECC Disable Mode, Page Mode, Diagnostic Check Mode, and Enable Interrupt which are all in CSR1, are not stuck at 0 or 1.
2. Next, check that page mode works. To do this, we put the controller in ECC Disable Mode and Page Mode, write a location in the given page with a pattern. This stores the check bits into CSR1<06:00>. The check bits are read back again, checking that they are as expected. Now turn off Page Mode and read the check bits in CSR1. With page mode off, the pattern will change with any read from memory, including an instruction-stream fetch.

3. Determine if disabling single-bit error reporting works. Turn off the report single-bit error bit in CSR1. Force a single-bit error, as in step 5. Check that no interrupt occurred, and that the single-bit error flag did not set in CSR0.
4. Write a series of patterns into memory in ECC Disable Mode. For each write, check the contents of CSR1<6:0> (the check bits). Read each location, and verify the check bits for each read cycle. The patterns are chosen to verify each gate in each checkbit generator.
5. Put the controller into ECC Disable Mode and write a pattern into memory. This stores the generated checkbits into CSR1. Turn off ECC disable Mode and write a new pattern into the same location (the pattern is chosen to simulate a single bit error in the first pattern written). Put the controller into Diagnostic Check Mode and read the pattern. This should cause the correctable error flag to set, and the page address of the error to be logged. Check that the error syndrome is correct. Check also that the failing bit was corrected. Check also that bit 0 in the CPU BUS ERROR register sets. Repeat for all 32 bits for a one and a zero.
6. Turn off Diagnostic Check Mode, and write a third pattern into the same location which will cause a double bit error when Diagnostic Check Mode is turned on again. Check that the uncorrectable error flag is set, and that the error is properly logged in CSR0. Check that the error syndrome is correct. Check that memory did not change.
7. With the uncorrectable error flag still set, force an additional error in a different page and check to see if the uncorrectable error, information lost flag is set. Check that the page address did not change. Also check that BUS ERROR REGISTER has the LOST ERROR bit set (bit 1).
8. Force a single bit error, as in step 5, with the single bit error interrupt enabled. Check that the interrupt occurred.
9. Determine if disabling single bit error reporting also inhibits the setting of the error bit in CSR0 and latching of the address in CSR0. Perform step 3 and check that bit <29> is clear and that bits <23:9> are unchanged in CSR0.
10. Check all combinations of two bits in error in a field of zero's and a field of one's. The data patterns used will be as follows:

Field of Zero's	Field of One's
00000003	FFFFFFFC
00000005	FFFFFFFA
80000001	7FFFFFFE
00000006	FFFFFFF9
00000009	FFFFFFF5
80000002	7FFFFFFD
C0000000	3FFFFFFF

11. Force single-bit errors by changing the check bits for a given pattern. Check that the error is detected and that the syndrome is correct. This is done by loading a data pattern whose check bits are all zero (or one) and floating a zero (or one) through the check bits. The syndrome bit asserted will correspond to the check bit be tested.

6.6 Test 6: CSRO Test

This test, tests additional functions of CSRO that have not previously been tested. This entire test was added in Version 01-02.

Test algorithm:

1. Check that there are no stuck bits in the error address field of CSRO. This is done by forcing a single bit error in every page (starting with the second array card above the Diagnostic Supervisor) in all available memory and verifying the error address is correct.
2. Check that a double bit error overwrites the error address and syndrome of a single bit error. This is done by forcing a single bit error in one page followed by a double bit error in a different page and verifying the error address is that of the second page.
3. Check that the uncorrectable error bit in CSRO cannot be cleared if the Information Lost bit is set.
4. Check that the correctable and uncorrectable and information lost bits can be cleared individually.

5. Check that CSRD dose not update when a single bit error is forced with the double bit error flag already set.

6.7 Test 7: Bootstrap Rom Test

In this test, the Bootstrap Roms are tested. The test scans all 4 Rom Addresses to ensure that the addresses respond. The 4 addresses are:

F20400, F20500, F20600, and F20700

Then, the Rom is checked to see if it is present. If the first byte of the Rom is not all zero's or all one's, then it is assumed to be present.

If the Rom is present, the Rom number (A, B, C, or D) is typed with the two character device name that it will boot. Then, the BYTE checksum of the Rom is calculated and compared with the contents of the last byte of the Rom.

If no ROMS are found, a message is typed indicating so.

This entire test was added in version 01-02.

6.8 Test 8: CPU Lost Error Test

This test checks that the LOST ERROR bit in the CPU BUS ERROR register sets correctly. This is done by forcing two errors without clearing the BUS ERROR bit in the MACHINE CHECK ERROR SUMMARY register after the first error.

1. Force two nonexistent memory errors.
2. Force two uncorrectable data errors.
3. Force two correctable data errors.

This entire test was added in version 01-02.

6.9 Test 9: CPU XB Lost Error Test

This test checks that XB bit in the MACHINE CHECK ERROR SUMMARY register functions correctly. This is done by forcing a NXM, and UNCORRECTABLE error with a DATA reference and checking that the bit does not set and forcing the errors with an I-STREAM reference and checking that the bit sets.

1. Nonexistent memory error.
2. Uncorrectable memory error.

This entire test was added in version 01-02.

6.10 Test 10: Moving Inversions Test

In this test the arrays are tested rigorously. The moving inversions algorithm is used in this test, providing an exhaustive test of the RAMs. This test will take approximately 10 minutes per 256kb array, 45 minutes per 1024kb array, and 3 hours per 4Mb array. Set flag quick for a quick verify of the memory.

Test algorithm:

1. Write all rows of all arrays under test with all zeros.
2. Go to the beginning of the first row under test. Read all zeros, write all ones, and read all ones. Continue sequentially until all longword in this row are filled with all ones.
3. Go back to the beginning of the first row under test. Read all ones, write all zeros, and read all zeros. Continue sequentially until all longword in this row are filled with all zeros.
4. Go to the end of the first row under test. Read all zeros, write all ones, and read all ones. Continue backwards until all longword in this row are filled with all ones.
5. Go back to the end of the first row under test. Read all ones, write all zeros, and read all zeros. Continue backwards until all longword in this row are filled with all zeros.
6. Repeat steps 2 and 3 for 14 or 16 passes (the number of address bits per chip) dependent on type of array. For each successive pass, a different address bit is used as the least significant bit in counting such that every address line is toggled at the fastest possible rate.
7. Repeat steps 2 - 6 for data patterns which provide the check bit memory with complementing patterns.

8. Repeat step 1 through 7 for all rows of all arrays under test.

6.11 Test 11: Moving Inversion With Manual Array Select

This test is identical to the previous test, except that the user may select the arrays he desires to test.

Test algorithm:

1. Write all rows of all arrays under test with all zeros.
2. Go to the beginning of the first row under test. Read all zeros, write all ones, and read all ones. Continue sequentially until all longword in this row are filled with all ones.
3. Go back to the beginning of the first row under test. Read all ones, write all zeros, and read all zeros. Continue sequentially until all longword in this row are filled with all zeros.
4. Go to the end of the first row under test. Read all zeros, write all ones, and read all ones. Continue backwards until all longword in this row are filled with all ones.
5. Go back to the end of the first row under test. Read all ones, write all zeros, and read all zeros. Continue backwards until all longword in this row are filled with all zeros.
6. Repeat steps 2 and 3 for 14 or 16 passes (the number of address bits per chip) dependent on array type. For each successive pass, a different address bit is used as the least significant bit in counting such that every address line is toggled at the fastest possible rate.
7. Repeat steps 2 - 6 for data patterns which provide the check bit memory with complementing patterns.
8. Repeat step 1 through 7 for all rows of all arrays under test.

6.12 Test 12: Memory Quick Verify Test

In this test the arrays are tested for any gross errors. A memory march algorithm is used in this test, providing a quick check of all the RAMs. Any gross errors will be detected. Set the Quick Flag to invoke this test.

Test algorithm:

1. Write all rows of all arrays under test with all zeros. One row at a time. (A longword at a time, row size dependent on array type either 64/256/1024kb boundaries).
2. Go to the beginning of the first row under test. Read all zeros, write all ones, and read all ones. Continue sequentially until all longword in this row are filled with all ones.
3. Go back to the beginning of the first row under test. Read all ones, write all zeros, and read all zeros. Continue sequentially until all longword in this row are filled with all zeros.
4. Go to the end of the first row under test. Read all zeros, write all ones, and read all ones. Continue backwards until all longword in this row are filled with all ones.
5. Go back to the end of the first row under test. Read all ones, write all zeros, and read all zeros. Continue backwards until all longword in this row are filled with all zeros.
6. Repeat steps 2 - 5 for data patterns which provide the check bit memory with complementing patterns.
7. Repeat step 1 through 7 for all rows of all arrays under test.

7 MAINTENANCE HISTORY

AUTHOR: Rand Gray, CREATION DATE: 22-Jul-78

MODIFIED BY: Don Monroe

00-02 May 1979

Fixed documentation in CORR_ERR_ISR routine.
Added error messages 14 for ECC test.
Added code to disable single bit error interrupts in the

- initialize routine.
- Added a check for max error count when logging correctable and uncorrectable errors.
- 00-03 June 1979
See module M1.
- 00-04 July 1979
Changed Correctable Error Interrupt Routine to handle new Stack format.
- 00-05 July 1979 Don Monroe
Added typeout of VA on Unexpected Uncorrectable Error.
- 00-06 August 1979 Don Monroe
Added the saving of the Bus Error Register on Single Bit Error Interrupts.
- 00-07 September 1979 Don Monroe
Added documentation to expected error flags. Added a forced loop to unexpected uncorrectable errors.
- 00-08 October 1979 Don Monroe
Added the SUMMARY section to print the address and syndrome of correctable errors detected in the moving inversions test.
- 00-09 October 15 1979 Don Monroe
Fixed summary message and bootstrap ROM messages on multiple passes. Fixed UNEXPECTED CORRECTABLE ERROR when trying to execute multiple passes of tests 2,3,4,10, and 11.
- 00-10 March 3, 1980 Don Monroe
Modified SUMMARY code to print the bit in error instead of just the syndrome. Modified all ERRHARD calls to use a PRINT LINK to the PRINT_GENERAL routine. Fixed bugs in test 6.
- 01-00 June 16, 1980 Don Monroe
Initial Release.
- 01-01 July 10, 1980 Don Monroe
Fixed bug in INIT that sizes memory. Loop was one too small for a fully populated memory (2 MB).
- 01-02 September 17, 1980 Don Monroe
Modified test 6,10,11 to only test pages outside the size of the Diagnostic Supervisor.
- 01-03 May 27, 1981 Paul Hakala
Fixed bug in INIT that sizes memory. Max Address was wrong if more than four arrays were installed.
- 02-01 December 31, 1981 Paul Hakala
Modified to accommodate both types of memory controllers (L0011, I0016) and both arrays (MS750AA-16K RAMS, MS750CA-64K RAMS).
- 02-02 March 1, 1982 Paul Hakala
Modified tests 10 and 11 to provide better Ram coverage.
- 02-03 March 8, 1982 Paul Hakala
Added a quick verify test for memory to detect any gross errors with the RAMs. (Test 12)
- 02-04 May 1, 1982 Paul Hakala
1. Fixed manipulation of Reed-Muller scheme in tests 3 and 4 to function properly.

2. Corrected misleading tests results in test 5,6,and 8 to reflect what actually happens in the error messages.
 3. Added a check to test 1 to determine if there was a memory sizing error.
 4. Added a warning message to the INIT code to inform users of the mode (exhaustive) that the memory defaults to, and also a warning that only memory above the Diagnostic Supervisor is tested.
 5. Modified the single bit error routine to log the actual VA that caused a CRD interrupt.
- 03-00 March 31, 1986 Susan E. Hutcheson
Added for support for M199 4Mb array and L0022 controller.
Search on [3.0]
- 03-01 June 20, 1986 Susan E. Hutcheson Version 3.1
Removed test for full error log from CORR_ERR_ISR and put
it in tests 10,11, and 12 so that ERRHARD for APT would
not be skipped over. Search on [3.1]
- 03-02 January 23,1987 Richard Zecchino Version 3.2
Fixed Problem in Test 12 with Flag1 not being set to
indicate 1024kb array with 256kb rows.

(2)	122	DECLARATIONS
(4)	721	SUMMARY MODULE
(5)	788	PRINT SUBROUTINE
(6)	864	PRINT GENERAL ROUTINE
(7)	912	PRINT UNEXPECTED UNCORRECTABLE ERROR ROUTINE
(8)	950	PRINT MEMORY SIZING ERROR
(9)	978	PRINT CORRUPTED TEST LOCATION
(10)	1008	PRINT UNEXPECTED MACHINE CHECK ROUTINE
(11)	1040	WHICH ARRAY ROUTINE
(12)	1084	DECODER ROUTINE
(13)	1250	INITIALIZATION ROUTINE
(14)	1415	CLEAN UP ROUTINE
(15)	1461	SINGLE BIT ERROR INTERRUPT SERVICE ROUTINE
(16)	1550	MACHINE CHECK SERVICE ROUTINE
(17)	1703	READ MEMORY CONFIGURATION MAP ROUTINE
(18)	1884	GET ARRAY CARDS TO TEST ROUTINE

```
0000 1 .TITLE ECKAM MEMORY DIAGNOSTIC HEADER
0000 2 .IDENT /03-02/
0000 3
0000 4 ;
0000 5 ; COPYRIGHT (C) 1980, 1981, 1982, 1986
0000 6 ; DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS 01754
0000 7 ;
0000 8 ; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY ON A SINGLE
0000 9 ; COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH THE INCLUSION OF THE
0000 10 ; ABOVE COPYRIGHT NOTICE. THIS SOFTWARE, OR ANY OTHER COPIES THEREOF,
0000 11 ; MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY OTHER PERSON
0000 12 ; EXCEPT FOR USE ON SUCH SYSTEM AND TO ONE WHO AGREES TO THESE LICENSE
0000 13 ; TERMS. TITLE TO AND OWNERSHIP OF THE SOFTWARE SHALL AT ALL TIMES
0000 14 ; REMAIN IN DEC.
0000 15 ;
0000 16 ; THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 17 ; AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 18 ; CORPORATION.
0000 19 ;
0000 20 ; DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 21 ; SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.
0000 22 ;
0000 23 ;
0000 24 ;**
0000 25 ; FACILITY: COMET Diagnostic Library
0000 26 ;
0000 27 ; ABSTRACT: This module contains the global data and subroutines for the
0000 28 ; COMET memory diagnostic program.
0000 29 ;
0000 30 ; ENVIRONMENT: Diagnostic Supervisor
0000 31 ;
0000 32 ; AUTHOR: Rand Gray, CREATION DATE: 22-Jul-78
0000 33 ;
0000 34 ; MODIFIED BY: Don Monroe
0000 35 ;
0000 36 ; 00-02 May 1979
0000 37 ; Fixed documentation in CORR_ERR_ISR routine.
0000 38 ; Added error messages 14 for ECC test.
0000 39 ; Added code to disable single bit error interrupts in the
0000 40 ; initialize routine.
0000 41 ;
0000 42 ; Added a check for max error count when logging correctable and
0000 43 ; uncorrectable errors.
0000 44 ;
0000 45 ; 00-03 June 1979
0000 46 ; See module M1.
0000 47 ;
0000 48 ; 00-04 July 1979
0000 49 ; Changed Correctable Error Interrupt Routine to handle new
0000 50 ; Stack format.
0000 51 ;
0000 52 ; 00-05 July 1979 Don Monroe
0000 53 ; Added typeout of VA on Unexpected Uncorrectable Error.
0000 54 ;
0000 55 ; 00-06 August 1979 Don Monroe
0000 56 ; Added the saving of the Bus Error Register on Single Bit Error
0000 57 ; Interrupts.
```

0000	58 :	00-07	September 1979 Don Monroe
0000	59 :		Added documentation to expected error flags. Added a forced
0000	60 :		loop to unexpected uncorrectable errors.
0000	61 :	00-08	October 1979 Don Monroe
0000	62 :		Added the SUMMARY section to print the address and syndrome
0000	63 :		of correctable errors detected in the moving inversions test.
0000	64 :	00-09	October 15 1979 Don Monroe
0000	65 :		Fixed summary message and bootstrap ROM messages on multiple
0000	66 :		passes. Fixed UNEXPECTED CORRECTABLE ERROR when trying to
0000	67 :		execute multiple passes of tests 2,3,4,10, and 11.
0000	68 :	00-10	March 3, 1980 Don Monroe
0000	69 :		Modified SUMMARY code to print the bit in error instead of
0000	70 :		just the syndrome. Modified all ERRHARD calls to use a
0000	71 :		PRINT LINK to the PRINT_GENERAL routine. Fixed bugs in
0000	72 :		test 6.
0000	73 :	01-00	June 16,1980 Don Monroe
0000	74 :		Initial Release.
0000	75 :	01-01	July 10,1980 Don Monroe
0000	76 :		Fixed bug in INIT that sizes memory. Loop was one too small
0000	77 :		for a fully populated memory (2 MB).
0000	78 :	01-02	September 17, 1980 Don Monroe
0000	79 :		Modified test 6,10,11 to only test pages outside the seize of the
0000	80 :		Diagnostic Supervisor.
0000	81 :	01-03	May 27,1981 Paul Hakala
0000	82 :		Fixed bug in INIT that sizes memory. Max Address was wrong if
0000	83 :		more than four arrays were installed.
0000	84 :	02-01	December 31,1981 Paul Hakala
0000	85 :		Modified to accommodate both types of memory controllers
0000	86 :		(L0011,10016) and both arrays (MS750AA-16K RAMS, MS750CA-64K RAMS).
0000	87 :	02-02	March 1,1982 Paul Hakala
0000	88 :		Modified tests 10 and 11 to provide better Ram coverage.
0000	89 :	02-03	March 8, 1982 Paul Hakala
0000	90 :		Added a quick verify test for memory to detect any gross
0000	91 :		errors with the RAMs. (Test 12)
0000	92 :	02-04	May 1, 1982 Paul Hakala
0000	93 :		1. Fixed manipulation of Reed-Muller scheme in tests 3 and 4
0000	94 :		to function properly.
0000	95 :		2. Corrected misleading tests results in test 5,6,and 8 to
0000	96 :		reflect what actually happens in the error messages.
0000	97 :		3. Added a check to test 1 to determine if there was a memory
0000	98 :		sizing error.
0000	99 :		4. Added a warning message to the INIT code to inform users of
0000	100 :		the mode (exhaustive) that the memory defaults to, and also
0000	101 :		a warning that only memory above the Diagnostic Supervisor is
0000	102 :		tested.
0000	103 :		5. Modified the single bit error routine to log the actual VA
0000	104 :		that caused a CRD interrupt.
0000	105 :		
0000	106 :	03-00	March 31, 1986 Susan E. Hutcheson
0000	107 :		Added for support for M1799 4Mb array and L0022 controller.
0000	108 :		Search on [3.0]
0000	109 :		
0000	110 :	03-01	June 20, 1986 Susan E. Hutcheson Version 3.1
0000	111 :		Removed test for full error log from CORR_ERR_ISR and put
0000	112 :		it in tests 10,11, and 12 so that ERRHARD for APT would
0000	113 :		not be skipped over. Search on [3.1]
0000	114 :		

ZZ-ECKAM-3.2
ECKAM
03-02

ECKAM MEMORY DIAGNOSTIC HEADER
MEMORY DIAGNOSTIC HEADER

M 2

13-MAY-1987

Fiche 1 Frame M2

Sequence 25

23-JAN-1987 14:30:12

VAX/VMS Macro V04-00

Page 3

23-JAN-1987 14:26:49

[ZECCHINO.ECKAM]ECKAM0.MAR;4

(1)

0000 115 ; 03-02
0000 116 ;
0000 117 ;
0000 118 ;
0000 119 ;
0000 120 ;--

January 23, 1987 Richard Zecchino Version 3.2
Fixed Problem in Test 12 with Flag1 not being set to
indicate 1024kb array with 256kb rows.

```
0000 0000 122 .SBTTL DECLARATIONS
0000 0000 123 .PSECT HEADER, LONG
0000 0000 124
0000 0000 125 ;
0000 0000 126 ; INCLUDE FILES:
0000 0000 127 ;
0000 0000 128 .LIBRARY /SYS$LIBRARY:DIAG/
0000 0000 129
0000 0000 130 $DS_BGNMOD CEP_REPAIR
0000 0000 ::: Macro-32 Diagnostic macro library Version 10.0 "DIAG.MLB (271)"
0000 0000 131 $DS_ERRDEF
0000 0000 132 ;
0000 0000 133 ; EQUATED SYMBOLS:
0000 0000 134 ;
0000 0000 135 $DS_QADEF
0000 0000 136 $DS_BITDEF
0000 0000 137
0000 0000 138 ;++
0000 0000 139 ; These are PC offsets for accessing memory controller registers
0000 0000 140 ;--
0000 0000 141
0000 0000 142 CSR0 = 0 ; Control/status register 0
0000 0000 143 CSR1 = 4 ; Control/status register 1
0000 0000 144 CSR2 = 8 ; Control/status register 2
0000 0000 145
0000 0000 146 ;++
0000 0000 147 ; This is the memory controller physical address
0000 0000 148 ;--
0000 0000 149
00F20000 0000 150 CONTROLLER = ^XF20000 ; Memory controller 0
0000 0000 151
0000 0000 152 ;++
0000 0000 153 ; These are bit definitions for single bit fields in the memory controller
0000 0000 154 ; Registers.
0000 0000 155 ;--
0000 0000 156
0000 0000 157 ;
0000 0000 158 ; For CSR0
0000 0000 159 ;
0000 0000 160
80000000 0000 161 UCE = BIT31 ; Uncorrectable error flag
40000000 0000 162 UCEIL = BIT30 ; Uncorrectable error, info lost flag
20000000 0000 163 CE = BIT29 ; Correctable error flag
0000 0000 164
0000 0000 165 ;
0000 0000 166 ; For CSR1
0000 0000 167 ;
0000 0000 168
10000000 0000 169 ERCE = BIT28 ; Enable reporting correctable errors A3
08000000 0000 170 PM = BIT27 ; Page mode
04000000 0000 171 DCM = BIT26 ; Diagnostic check mode
02000000 0000 172 EDM = BIT25 ; ECC disable mode
0000 0000 173
0000 0000 174 ;++
0000 0000 175 ; Cache disable register
0000 0000 176 ;--
0000 0000 177
```


ZZ-ECKAM-3.2
ECKAM
03-02

DECLARATIONS

MEMORY DIAGNOSTIC HEADER
DECLARATIONS

C 3

13-MAY-1987

Fiche 1 Frame C3

Sequence 28

23-JAN-1987 14:30:12

VAX/VMS Macro V04-00

Page 6

23-JAN-1987 14:26:49

[ZECCHINO.ECKAM]ECKAM0.MAR;4

(2)

07 0000
0008
4C 4C 41 00' 0008
03 0008
000C
45 56 49 54 53 55 41 48 58 45 00' 000C
0A 000C
0017
4C 41 55 4E 41 4D 00' 0017
06 0017
001E
00000004 001E
00000000' 0022
00000008' 0026
0000000C' 002A
00000017' 002E
0032
30 35 37 53 4D 00' 0032
05 0032
00 0038
00000001 0039
00000032' 003D

S_ALL: .ASCIC \ALL\
S_EXHAUSTIVE: .ASCIC \EXHAUSTIVE\
S_MANUAL: .ASCIC \MANUAL\
SECTION: .LONG 4
.LONG S_DEFAULT
.LONG S_ALL
.LONG S_EXHAUSTIVE
.LONG S_MANUAL
193 \$DS_DEVTYP <MS750>
T_MS750: .ASCIC \MS750\
.BYTE 0
AL_DEVTYP: .LONG \$\$N
.LONG T_MS750

```
00000000 195 .PSECT DATA, LONG
0000 196 ;++
0000 197 ; The following are used in the moving inversions test.
0000 198 ;--
0000 199
00000000 0000 200 DATA:: .LONG 0
00000000 0004 201 SEQUENCE:: .LONG 0
00000000 0008 202 PATCNT:: .LONG 0
000C 203
000C 204 ;++
000C 205 ; This location is set up before any exception is expected. In the exception
000C 206 ; handler, the PC saved on the stack is replaced with this the contents of
000C 207 ; this longword.
000C 208 ;--
000C 209
00000000 000C 210 RETURN_PC:: .LONG 0 ; New PC to use in REI
0010 211
0010 212 ;++
0010 213 ; This location is setup at the beginning of every test. If an unexpected
0010 214 ; uncorrectable error occurs, the PC on the stack is replaced with the
0010 215 ; contents of this longword.
0010 216 ;--
0010 217
00000000 0010 218 UNX_UNC_PC:: .LONG 0 ; New PC to use in REI A7
0014 219 ;++
0014 220 ; This location is used to indicate to the print subroutine which type of
0014 221 ; error is to be printed.
0014 222 ;--
0014 223
00000000 0014 224 ERROR_TYPE:: .LONG 0 ; Indicates type of message to print
0018 225
0018 226 ;++
0018 227 ; The following is used in the ROM test to print the device type of the ROM
0018 228 ;--
0018 229
00 00 02 0018 230 DEVICE_TYPE:: .BYTE 2,0,0 ; Counted ASCII string
00 01 001B 231 ROM_TYPE:: .BYTE 1,0 ; Counted ASCII string
001D 232
001D 233 ;++
001D 234 ; These flags are used to indicate that certain types of exception or
001D 235 ; interrupt are expected.
001D 236 ;--
001D 237
00000000 001D 238 EXP_COR_ERR:: .LONG 0 ; Correctable error expected
0021 239 ; Bit 0 = error expected
0021 240 ; Bit 16 = inhibit clearing BUS ERR REG
00000000 0021 241 EXP_UNC_ERR:: .LONG 0 ; Uncorrectable error expected
0025 242 ; Bit 0 = error expected
0025 243 ; Bit 8 = ignore bus error reg
0025 244 ; Bit 16 = inhibit clearing BUS ERR REG
00000000 0025 245 EXP_INF_LST:: .LONG 0 ; Information lost error expected
0029 246 ; Bit 0 = error expected
0029 247 ; Bit 8 = ignore bus error reg
0029 248 ; Bit 16 = inhibit clearing BUS ERR REG
00000000 0029 249 EXP_NX_MEM:: .LONG 0 ; Non existent memory error expected
002D 250 ; Bit 0 = error expected
002D 251 ; Bit 8 = ignore bus error reg
```

; Bit 16 = inhibit clearing BUS ERR REG

```
002D 252
002D 253
002D 254 ;++
002D 255 ; This location gets loaded with the contents of the BUS ERROR register by
002D 256 ; the machine check routine and corrected data interrupt routine.
002D 257 ;--
002D 258
00000000 002D 259 BUS_ERR_REG:: .LONG 0 ; A2
0031 260
0031 261 ;++
0031 262 ; These longwords keep track of the number of correctable and
0031 263 ; uncorrectable errors logged.
0031 264 ;--
00000000 0031 265 PGE_COR_ERR:: .LONG 0 ; Counts number of correctable errors -
0035 266 ; per page
00000435 0035 267 VA_COR_ADDRESS:: .BLKL 256 ; Keeps track of the actual VA that -
0435 268 ; caused the correctable error
00000000 0435 269 NUM_COR_ERR:: .LONG 0 ; Counts number of CRD errors per row
00000000 0439 270 TOTAL_COR_ERR:: .LONG 0 ; Counts total number of CRD errors
00000000 043D 271 SUMMARY_FLAG:: .LONG 0 ; Non zero means print summary section
0441 272 ; messages A9
00000000 0441 273 ERR_LOG_FULL:: .LONG 0 ; Non zero means error log is full [3.1]
0445 274
0445 275 ;++
0445 276 ; These three locations contain the maximum page address, maximum
0445 277 ; longword address, and maximum number of rows in the memory subsystem.
0445 278 ;--
0445 279
00000000 0445 280 MAXPFN:: .LONG 0 ; This holds the highest page address
00000000 0449 281 MAXLWA:: .LONG 0 ; This holds the highest longword addr
00000000 044D 282 MAXROW:: .LONG 0 ; Number of rows available to test
00000000 0451 283 TEST_ADDRESS:: .LONG 0 ; Used to double check MAXLWA
0455 284 ;++
0455 285 ; These are used to store the low row number, high row number, low address
0455 286 ; and high address used by the manual array select moving inversions test.
0455 287 ;--
0455 288
00000000 0455 289 LOWROW:: .LONG 0
00000000 0459 290 HIGHROW:: .LONG 0
00000000 045D 291 LOWADDR:: .LONG 0
00000000 0461 292 HIGHADDR:: .LONG 0
0465 293
0465 294 ;+
0465 295 ; These locations contain the physical and virtual address of the first
0465 296 ; page of memory above the Diagnostic Supervisor and the first array.
0465 297 ;-
0465 298
00000000 00000000 0465 299 FREE_PAGE_P:: .QUAD
00000000 00000000 046D 300 FREE_PAGE_V:: .QUAD
0475 301
0475 302 ;+
0475 303 ; These are vectors used in the generation of the Reed-Muller codewords
0475 304 ; Which are used in various tests
0475 305 ;-
0475 306
0475 307 CODEVECTORS::
FFFFF000 0475 308 .LONG ^XFFFFFFFF ; Identity vector
```

```

55555555 0479 309 .LONG ^X55555555 ; Vector 1
33333333 047D 310 .LONG ^X33333333 ; Vector 2
0F0F0F0F 0481 311 .LONG ^X0F0F0F0F ; Vector 3
00FF00FF 0485 312 .LONG ^X00FF00FF ; Vector 4
          0489 313
          0489 314 ;+
          0489 315 ; The Reed-Muller (32,6) codewords are stored here after they are generated
          0489 316 ; In the initialization routine.
          0489 317 ;-
          0489 318
00000509 0489 319 CODEWORDS:: .BLKL 32 ; Reserved for Reed-Muller codewords
          0509 320
          0509 321 ;+
          0509 322 ; These are the patterns used to test the ECC logic.
          0509 323 ;-
          0509 324
          0509 325 CKBIT::
8A208880 0509 326 CKBIT0:: .LONG ^B10001010001000001000100010000000
88A222A8 050D 327 .LONG ^B10001000101000100010001010101000
2208A802 0511 328 .LONG ^B00100010000010001010100000000010
A8282082 0515 329 .LONG ^B10101000001010000010000010000010
208A022A 0519 330 .LONG ^B00100000100010100000001000101010
0282AA28 051D 331 .LONG ^B00000010100000101010101000101000
4C404840 0521 332 CKBIT1:: .LONG ^B01001100010000000100100001000000
48C484C4 0525 333 .LONG ^B01001000110001001000010011000100
8408C808 0529 334 .LONG ^B10000100000010001100100000001000
C8488048 052D 335 .LONG ^B11001000010010001000000001001000
808C048C 0531 336 .LONG ^B100000001000110000000010010001100
0484CC84 0535 337 .LONG ^B00000100100001001100110010000100
CCCC4CCC 0539 338 .LONG ^B11001100110011000100110011001100
00805080 053D 339 CKBIT2:: .LONG ^B00000000100000000101000010000000
E0D0F0D0 0541 340 .LONG ^B11100000110100001111000011010000
D0210020 0545 341 .LONG ^B11010000001000010000000000100000
00A0D0A0 0549 342 .LONG ^B00000000101000001101000010100000
30702070 054D 343 .LONG ^B00110000011100000010000001110000
C050A050 0551 344 .LONG ^B11000000010100001010000001010000
70F070F0 0555 345 .LONG ^B01110000111100000111000011110000
B0000000 0559 346 .LONG ^B10110000000000000000000000000000
AD0063F1 055D 347 CKBIT3:: .LONG ^B10101101000000000110001111110001
F100AD92 0561 348 .LONG ^B11110001000000001010110110010010
9200F13F 0565 349 .LONG ^B10010010000000001111000100111111
3F0092CE 0569 350 .LONG ^B00111111000000001001001011001110
CE003F5C 056D 351 .LONG ^B11001110000000000011111101011100
5C00CE63 0571 352 .LONG ^B01011100000000001100111001100011
63005CAD 0575 353 .LONG ^B01100011000000000101110010101101
ACF100CE 0579 354 CKBIT4:: .LONG ^B10101100111100010000000011001110
F092005C 057D 355 .LONG ^B1111000010010010000000001011100
933F0063 0581 356 .LONG ^B10010011001111110000000001100011
2FCE00AD 0585 357 .LONG ^B00101111110011100000000010101101
DE5C00F1 0589 358 .LONG ^B11011110010111000000000011110001
4C630092 058D 359 .LONG ^B01001100011000110000000010010010
73AD003F 0591 360 .LONG ^B01110011101011010000000001111111
01000000 0595 361 .LONG ^B00000001000000000000000000000000
D2B7C400 0599 362 CKBIT5:: .LONG ^B11010010101101111100010000000000
B3C64800 059D 363 .LONG ^B10110011110001100100100000000000
2E48FC00 05A1 364 .LONG ^B00101110010010001111110000000000
DCFF3800 05A5 365 .LONG ^B11011100111111110011100000000000

```

```

6B397000 05A9 366 .LONG ^B01101011001110010111000000000000
41718C00 05AD 367 .LONG ^B0100000101110001100011000000000000
BD8EB400 05B1 368 .LONG ^B1011110110001110101101000000000000
92402840 05B5 369 CKBIT6:: .LONG ^B10010010010000000010100001000000
82686968 05B9 370 .LONG ^B10000010011010000110100101101000
96014001 05BD 371 .LONG ^B10010110000000010100000000000001
04416841 05C1 372 .LONG ^B00000100010000010110100001000001
86290129 05C5 373 .LONG ^B10000110001010010000000100101001
10284128 05C9 374 .LONG ^B00010000001010000100000100101000
14692969 05CD 375 .LONG ^B00010100011010010010100101101001
05D1 376
05D1 377 ;**
05D1 378 ; These are the expected check-bit values resulting from applying the previous
05D1 379 ; set of patterns:
05D1 380 ;--
05D1 381
05D1 382 CK::
02 45 42 28 3B 56 2F 05D1 383 CK0:: .BYTE ^X2F, ^X56, ^X3B, ^X28, ^X42, ^X45
13 46 41 69 3B 14 6E 05D7 384 CK1:: .BYTE ^X6E, ^X14, ^X3B, ^X69, ^X41, ^X46, ^X13
02 6C 3F 38 51 59 68 56 05DE 385 CK2:: .BYTE ^X56, ^X68, ^X59, ^X51, ^X38, ^X3F, ^X6C, ^X02
26 54 07 4E 6F 75 1D 05E6 386 CK3:: .BYTE ^X1D, ^X75, ^X6F, ^X4E, ^X07, ^X54, ^X26
04 30 1B 62 6B 7D 0D 56 05ED 387 CK4:: .BYTE ^X56, ^X0D, ^X7D, ^X6B, ^X62, ^X1B, ^X30, ^X04
78 13 28 6A 7D 04 16 05F5 388 CK5:: .BYTE ^X16, ^X04, ^X7D, ^X6A, ^X28, ^X13, ^X78
54 46 00 2E 7A 12 68 05FC 389 CK6:: .BYTE ^X68, ^X12, ^X7A, ^X2E, ^X00, ^X46, ^X54
3C 0603 390 CKALL:: .BYTE ^X3C
0604 391
0604 392 ;**
0604 393 ; These are the expected syndrome patterns used in the subtest which verifies
0604 394 ; the single-bit error correcting logic.
0604 395 ;--
0604 396
0604 397 SYNDROME::
5B 1A 19 58 0604 398 .BYTE ^B1011000, ^B0011001, ^B0011010, ^B1011011
1F 5E 5D 1C 0608 399 .BYTE ^B0011100, ^B1011101, ^B1011110, ^B0011111
6B 2A 29 68 060C 400 .BYTE ^B1101000, ^B0101001, ^B0101010, ^B1101011
2F 6E 6D 2C 0610 401 .BYTE ^B0101100, ^B1101101, ^B1101110, ^B0101111
73 32 31 70 0614 402 .BYTE ^B1110000, ^B0110001, ^B0110010, ^B1110011
37 76 75 34 0618 403 .BYTE ^B0110100, ^B1110101, ^B1110110, ^B0110111
3B 7A 79 38 061C 404 .BYTE ^B0111000, ^B1111001, ^B1111010, ^B0111011
7F 3E 3D 7C 0620 405 .BYTE ^B1111100, ^B0111101, ^B0111110, ^B1111111
0624 406
0624 407 ;**
0624 408 ; This area is allocated for temporary storage. First, we make sure that we
0624 409 ; enter another page.
0624 410 ;--
00000824 0624 411 .BLKB 512 ; Put the following in another page
0824 412 .ALIGN LONG
000008A4 0824 413 TEMP:: .BLKL 32 ; Temporary storage
00000AA4 08A4 414 .BLKB 512 ; Put the following in another page
00000B24 0AA4 415 TEMP1:: .BLKL 32 ; Temporary storage
0B24 416
0B24 417 ;**
0B24 418 ; These are the patterns for the moving inversions test.
0B24 419 ;--
0B24 420
00000000 0B24 421 MOVI_PAT_0:: .LONG 0
B0000000 0B28 422 .LONG ^XB0000000

```



```

50 41 4D 20 59 52 4F 4D 45 4D 2C 45 1189
      58 5E 36 31 3D 1195
      28 1171
      119A 469 ;++
      119A 470 ; Failing module messages:
      119A 471 ;--
      119A 472
      119A 473 ;+
      119A 474 ; This construct is set up so that the failing module message can be contained
      119A 475 ; in the error header printed by the Diagnostic Supervisor. Since this can
      119A 476 ; not be an FAO control string, the following rather odd-looking construct is
      119A 477 ; used. The subroutine WHICH_ARRAY writes the byte called DIGIT with the
      119A 478 ; ASCII representation of the number of the failing module just before the
      119A 479 ; call to ERRHARD.
      119A 480 ;-
      119A 481
      119A 482     ARRAY::      .BYTE   ARRAY_LEN
      119B 483           .ASCII  aARRAY[a
      11A1 484     DIGIT::  .BYTE   0
      11A2 485           .ASCII  a]a
      11A3 486     ARRAY_LEN =      .-ARRAY-1
      11A3 487
      11A3 488 ALLMOD::      .ASCIC  aSUBSYSTEM FAILUREa
      11AF 489
      11A3 489 CONTROL::      .ASCIC  aCONTROLLER FAILUREa
      11B5 490
      11C1 490 CPU::          .ASCIC  aCPU FAILUREa ; A2
      11B5 491
      11C8 491 ARRAY_SGL_BIT:: .ASCIC  aARRAY FAILUREa
      11C8 492
      11D4 493 ;++
      11E0 494 ; Information messages:
      11D4 495 ;--
      11E2 496
      11E2 497 FMT_MEM_SIZ::      .ASCIC  a!/HIGHEST LONGWORD ADDRESS IS: !XL(X).a
      11EE 498
      11FA 498 FMT_MEM_MAP::      .ASCIC  a!/MEMORY MAP VALID: !XL(X) FOR: L0016 CONTROLLERa
      1206 499
      11E2 499 FMT_MEM_MAP2::      .ASCIC  a!/MEMORY MAP VALID: !XL(X) FOR: L0011 CONTROLLERa
      1209 500
      1215 500 FMT_MEM_MAP3::      .ASCIC  a!/MEMORY MAP VALID: !XL(X) FOR: L0022 CONTROLLERa ;(3.0)
      1221
      122D
      1239
      1209
      123A
      1246
      1252
      125E
      126A
      123A
      126B
      1277
  
```

30 30 4C 20 3A 52 4F 46 20 29 58 28	1283		
45 4C 4C 4F 52 54 4E 4F 43 20 32 32	128F		
	52		
	30		
5D 42 55 21 5B 54 4F 4C 53 2F 21 00	129C	501 FMT_256_KB::	.ASCIC a!/SLOT[!UB] IS A 256KB ARRAYa
20 42 4B 36 35 32 20 41 20 53 49 20	12A8		
	59 41 52 52 41		
	1C		
5D 42 55 21 5B 54 4F 4C 53 2F 21 00	12B9	502 FMT_1024_KB::	.ASCIC a!/SLOT[!UB] IS A 1024KB ARRAYa
42 4B 34 32 30 31 20 41 20 53 49 20	12C5		
	59 41 52 52 41 20		
	1D		
5D 42 55 21 5B 54 4F 4C 53 2F 21 00	12D7	503 FMT_4_MB::	.ASCIC a!/SLOT[!UB] IS A 4MB ARRAYa ;[3.0]
52 41 20 42 4D 34 20 41 20 53 49 20	12E3		
	59 41 52		
	1A		
20 53 49 20 42 4B 38 32 31 2F 21 00	12F2	504 FMT_NO_SUPP::	.ASCIC a!/128KB IS NOT SUPPORTED IN THIS VERSION OF ECKAMa
45 54 52 4F 50 50 55 53 20 54 4F 4E	12FE		
45 56 20 53 49 48 54 20 4E 49 20 44	130A		
4B 43 45 20 46 4F 20 4E 4F 49 53 52	1316		
	4D 41		
	31		
5D 42 55 21 5B 54 4F 4C 53 2F 21 00	1324	505 FMT_EMPTY::	.ASCIC a!/SLOT[!UB] IS EMPTYa
	59 54 50 4D 45 20 53 49 20		
	14		
52 41 43 20 59 41 52 52 41 2F 21 00	1339	506 FMT_AVAILABLE::	.ASCIC a!/ARRAY CARDS !UB - !UB ARE AVAILABLE FOR TEST.a-
42 55 21 20 2D 20 42 55 21 20 53 44	1345		
42 41 4C 49 41 56 41 20 45 52 41 20	1351		
2E 54 53 45 54 20 52 4F 46 20 45 4C	135D		
54 4E 45 20 45 53 41 45 4C 50 2F 21	1369	507	a!/PLEASE ENTER FIRST THE LOW ARRAY, THEN THE HIGH:a
45 48 54 20 54 53 52 49 46 20 52 45	1375		
20 2C 59 41 52 52 41 20 57 4F 4C 20	1381		
47 49 48 20 45 48 54 20 4E 45 48 54	138D		
	3A 48		
	61		
57 4F 4C 20 20 20 20 20 2F 21 00	139B	508 FMT_ASK_LOW::	.ASCIC a!/ LOWa
	0A		
48 47 49 48 20 20 20 20 2F 21 00	13A6	509 FMT_ASK_HIGH::	.ASCIC a!/ HIGHa
	0A		
	3F 00	510 ASK::	.ASCIC a?a
	01		
53 59 41 52 52 41 20 4F 4E 2F 21 00	13B3	511 FMT_NO_ARRAYS::	.ASCIC a!/NO ARRAYS AVAILABLE FOR TEST. SKIPPING TESTa
46 20 45 4C 42 41 4C 49 41 56 41 20	13BF		
49 4B 53 20 2E 54 53 45 54 20 52 4F	13CB		
	54 53 45 54 20 47 4E 49 50 50		
	2D		
45 42 4D 55 4E 20 4D 4F 52 2F 21 00	13E1	512 FMT_ROM_1::	.ASCIC a!/ROM NUMBER: !AC DEVICE TYPE: !ACa ; A2
49 56 45 44 20 20 43 41 21 20 3A 52	13ED		
43 41 21 20 3A 45 50 59 54 20 45 43	13F9		
	23		
54 53 54 4F 4F 42 20 4F 4E 2F 21 00	1405	513 FMT_ROM_3::	.ASCIC a!/NO BOOTSTRAP ROMSa ; A2
	53 4D 4F 52 20 50 41 52		
	13		
	1419	514 ;*	
	1419	515 ;	This is used to print out the array under test in the moving inversions
	1419	516 ;	test. It is a mixture of FAO and the previously mentioned construct, which
	1419	517 ;	takes advantage of the WHICH_ARRAY subroutine.

20	3A	44	45	54	43	45	50	58	45	2F	21	1598	548		a!/EXPECTED: !XB(X)a-	: A2	
						29	58	28	42	58	21	15A4					
20	3A	44	45	56	49	45	43	45	52	2F	21	15AA	549		a!/RECEIVED: !XB(X)a-	: A2	
						29	58	28	42	58	21	15B6					
20	3A	52	4F	58	20	20	20	20	20	2F	21	15BC	550		a!/ XOR: !XB(X)a	: A2	
						29	58	28	42	58	21	15C8					
											54	1579					
												15CE	551				
43	20	45	4E	49	48	43	41	4D	2F	21	00	15CE	552	FMT_CPU_4::	.ASCIC	a!/MACHINE CHECK ERROR SUMMARY REG INCORRECTa-	: A2
53	20	52	4F	52	52	45	20	4B	43	45	48	15DA					
49	20	47	45	52	20	59	52	41	4D	4D	55	15E6					
				54	43	45	52	52	4F	43	4E	15F2					
20	3A	44	45	54	43	45	50	58	45	2F	21	15FA	553		a!/EXPECTED: !XB(X)a-	: A2	
						29	58	28	42	58	21	1606					
20	3A	44	45	56	49	45	43	45	52	2F	21	160C	554		a!/RECEIVED: !XB(X)a-	: A2	
						29	58	28	42	58	21	1618					
20	3A	52	4F	58	20	20	20	20	20	2F	21	161E	555		a!/ XOR: !XB(X)a	: A2	
						29	58	28	42	58	21	162A					
											61	15CE					
												1630	556				
4B	43	45	48	43	20	4D	4F	52	2F	21	00	1630	557	FMT_ROM_2::	.ASCIC	a!/ROM CHECKSUM INCORRECTa-	: A2
43	45	52	52	4F	43	4E	49	20	4D	55	53	163C					
											54	1648					
52	45	42	4D	55	4E	20	4D	4F	52	2F	21	1649	558		a!/ROM NUMBER: !ACa-	: A2	
20	20	20	20	20	20	20	20	20	20	20	3A	1655					
											43	41	21	20	1661		
49	20	4D	55	53	4B	43	45	48	43	2F	21	1665	559		a!/CHECKSUM IN ROM: !XB(X)a-	: A2	
20	20	20	20	20	20	3A	4D	4F	52	20	4E	1671					
						29	58	28	42	58	21	20	167D				
44	45	54	41	4C	55	43	4C	41	43	2F	21	1684	560		a!/CALCULATED CHECKSUM: !XB(X)a	: A2	
20	20	3A	4D	55	53	4B	43	45	48	43	20	1690					
						29	58	28	42	58	21	20	169C				
											72	1630					
												16A3	561				
50	41	52	54	53	54	4F	4F	42	2F	21	00	16A3	562	FMT_ROM_4::	.ASCIC	a!/BOOTSTRAP ROM ADDRESS DID NOT RESPONDa-	: A2
53	53	45	52	44	44	41	20	4D	4F	52	20	16AF					
53	45	52	20	54	4F	4E	20	44	49	44	20	16BB					
											44	4E	4F	50	16C7		
53	45	52	44	44	41	20	4D	4F	52	2F	21	16CB	563		a!/ROM ADDRESS: !XL(X)a	: A2	
						29	58	28	4C	58	21	20	3A	53	16D7		
												3C	16A3				
												16E0	564				
45	48	43	20	47	4E	4F	52	57	2F	21	00	16E0	565	FMT_ECC_1::	.ASCIC	a!/WRONG CHECKBITS IN CSR1a-	
53	43	20	4E	49	20	53	54	49	42	4B	43	16EC					
											31	52	16F8				
20	3A	44	45	54	43	45	50	58	45	2F	21	16FA	566		a!/EXPECTED: !XL(X)a-		
						29	58	28	4C	58	21	1706					
20	3A	44	45	56	49	45	43	45	52	2F	21	170C	567		a!/RECEIVED: !XL(X)a-		
						29	58	28	4C	58	21	1718					
20	3A	52	4F	58	20	20	20	20	20	2F	21	171E	568		a!/ XOR: !XL(X)a		
						29	58	28	4C	58	21	172A					
											4F	16E0					
												1730	569				
45	4C	49	41	46	20	43	43	45	2F	21	00	1730	570	FMT_ECC_2::	.ASCIC	a!/ECC FAILED TO CORRECT SINGLE-BIT ERRORa-	
54	43	45	52	52	4F	43	20	4F	54	20	44	173C					
20	54	49	42	2D	45	4C	47	4E	49	53	20	1748					
							52	4F	52	52	45	1754					

ZZ-ECKAM-3.2
ECKAM
03-02

DECLARATIONS

MEMORY DIAGNOSTIC HEADER
DECLARATIONS

M 3

13-MAY-1987

Fiche 1 Frame M3

Sequence 38

23-JAN-1987

14:30:12

VAX/VMS Macro V04-00

Page 16

23-JAN-1987

14:26:49

[ZECCHINO.ECKAM]ECKAM0.MAR;4

(3)

20	3A	44	45	54	43	45	50	58	45	2F	21	1759	571	
						29	58	28	4C	58	21	1765		
20	3A	44	45	56	49	45	43	45	52	2F	21	176B	572	
						29	58	28	4C	58	21	1777		
20	3A	52	4F	58	20	20	20	20	20	2F	21	177D	573	
						29	58	28	4C	58	21	1789		
											5E	1730		
												178F	574	
54	49	52	57	20	47	4E	4F	4C	2F	21	00	178F	575	
45	4C	42	55	4F	44	20	44	4E	41	20	45	179B	FMT_ECC_3::	
46	20	52	4F	52	52	45	20	54	49	42	2D	17A7	.ASCIC	
							44	45	4C	49	41	17B3		
48	54	20	45	54	49	52	57	20	4F	54	20	17B8	576	
						59	52	4F	4D	45	4D	20	17C4	
20	3A	44	45	54	43	45	50	58	45	2F	21	17CC	577	
						29	58	28	4C	58	21	17D8		
20	3A	44	45	56	49	45	43	45	52	2F	21	17DE	578	
						29	58	28	4C	58	21	17EA		
20	3A	52	4F	58	20	20	20	20	20	2F	21	17F0	579	
						29	58	28	4C	58	21	17FC		
											72	178F		
												1802	580	
4D	45	54	54	41	20	43	43	45	2F	21	00	1802	581	
52	52	4F	43	20	4F	54	20	44	45	54	50	180E	FMT_ECC_4::	
42	2D	45	4C	42	55	4F	44	20	54	43	45	181A	.ASCIC	
						52	4F	52	52	45	20	54	49	1826
20	3A	44	45	54	43	45	50	58	45	2F	21	182E	582	
						29	58	28	4C	58	21	183A		
20	3A	44	45	56	49	45	43	45	52	2F	21	1840	583	
						29	58	28	4C	58	21	184C		
20	3A	52	4F	58	20	20	20	20	20	2F	21	1852	584	
						29	58	28	4C	58	21	185E		
											61	1802		
												1864	585	
54	43	45	52	52	4F	43	4E	55	2F	21	00	1864	586	
46	20	52	4F	52	52	45	20	45	4C	42	41	1870	FMT_ECC_5::	
20	54	45	53	20	54	4F	4E	20	47	41	4C	187C	.ASCIC	
						30	52	53	43	20	4E	49	1888	
20	3A	44	45	54	43	45	50	58	45	2F	21	188F	587	
						29	58	28	4C	58	21	189B		
20	3A	44	45	56	49	45	43	45	52	2F	21	18A1	588	
						29	58	28	4C	58	21	18AD		
20	3A	52	4F	58	20	20	20	20	20	2F	21	18B3	589	
						29	58	28	4C	58	21	18BF		
											60	1864		
												18C5	590	
												18C5	591	
4D	45	54	54	41	20	43	43	45	2F	21	00	18C5	592	
52	52	4F	43	20	4F	54	20	44	45	50	54	18D1	FMT_ECC_6::	
52	52	45	20	54	53	4F	4C	20	54	43	45	18DD	.ASCIC	
											52	4F	18E9	
20	3A	44	45	54	43	45	50	58	45	2F	21	18EB	593	
						29	58	28	4C	58	21	18F7		
20	3A	44	45	56	49	45	43	45	52	2F	21	18FD	594	
						29	58	28	4C	58	21	1909		
20	3A	52	4F	58	20	20	20	20	20	2F	21	190F	595	
						29	58	28	4C	58	21	191B		

a!/EXPECTED: !XL(X)a-

a!/RECEIVED: !XL(X)a-

a!/ XOR: !XL(X)a

a!/LONG WRITE AND DOUBLE-BIT ERROR FAILEDa-

; M2

a TO WRITE THE MEMORYa-

; A2

a!/EXPECTED: !XL(X)a-

a!/RECEIVED: !XL(X)a-

a!/ XOR: !XL(X)a

a!/ECC ATTEMPTED TO CORRECT DOUBLE-BIT ERRORa-

a!/EXPECTED: !XL(X)a-

a!/RECEIVED: !XL(X)a-

a!/ XOR: !XL(X)a

a!/UNCORRECTABLE ERROR FLAG NOT SET IN CSR0a-

a!/EXPECTED: !XL(X)a-

a!/RECEIVED: !XL(X)a-

a!/ XOR: !XL(X)a

a!/ECC ATTEMPTED TO CORRECT LOST ERRORa-

a!/EXPECTED: !XL(X)a-

a!/RECEIVED: !XL(X)a-

a!/ XOR: !XL(X)a

MEMORY DIAGNOSTIC HEADER
DECLARATIONS

											5B	18C5					
55	20	44	4E	41	20	45	43	55	2F	21	00	1921	596				
53	4F	4C	20	4F	46	4E	49	20	2C	45	43	1921	597				
4E	49	20	54	45	53	20	54	4F	4E	20	54	1921	598	FMT_ECC_7::	.ASCIC	a!/UCE AND UCE, INFO LOST NOT SET IN CSR0a-	
							30	52	53	43	20	1945					
20	3A	44	45	54	43	45	50	58	45	2F	21	194A	599			a!/EXPECTED: !XL(X)a-	
							29	58	28	4C	58	21	1956				
20	3A	44	45	56	49	45	43	45	52	2F	21	195C	600			a!/RECEIVED: !XL(X)a-	
							29	58	28	4C	58	21	1968				
20	3A	52	4F	58	20	20	20	20	20	2F	21	196E	601			a!/ XOR: !XL(X)a	
							29	58	28	4C	58	21	197A				
												5E	1921				
												1980	602				
49	54	53	4F	4E	47	41	49	44	2F	21	00	1980	603	FMT_ECC_8::	.ASCIC	a!/DIAGNOSTIC BITS STUCK AT ONE IN CSR1a-	
4B	43	55	54	53	20	53	54	49	42	20	43	198C					
43	20	4E	49	20	45	4E	4F	20	54	41	20	1998					
									31	52	53	19A4					
20	3A	44	45	54	43	45	50	58	45	2F	21	19A7	604			a!/EXPECTED: !XL(X)a-	
							29	58	28	4C	58	21	19B3				
20	3A	44	45	56	49	45	43	45	52	2F	21	19B9	605			a!/RECEIVED: !XL(X)a-	
							29	58	28	4C	58	21	19C5				
20	3A	52	4F	58	20	20	20	20	20	2F	21	19CB	606			a!/ XOR: !XL(X)a	
							29	58	28	4C	58	21	19D7				
												5C	1980				
												19DD	607				
49	54	53	4F	4E	47	41	49	44	2F	21	00	19DD	608	FMT_ECC_9::	.ASCIC	a!/DIAGNOSTIC CHECK MODE OR PAGE MODE STUCK AT ZEROa-	
45	44	4F	4D	20	4B	43	45	48	43	20	43	19E9					
44	4F	4D	20	45	47	41	50	20	52	4F	20	19F5					
5A	20	54	41	20	4B	43	55	54	53	20	45	1A01					
									4F	52	45	1A0D					
							31	52	53	43	20	4E	49	20	1A10	609	a IN CSR1a-
20	3A	44	45	54	43	45	50	58	45	2F	21	1A18	610			a!/EXPECTED: !XL(X)a-	
							29	58	28	4C	58	21	1A24				
20	3A	44	45	56	49	45	43	45	52	2F	21	1A2A	611			a!/RECEIVED: !XL(X)a-	
							29	58	28	4C	58	21	1A36				
20	3A	52	4F	58	20	20	20	20	20	2F	21	1A3C	612			a!/ XOR: !XL(X)a	
							29	58	28	4C	58	21	1A48				
												70	19DD				
												1A4E	613				
42	41	53	49	44	20	43	43	45	2F	21	00	1A4E	614	FMT_ECC_10::	.ASCIC	a!/ECC DISABLE MODE OR PAGE MODE STUCK AT ZEROa-	
50	20	52	4F	20	45	44	4F	4D	20	45	4C	1A5A					
55	54	53	20	45	44	4F	4D	20	45	47	41	1A66					
		4F	52	45	5A	20	54	41	20	4B	43	1A72					
							31	52	53	43	20	4E	49	20	1A7C	615	a IN CSR1a-
20	3A	44	45	54	43	45	50	58	45	2F	21	1A84	616			a!/EXPECTED: !XL(X)a-	
							29	58	28	4C	58	21	1A90				
20	3A	44	45	56	49	45	43	45	52	2F	21	1A96	617			a!/RECEIVED: !XL(X)a-	
							29	58	28	4C	58	21	1AA2				
20	3A	52	4F	58	20	20	20	20	20	2F	21	1AA8	618			a!/ XOR: !XL(X)a	
							29	58	28	4C	58	21	1AB4				
												6B	1A4E				
												1ABA	619				
45	4C	4C	4F	52	54	4E	4F	43	2F	21	00	1ABA	620	FMT_ECC_11::	.ASCIC	a!/CONTROLLER STUCK IN PAGE MODEa	
50	20	4E	49	20	4B	43	55	54	53	20	52	1AC6					
							45	44	4F	4D	20	45	47	41	1AD2		

ZZ-ECKAM-3.2
ECKAM
03-02

DECLARATIONS

MEMORY DIAGNOSTIC HEADER
DECLARATIONS

C 4

13-MAY-1987

Fiche 1 Frame C4

Sequence 41

23-JAN-1987 14:30:12

VAX/VMS Macro V04-00

Page 19

23-JAN-1987 14:26:49

[ZECCHINO.ECKAM]ECKAM0.MAR;4

(3)

41 54 43 45 52 52 4F 43 4E 55 20 44 1C89
43 4F 20 52 4F 52 52 45 20 45 4C 42 1C95
44 45 52 52 55 43 1CA1
45 52 20 53 53 45 52 44 44 41 2F 21 1CA7
58 21 20 3A 44 45 43 4E 45 52 45 46 1CB3
29 58 28 4C 1CBF
45 1C7D
58 28 4C 58 21 20 43 41 21 2F 21 00' 1CC3
58 28 4C 58 21 20 43 41 21 2F 21 29 1CCF
29 1CDB
18 1CC3
58 28 4C 58 21 20 43 41 21 2F 21 00' 1CDC
58 28 4C 58 21 20 43 41 21 2F 21 29 1CE8
58 28 4C 58 21 20 43 41 21 2F 21 29 1CF4
29 1D00
24 1CDC
53 20 4C 4F 52 54 4E 4F 43 2F 21 00' 1D01
55 52 52 45 54 4E 49 20 45 52 4F 54 1D0D
54 50 1D19
19 1D01
42 20 53 53 45 52 44 44 41 2F 21 00' 1D1B
53 20 42 55 21 20 54 49 42 20 53 55 1D27
4B 43 55 54 1D33
1B 1D1B
20 53 55 42 20 41 54 41 44 2F 21 00' 1D37
20 42 55 21 20 54 49 42 1D43
13 1D37
54 43 45 52 52 4F 43 4E 55 2F 21 00' 1D4B
20 2C 52 4F 52 52 45 20 45 4C 42 41 1D57
20 4E 4F 49 54 41 4D 52 4F 46 4E 49 1D63
54 53 4F 4C 1D6F
27 1D4B
52 41 50 20 45 48 43 41 43 2F 21 00' 1D73
52 4F 52 52 45 20 59 54 49 1D7F
14 1D73
43 45 4C 45 53 20 57 4F 52 2F 21 00' 1D88
55 21 20 57 4F 52 20 53 55 42 20 54 1D94
4B 43 55 54 53 20 42 1DA0
1E 1D88
4F 50 50 55 53 20 54 4F 4E 2F 21 00' 1DA7
45 48 20 45 42 20 4F 54 20 44 45 53 1DB3
45 52 1DBF
19 1DA7
54 53 49 58 45 2D 4E 4F 4E 2F 21 00' 1DC1
46 20 59 52 4F 4D 45 4D 20 54 4E 45 1DCD
54 4C 55 41 1DD9
1B 1DC1
49 42 20 45 4C 42 55 4F 44 2F 21 00' 1DDD
41 20 54 41 20 52 4F 52 52 45 20 54 1DE9
28 4C 58 21 20 3A 53 53 45 52 44 44 1DF5
29 58 1E01
25 1DDD
49 20 45 52 55 4C 49 41 46 2F 21 00' 1E03
43 20 2C 42 55 21 20 57 4F 52 20 4E 1E0F
42 55 21 20 4E 4D 55 4C 4F 1E1B
20 1E03
45 5A 20 54 41 20 4B 43 55 54 53 00' 1E24

643

a!/ADDRESS REFERENCED: !XL(X)a

; A5

644 FMT_BODY::

.ASCIC a!/!AC !XL(X)!/!AC !XL(X)a

645 FMT_COMP_ERR::

.ASCIC a!/!AC !XL(X)!/!AC !XL(X)!/!AC !XL(X)a

646 FMT_CS_MC::

.ASCIC a!/CONTROL STORE INTERRUPTa

647 FMT_ADR_BUS::

.ASCIC a!/ADDRESS BUS BIT !UB STUCKa

648 FMT_DATA_BUS::

.ASCIC a!/DATA BUS BIT !UB a

649 FMT_INF_LST::

.ASCIC a!/UNCORRECTABLE ERROR, INFORMATION LOSTa

650 FMT_CPE_MC::

.ASCIC a!/CACHE PARITY ERRORa

651 FMT_CS_BUS::

.ASCIC a!/ROW SELECT BUS ROW !UB STUCKa

652 FMT_NSBH::

.ASCIC a!/NOT SUPPOSED TO BE HEREa

653 FMT_NX_MEM::

.ASCIC a!/NON-EXISTENT MEMORY FAULTa

654 FMT_RAM::

.ASCIC a!/DOUBLE BIT ERROR AT ADDRESS: !XL(X)a

; M2

655 FMT_ROW::

.ASCIC a!/FAILURE IN ROW !UB, COLUMN !UBa

656 FMT_SA0::

.ASCIC aSTUCK AT ZEROa

4E 4F 20 54 41 20 4B 43 55 54 53 00	4F 52 1E30		
	0D 1E24		
	1E32	657 FMT_SA1::	.ASCIC aSTUCK AT ONEa
	45 1E3E		
	0C 1E32		
20 4F 54 20 44 45 54 52 4F 48 53 00	1E3F	658 FMT_SHORT::	.ASCIC aSHORTED TO BIT !UBa
	42 55 21 20 54 49 42 1E4B		
	12 1E3F		
49 53 20 59 52 4F 4D 45 4D 2F 21 00	1E52	659 FMT_SIZ_ERR::	.ASCIC a!/MEMORY SIZING ERROR:!/a-
21 3A 52 4F 52 52 45 20 47 4E 49 5A	1E5E		
	2F 1E6A		
54 53 45 48 47 49 48 20 32 52 53 43	1E6B	660	aCSR2 HIGHEST ADDRESS IS: !XL(X)!/HIGHEST ADDRESS a-
3A 53 49 20 53 53 45 52 44 44 41 20	1E77		
49 48 2F 21 29 58 28 4C 58 21 20 20	1E83		
53 45 52 44 44 41 20 54 53 45 48 47	1E8F		
	20 53 1E9B		
58 21 20 3A 53 49 20 44 4E 55 4F 46	1E9D	661	aFOUND IS: !XL(X)a
	29 58 28 4C 1EA9		
	5A 1E52		
44 45 54 50 55 52 52 4F 43 2F 21 00	1EAD	662 FMT_BAD_LOC::	.ASCIC a!/CORRUPTED TEST LOCATION FOUND IN LOWER 256KB.!/a-
49 54 41 43 4F 4C 20 54 53 45 54 20	1EB9		
20 4E 49 20 44 4E 55 4F 46 20 4E 4F	1EC5		
2E 42 4B 36 35 32 20 52 45 57 4F 4C	1ED1		
	2F 21 1EDD		
43 4F 20 52 4F 52 52 45 20 53 44 52	1EDF	663	aRDS ERROR OCCURED WHILE TRYING TO FORCE A CRD.!/a-
20 45 4C 49 48 57 20 44 45 52 55 43	1EEB		
4F 46 20 4F 54 20 47 4E 49 59 52 54	1EF7		
2F 21 2E 44 52 43 20 41 20 45 43 52	1F03		
45 52 50 20 45 4C 42 49 53 53 4F 50	1F0F	664	aPOSSIBLE PRE-EXISTING CRD ERROR IN TEST LOCATION!/a-
52 43 20 47 4E 49 54 53 49 58 45 2D	1F1B		
54 20 4E 49 20 52 4F 52 52 45 20 44	1F27		
4E 4F 49 54 41 43 4F 4C 20 54 53 45	1F33		
	2F 21 1F3F		
41 20 4C 4C 49 57 20 54 53 45 54 28	1F41	665	a(TEST WILL ABORT).!/a-
	2F 21 2E 29 54 52 4F 42 1F4D		
45 46 45 52 20 53 53 45 52 44 44 41	1F55	666	aADDRESS REFERENCED:: !XL(X)a
4C 58 21 20 3A 3A 44 45 43 4E 45 52	1F61		
	29 58 28 1F6D		
	C2 1EAD		
2A 2A 2A 2A 2A 2A 2A 2A 2A 2F 21 00	1F70	667 FMT_WARN_MESS::	.ASCIC a!/*****!/a-
2A	1F7C		
2A	1F88		
2A	1F94		
	2F 21 2A 2A 2A 1FA0		
41 49 44 20 3A 47 4E 49 4E 52 41 57	1FA5	668	aWARNING: DIAGNOSTIC DEFAULTS TO AN EXHAUSTIVE RUN.!/a-
41 46 45 44 20 43 49 54 53 4F 4E 47	1FB1		
45 20 4E 41 20 4F 54 20 53 54 4C 55	1FBD		
55 52 20 45 56 49 54 53 55 41 48 58	1FC9		
	2F 21 2E 4E 1FD5		
20 4B 43 49 55 51 20 41 20 52 4F 46	1FD9	669	aFOR A QUICK VERIFY OF THE MEMORY, TYPE CTRL C AND!/a-
48 54 20 46 4F 20 59 46 49 52 45 56	1FES		
59 54 20 2C 59 52 4F 4D 45 4D 20 45	1FF1		
4E 41 20 43 20 4C 52 54 43 20 45 50	1FFD		
	2F 21 44 2009		
45 52 20 4E 45 48 57 20 2E 4F 42 41	200C	670	aABO. WHEN RETURNED TO DS PROMPT, SET FLAG QUICK!/a-
53 44 20 4F 54 20 44 45 4E 52 55 54	2018		
54 45 53 20 2C 54 50 4D 4F 52 50 20	2024		

21 4B 43 49 55 51 20 47 41 4C 46 20	2030		
	2F 203C		
20 54 52 41 54 53 45 52 20 44 4E 41	203D	671	aAND RESTART THE PROGRAM.!/a
2E 4D 41 52 47 4F 52 50 20 45 48 54	2049		
	2F 21 2055		
	E6 1F70		
20 2C 4F 53 4C 41 20 45 54 4F 4E 00	2057	672	FMT_WARN_MES2:: .ASCIC aNOTE ALSO, ONLY MEMORY ABOVE THE DIAGNOSTIC!/a-
20 59 52 4F 4D 45 4D 20 59 4C 4E 4F	2063		
49 44 20 45 48 54 20 45 56 4F 42 41	206F		
	2F 21 43 49 54 53 4F 4E 47 41		
49 20 52 4F 53 49 56 52 45 50 55 53	207B	673	aSUPERVISOR IS TESTED. ARRAY + ROW UNDER TEST!/a-
52 41 20 2E 44 45 54 53 45 54 20 53	2085		
4E 55 20 57 4F 52 20 2B 20 59 41 52	2091		
	2F 21 54 53 45 54 20 52 45 44		
4E 49 20 53 54 55 4F 54 4E 49 52 50	209D	674	aPRINTOUTS IN TESTS 10, 11, AND 12 WILL INDICATE!/a-
31 20 2C 30 31 20 53 54 53 45 54 20	20A9		
49 57 20 32 31 20 44 4E 41 20 2C 31	20B3		
21 45 54 41 43 49 44 4E 49 20 4C 4C	20BF		
	2F 21 2A 20CB		
	D5 20D7		
	2F 20E3		
47 4E 49 45 42 20 53 27 54 41 48 57	20E4	675	aWHAT'S BEING TESTED.!/a-
	2F 21 2E 44 45 54 53 45 54 20		
2A	20F0	676	a*****!/a
2A	20FA		
2A	2106		
2A	2112		
	2F 21 2A 211E		
	D5 212A		
	2F 21 00 212D	677	FMT_WBE_MC:: .ASCIC a!/WRITE BUS ERRORa
53 55 42 20 45 54 49 52 57 2F 21 00	2139		
	11 212D		
43 20 4C 41 47 45 4C 4C 49 2F 21 00	213F	678	FMT_ILL_CFG:: .ASCIC a!/ILLEGAL CONFIGURATION IN MEMORY MAP!/a-
4E 4F 49 54 41 52 55 47 49 46 4E 4F	214B		
4D 20 59 52 4F 4D 45 4D 20 4E 49 20	2157		
	2F 21 50 41 2163		
20 46 4F 20 53 54 4E 45 54 4E 4F 43	2167	679	aCONTENTS OF MEMORY MAP: !XL(X)a
20 3A 50 41 4D 20 59 52 4F 4D 45 4D	2173		
	29 58 28 4C 58 21 217F		
	45 213F		
41 4D 20 59 52 4F 4D 45 4D 2F 21 00	2185	680	FMT_REG_ERR:: .ASCIC a!/MEMORY MAP REGISTER (CSR2<15:0>) INVALID!/a-
28 20 52 45 54 53 49 47 45 52 20 50	2191		
20 29 3E 30 3A 35 31 3C 32 52 53 43	219D		
	2F 21 44 49 4C 41 56 4E 49 21A9		
20 46 4F 20 53 54 4E 45 54 4E 4F 43	21B2	681	aCONTENTS OF MEMORY MAP: !XL(X)a
20 3A 50 41 4D 20 59 52 4F 4D 45 4D	21BE		
	29 58 28 4C 58 21 21CA		
	4A 2185		
20 3A 44 45 56 49 45 43 45 52 20 00	21D0	682	REC_MSG:: .ASCIC a RECEIVED: a
	0B 21D0		
20 3A 44 45 54 43 45 50 58 45 20 00	21DC	683	EXP_MSG:: .ASCIC a EXPECTED: a
	0B 21DC		
20 3A 52 4F 58 20 20 20 20 20 20 00	21E8	684	XOR_MSG:: .ASCIC a XOR: a
	0B 21E8		
	21F4	685	
46 4F 20 52 45 42 4D 55 4E 2F 21 00	21F4	686	FMT_SUMMARY1: .ASCIC a!/NUMBER OF CORRECTABLE ERRORS = !UL!/a-
45 4C 42 41 54 43 45 52 52 4F 43 20	2200		
55 21 20 3D 20 53 52 4F 52 52 45 20	220C		
	2F 21 4C 2218		

ZZ-ECKAM-3.2
ECKAM
03-02

DECLARATIONS

MEMORY DIAGNOSTIC HEADER
DECLARATIONS

F 4

13-MAY-1987

Fiche 1 Frame F4

Sequence 44

23-JAN-1987 14:30:12

VAX/VMS Macro V04-00

Page 22

23-JAN-1987 14:26:49

[ZECCHINO.ECKAM]ECKAM0.MAR;4

(3)

50	28	20	53	53	45	52	44	44	41	2F	21	221B	687		
20	20	20	20	20	20	20	20	29	45	47	41	2227			
53	2F	52	45	42	4D	55	4E	20	54	49	42	2233			
			2F	21	45	4D	4F	52	44	4E	59	223F			
2D	2248	688													
2D	2D	20	20	20	20	20	20	20	20	2D	2D	2254			
2D	2260														
			2F	21	2F	21	2D	2D	2D	2D	2D	226C			
											80	21F4			
20	20	20	29	58	28	4C	58	21	20	20	00	2275	689 FMT_SUMMARY2:		
58	21	20	20	20	20	20	20	20	20	20	20	2281	.ASCIC a !XL(X)		
4F	52	44	4E	59	53	28	20	29	58	28	42	228D	!XB(X) (SYNDROME)!/a		
						2F	21	29	45	4D	2299				
											28	2275			
52	45	57	20	45	52	45	48	54	2F	21	00	229E	690 FMT_SUMMARY3:		
54	43	45	52	52	4F	43	20	4F	4E	20	45	22AA	.ASCIC a!/THERE WERE NO CORRECTABLE ERRORS!/a		
21	53	52	4F	52	52	45	20	45	4C	42	41	22B6			
											2F	22C2			
											24	229E			
20	20	20	29	58	28	4C	58	21	20	20	00	22C3	691 FMT_SUMMARY4:		
55	21	20	20	20	20	20	20	20	20	20	20	22CF	.ASCIC a !XL(X)		
42	20	41	54	41	44	28	20	29	44	28	42	22DB	!UB(D) (DATA BIT)!/a		
						2F	21	29	54	49	22E7				
											28	22C3			
20	20	20	29	58	28	4C	58	21	20	20	00	22EC	692 FMT_SUMMARY5:		
55	21	20	20	20	20	20	20	20	20	20	20	22F8	.ASCIC a !XL(X)		
20	4B	43	45	48	43	28	20	29	44	28	42	2304	!UB(D) (CHECK BIT)!/a		
						2F	21	29	54	49	42	2310			
											29	22EC			
2D	2F	21	00	2316	693 FMT_SUMMARY6:										
2D	2322	.ASCIC a!/-----!//a-													
2D	232E														
2F	21	2D	233A												
											2F	21	2346		
56	20	22	20	4C	41	55	54	43	41	20	20	2348	694		
41	43	20	54	41	48	54	20	20	22	20	41	2354	a ACTUAL " VA " THAT CAUSED CORRECTABLE ERROR!//a-		
54	43	45	52	52	4F	43	20	44	45	53	55	2360			
2F	21	52	4F	52	52	45	20	45	4C	42	41	236C			
											2F	21	2378		
2D	2F	21	237A	695											
2D	2386	a!/-----!//a													
2D	2392														
2F	21	2D	239E												
											93	2316			
			2F	21	29	58	28	4C	58	21	20	20	00	23AA	696 FMT_SUMMARY7:
											0A	23AA	.ASCIC a !XL(X)!/a		
20	52	4F	52	52	45	20	20	20	2F	21	00	23B5	697 FMT_LOG_FULL::		
		2F	21	4C	4C	55	46	20	47	4F	4C	23C1	.ASCIC a!/ ERROR LOG FULL!/a		
											15	23B5			
20	53	54	4E	45	54	4E	4F	43	2F	21	00	23CB	698 FMT_MEM_REG::		
52	4F	4D	45	4D	20	45	48	54	20	46	4F	23D7	.ASCIC a!/CONTENTS OF THE MEMORY CSR'Sa-		
					53	27	52	53	43	20	59	23E3			
21	20	2D	20	30	52	53	43	20	20	2F	21	23EA	699		
43	41	21	20	3B	20	20	29	58	28	4C	58	23F6	a!/ CSR0 - !XL(X) ; !ACa-		
21	20	2D	20	31	52	53	43	20	20	2F	21	2402	700		
43	41	21	20	3B	20	20	29	58	28	4C	58	240E	a!/ CSR1 - !XL(X) ; !ACa-		
21	20	2D	20	32	52	53	43	20	20	2F	21	241A	701		
													a!/ CSR2 - !XL(X) ; !AC!/a		

a!/ADDRESS (PAGE) BIT NUMBER/SYNDROME!/a-

a-----!//a

689 FMT_SUMMARY2: .ASCIC a !XL(X) !XB(X) (SYNDROME)!/a

690 FMT_SUMMARY3: .ASCIC a!/THERE WERE NO CORRECTABLE ERRORS!/a

691 FMT_SUMMARY4: .ASCIC a !XL(X) !UB(D) (DATA BIT)!/a

692 FMT_SUMMARY5: .ASCIC a !XL(X) !UB(D) (CHECK BIT)!/a

693 FMT_SUMMARY6: .ASCIC a!/-----!//a-

694 a ACTUAL " VA " THAT CAUSED CORRECTABLE ERROR!//a-

695 a!/-----!//a

696 FMT_SUMMARY7: .ASCIC a !XL(X)!/a

697 FMT_LOG_FULL:: .ASCIC a!/ ERROR LOG FULL!/a

698 FMT_MEM_REG:: .ASCIC a!/CONTENTS OF THE MEMORY CSR'Sa-

699 a!/ CSR0 - !XL(X) ; !ACa-

700 a!/ CSR1 - !XL(X) ; !ACa-

701 a!/ CSR2 - !XL(X) ; !AC!/a

```

43 41 21 20 3B 20 20 29 58 28 4C 58 2426
      2F 21 2432
      68 23CB
      2434 702
43 20 45 4E 49 48 43 41 4D 2F 21 00' 2434 703 FMT_MCH_STACK1::.ASCIC a!/MACHINE CHECK STACKa-
      4B 43 41 54 53 20 4B 43 45 48 2440
3B 09 29 58 28 4C 58 21 20 20 2F 21 244A 704 a!/ !XL(X) ; LENGTHa-
      48 54 47 4E 45 4C 20 2456
3B 09 29 58 28 4C 58 21 20 20 2F 21 245D 705 a!/ !XL(X) ; ERROR CODEa-
      45 44 4F 43 20 52 4F 52 52 45 20 2469
3B 09 29 58 28 4C 58 21 20 20 2F 21 2474 706 a!/ !XL(X) ; VA REGISTERa-
52 45 54 53 49 47 45 52 20 41 56 20 2480
3B 09 29 58 28 4C 58 21 20 20 2F 21 248C 707 a!/ !XL(X) ; PC AT TIME OF ERRORa-
20 45 4D 49 54 20 54 41 20 43 50 20 2498
      52 4F 52 52 45 20 46 4F 24A4
3B 09 29 58 28 4C 58 21 20 20 2F 21 24AC 708 a!/ !XL(X) ; MDRa-
      52 44 4D 20 24B8
3B 09 29 58 28 4C 58 21 20 20 2F 21 24BC 709 a!/ !XL(X) ; SAVED MODEa-
      45 44 4F 4D 20 44 45 56 41 53 20 24C8
3B 09 29 58 28 4C 58 21 20 20 2F 21 24D3 710 a!/ !XL(X) ; READ LOCK TIMEOUTa
54 20 4B 43 4F 4C 20 44 41 45 52 20 24DF
      54 55 4F 45 4D 49 24EB
      BC 2434
09 29 58 28 4C 58 21 20 20 2F 21 00' 24F1 711 FMT_MCH_STACK2::.ASCIC a!/ !XL(X) ; TB ERROR REGa-
20 52 4F 52 52 45 20 20 42 54 20 3B 24FD
      47 45 52 2509
3B 09 29 58 28 4C 58 21 20 20 2F 21 250C 712 a!/ !XL(X) ; CACHE ERROR REGa-
52 4F 52 52 45 20 45 48 43 41 43 20 2518
      47 45 52 20 2524
3B 09 29 58 28 4C 58 21 20 20 2F 21 2528 713 a!/ !XL(X) ; BUS ERROR REGa-
52 20 52 4F 52 52 45 20 53 55 42 20 2534
      47 45 2540
3B 09 29 58 28 4C 58 21 20 20 2F 21 2542 714 a!/ !XL(X) ; ERROR SUMMARY REGa-
41 4D 4D 55 53 20 52 4F 52 52 45 20 254E
      47 45 52 20 59 52 255A
3B 09 29 58 28 4C 58 21 20 20 2F 21 2560 715 a!/ !XL(X) ; PCa-
      43 50 20 256C
3B 09 29 58 28 4C 58 21 20 20 2F 21 256F 716 a!/ !XL(X) ; PSL!/a
      2F 21 4C 53 50 20 257B
      8F 24F1
      2581 717
42 55 21 20 59 41 52 52 41 2F 21 00' 2581 718 FMT_SGL_BIT_ERR::.ASCIC a!/ARRAY !UB, ROW !UB, HAS !UW SINGLE BIT ERRORS!/a
48 20 2C 42 55 21 20 57 4F 52 20 2C 258D
4C 47 4E 49 53 20 57 55 21 20 53 41 2599
53 52 4F 52 52 45 20 54 49 42 20 45 25A5
      2F 21 25B1
      31 2581
      25B3 719

```

```

00000000 721 .SBTTL SUMMARY MODULE
00000000 722 .PSECT ROUTINES,PAGE,EXE,NOWRT
00000000 723
00000000 724 $DS_BGNSUMMARY
00000000 .SAVE
00000000 .PSECT SUMMARY, LONG
00000000 SUMMARY:
00000000 .WORD ^M<> ; ENTRY MASK
0000043D'EF 95 0002 725 TSTB SUMMARY_FLAG ; Moving Inversions Executed? A9
03 12 0008 726 BNEQ 3$ ; Branch if yes A9
00DF 31 000A 727 BRW 60$
57 00000031'EF D0 000D 728 3$: MOVL PGE_COR_ERR,R7 ; Get the number of errors per page
58 00000439'EF D0 0014 729 MOVL TOTAL_COR_ERR,R8 ; Get total single bit errors
03 12 001B 730 BNEQ 5$ ; Branch if errors present
00BF 31 001D 731 BRW 50$
0020 732 5$: $DS_PRINTS_S FMT_SUMMARY1,R8 ; Print number of correctable
58 DD 0020 PUSHL R8
000021F4'EF 9F 0022 PUSHAB FMT_SUMMARY1
000100F8 9F 02 FB 0028 CALLS $$$N, a#DS$PRINTS
002F 733 ; Errors and Header
57 D7 002F 734 DECL R7 ; Make total page errors a loop count
52 D4 0031 735 CLRL R2 ; Init the index
FF0001FF 8F CB 0033 736 10$: BICL3 #AXFF0001FF,- ; Get the address from the table
53 00000C3C'EF42 0039 737 COR_ERRORS[R2],R3 ; ...
FFFFF80 8F CB 0040 738 BICL3 #AXFFFFFFF80,- ; Get the syndrome from the table
54 00000C3C'EF42 0046 739 COR_ERRORS[R2],R4 ; ...
004D 740 ;
004D 741 ; Convert syndrome to bit number.
004D 742 ;
55 D4 004D 743 CLRL R5 ; Init an index
00000604'EF45 54 91 004F 744 20$: CMPB R4,SYNDROME[R5] ; Check syndrome against table
2A 13 0057 745 BEQL 25$ ; Branch if this bit
F2 55 1F F3 0059 746 AOBLEQ #31,R5,20$ ; Check all 32 bits
005D 747 ;
005D 748 ; Not a data bit, check for a CHECK BIT failure
005D 749 ;
55 D4 005D 750 CLRL R5 ; Init an index
56 01 D0 005F 751 MOVL #1,R6 ; Init a check bit syndrome
54 56 91 0062 752 22$: CMPB R6,R4 ; Is it this syndrome?
30 13 0065 753 BEQL 30$ ; Branch if yes
56 56 01 78 0067 754 ASHL #1,R6,R6 ; Select next syndrome
F3 55 06 F3 006B 755 AOBLEQ #6,R5,22$ ; Check all 7 syndromes
006F 756 ;
006F 757 ; Undefined syndrome. Print the raw syndrome
006F 758 ;
006F 759 $DS_PRINTS_S FMT_SUMMARY2,R3,R4 ; Print the data
54 DD 006F PUSHL R4
53 DD 0071 PUSHL R3
00002275'EF 9F 0073 PUSHAB FMT_SUMMARY2
000100F8 9F 03 FB 0079 CALLS $$$N, a#DS$PRINTS
0025 31 0080 760 BRW 45$
0083 761 ;
0083 762 ; Data bit error, Print the bit in error.
0083 763 ;
0083 764 25$: $DS_PRINTS_S FMT_SUMMARY4,R3,R5 ; Print the failing data bit
55 DD 0083 PUSHL R5
53 DD 0085 PUSHL R3

```

```

000022C3'EF 9F 0087          PUSHAB  FMT_SUMMARY4
000100F8 9F 03  FB 008D          CALLS   $$$N, @DS$PRINTS
          0011 31 0094 765      BRW     45$
          0097 766      :
          0097 767      : Check bit error, Print the bit in error
          0097 768      :
          0097 769 30$:  $DS_PRINTS S  FMT_SUMMARY5,R3,R5 ; Print the failing check bit
          55  DD 0097          PUSHL   R5
          53  DD 0099          PUSHL   R3
          000022EC'EF 9F 009B          PUSHAB  FMT_SUMMARY5
000100F8 9F 03  FB 00A1          CALLS   $$$N, @DS$PRINTS
          FF85 52 01 57  F1 00A8 770
          00AE 771 45$:  ACBL   R7,#1,R2,10$          ; Print all the errors
          00AE 772      :
          00AE 773      : This is where the actual Virtual Address that caused the single bit error
          00AE 774      : is retrieved and printed from the log
          00AE 775      :
          00AE 776 47$:  $DS_PRINTS S  FMT_SUMMARY6          ; Print the Virtual Address message
          00002316'EF 9F 00AE          PUSHAB  FMT_SUMMARY6
000100F8 9F 01  FB 00B4          CALLS   $$$N, @DS$PRINTS
          52  D4 00BB 777      CLRL   R2          ; Init the index for VA log
          58  D7 00BD 778      DECL   R8          ; Decrement counter to allow for zero
53 00000035'EF42  D0 00BF 779 48$:  MOVL   VA_COR_ADDRESS[R2],R3 ; Get the VA from the log
          00C7 780      $DS_PRINTS S  FMT_SUMMARY7,R3 ; Print the actual Virtual Address
          53  DD 00C7          PUSHL   R3
          000023AA'EF 9F 00C9          PUSHAB  FMT_SUMMARY7
000100F8 9F 02  FB 00CF          CALLS   $$$N, @DS$PRINTS
          FFE3 52 01 58  F1 00D6 781      ACBL   R8,#1,R2,48$          ; Do for the whole log
          000D 31 00DC 782      BRW     60$          ; Exit
          00DF 783 50$:  $DS_PRINTS S  FMT_SUMMARY3          ; Print no errors message
          0000229E'EF 9F 00DF          PUSHAB  FMT_SUMMARY3
000100F8 9F 01  FB 00E5          CALLS   $$$N, @DS$PRINTS
          00EC 784 60$:
          00EC 785
          00EC 786      $DS_ENDSUMMARY
          00010058 9F 6E  FA 00EC          SUMMARY_X:  CALLG   (SP), @DS$BREAK
          04 00F3          RET          ; RETURN TO COMMAND MODE
          00000000          .RESTORE

```

```

0000 0000 788 .SBTTL PRINT SUBROUTINE
0000 0000 789 PRINT::
0000 0000 790 .WORD ^M<> ; Entry mask
0002 0002 791 ;**
0002 0002 792 ; FUNCTIONAL DESCRIPTION:
0002 0002 793 ;
0002 0002 794 ; This is the error information print subroutine. It prints the
0002 0002 795 ; following error information:
0002 0002 796 ;
0002 0002 797 ; for a RAM failure: Uncorrectable error
0002 0002 798 ;
0002 0002 799 ; failing address
0002 0002 800 ; failing array
0002 0002 801 ;
0002 0002 802 ; for any other type of error:
0002 0002 803 ;
0002 0002 804 ; failing register
0002 0002 805 ; good data, bad data, XOR (if appropriate)
0002 0002 806 ;
0002 0002 807 ; CALLING SEQUENCE:
0002 0002 808 ;
0002 0002 809 ; Called by $DS_ERRHARD_S.
0002 0002 810 ;
0002 0002 811 ; INPUT PARAMETERS:
0002 0002 812 ;
0002 0002 813 ; R3 - good (expected) data
0002 0002 814 ; R4 - bad (received) data
0002 0002 815 ; R5 - XOR of R3 and R4
0002 0002 816 ;
0002 0002 817 ; IMPLICIT INPUTS:
0002 0002 818 ;
0002 0002 819 ; ERROR_TYPE - 0 (address bus failure)
0002 0002 820 ; - 1 (data bus failure)
0002 0002 821 ; - 2 (row select bus failure)
0002 0002 822 ; - 3 (Ram failure)
0002 0002 823 ;
0002 0002 824 ; OUTPUT PARAMETERS:
0002 0002 825 ;
0002 0002 826 ; Printed ASCII error message on console.
0002 0002 827 ;
0002 0002 828 ; IMPLICIT OUTPUTS: NONE
0002 0002 829 ;
0002 0002 830 ; COMPLETION CODES: NONE
0002 0002 831 ;
0002 0002 832 ; SIDE EFFECTS: NONE
0002 0002 833 ;--
0002 0002 834 ;
04 00 00000014'EF 8F 0002 835 CASEB ERROR_TYPE,#0,#4 ; Decode type of error
0015' 000A 836 10$: .WORD 100%-10% ; Displacement for error type 0
0044' 000C 837 .WORD 200%-10% ; Displacement for error type 1
007B' 000E 838 .WORD 300%-10% ; Displacement for error type 2
00AE' 0010 839 .WORD 400%-10% ; Displacement for error type 3
0012 840 $DS_PRINTB_S FMT_UNK_ET ; Unknown error type
0001C1B'EF 9F 0012 PUSHAB FMT_UNK_ET
000100E0 9F 01 FB 0018 CALLS $$$N, @DS$PRINTB
001F 841 100$: $DS_PRINTB_S FMT_ADR_BUS,- ; Report address bus failure
001F 842 R7

```

00001D1B'EF	57	DD	001F				PUSHL	R7	
000100E0 9F	02	9F	0021				PUSHAB	FMT_ADR_BUS	
		FB	0027				CALLS	\$\$\$N, a#DS\$PRINTB	
			002E	843		\$DS_PRINTX_S		FMT_BODY,-	; Report expected, received data
			002E	844				#EXP_MSG,R3,-	
			002E	845				#REC_MSG,R4	
000021D0'8F	54	DD	002E				PUSHL	R4	
		DD	0030				PUSHL	#REC_MSG	
000021DC'8F	53	DD	0036				PUSHL	R3	
00001CC3'EF		DD	0038				PUSHL	#EXP_MSG	
000100E8 9F	05	9F	003E				PUSHAB	FMT_BODY	
	0079	FB	0044				CALLS	\$\$\$N, a#DS\$PRINTX	
		31	004B	846		BRW	PRINTX		; And exit
			004E	847	200\$:	\$DS_PRINTB_S		FMT_DATA_BUS,-	; Report data bus failure
			004E	848				R6	
00001D37'EF	56	DD	004E				PUSHL	R6	
000100E0 9F	02	9F	0050				PUSHAB	FMT_DATA_BUS	
		FB	0056				CALLS	\$\$\$N, a#DS\$PRINTB	
			005D	849		\$DS_PRINTX_S		FMT_COMP_ERR,-	; Report exp,rec, xor data
			005D	850				#EXP_MSG,R3,-	
			005D	851				#REC_MSG,R4,-	
			005D	852				#XOR_MSG,R5	
000021E8'8F	55	DD	005D				PUSHL	R5	
		DD	005F				PUSHL	#XOR_MSG	
000021D0'8F	54	DD	0065				PUSHL	R4	
		DD	0067				PUSHL	#REC_MSG	
000021DC'8F	53	DD	006D				PUSHL	R3	
00001CDC'EF		DD	006F				PUSHL	#EXP_MSG	
000100E8 9F	07	9F	0075				PUSHAB	FMT_COMP_ERR	
	0042	FB	007B				CALLS	\$\$\$N, a#DS\$PRINTX	
		31	0082	853		BRW	PRINTX		; And exit
			0085	854	300\$:	\$DS_PRINTB_S		FMT_CS_BUS,FROW	; Report row select bus failure
00001050'EF		DD	0085				PUSHL	FROW	
00001D88'EF		9F	008B				PUSHAB	FMT_CS_BUS	
000100E0 9F	02	FB	0091				CALLS	\$\$\$N, a#DS\$PRINTB	
			0098	855		\$DS_PRINTX_S		FMT_BODY,-	; Report expected, received data
			0098	856				#EXP_MSG,R3,-	
			0098	857				#REC_MSG,R4	
000021D0'8F	54	DD	0098				PUSHL	R4	
		DD	009A				PUSHL	#REC_MSG	
000021DC'8F	53	DD	00A0				PUSHL	R3	
00001CC3'EF		DD	00A2				PUSHL	#EXP_MSG	
000100E8 9F	05	9F	00A8				PUSHAB	FMT_BODY	
	000F	FB	00AE				CALLS	\$\$\$N, a#DS\$PRINTX	
		31	00B5	858		BRW	PRINTX		; And exit
			00B8	859	400\$:	\$DS_PRINTB_S		FMT_RAM,R6	; Report ram failure
00001DDD'EF	56	DD	00B8				PUSHL	R6	
000100E0 9F	02	9F	00BA				PUSHAB	FMT_RAM	
		FB	00C0				CALLS	\$\$\$N, a#DS\$PRINTB	
			00C7	860	PRINTX:				
		04	00C7	861		RET			; Exit
			00C8	862					

```

0000 00C8 864 .SBTTL PRINT GENERAL ROUTINE
00C8 865 PRINT_GENERAL::
0000 00C8 866 .WORD ^M<>
00CA 867 ;**
00CA 868 ; FUNCTIONAL DESCRIPTION:
00CA 869 ;
00CA 870 ; This routine is used to print a specified error message with the
00CA 871 ; specified parameters.
00CA 872 ;
00CA 873 ; CALLING SEQUENCE:
00CA 874 ;
00CA 875 ; Called by $DS_ERRHARD_S
00CA 876 ;
00CA 877 ; IMPLICIT INPUTS:
00CA 878 ;
00CA 879 ; ERR$_P1(AP) - Number of parameters
00CA 880 ; ERR$_P2(AP) - Address of Specific FAO String
00CA 881 ; ERR$_P3(AP) - Specific data for FAO string
00CA 882 ;
00CA 883 ;
00CA 884 ;
00CA 885 ;--
00CA 886
05 01 14 AC 8F 00CA 887 CASEB ERR$_P1(AP),#1,#5 ; Select number of parameters
000A' 00CF 888 10$: .WORD 100$ - 10$
0017' 00D1 889 .WORD 200$ - 10$
0027' 00D3 890 .WORD 300$ - 10$
003A' 00D5 891 .WORD 400$ - 10$
0050' 00D7 892 .WORD 500$ - 10$
00D9 893
00D9 894
00D9 895 100$: $DS_PRINTB_S aERR$_P2(AP)
000100E0 9F 18 BC 9F 00D9 PUSHAB aERR$_P2(AP)
000100E0 9F 01 FB 00DC CALLS $$$N, a#DS$PRINTB
004F 31 00E3 896 BRW PRINT_GEN X
00E6 897 200$: $DS_PRINTB_S aERR$_P2(AP),ERR$_P3(AP)
00E6 00E6 PUSHBL ERR$_P3(AP)
000100E0 9F 1C AC DD 00E9 PUSHAB aERR$_P2(AP)
000100E0 9F 18 BC 9F 00EC CALLS $$$N, a#DS$PRINTB
003F 31 00F3 898 BRW PRINT_GEN X
00F6 899 300$: $DS_PRINTB_S aERR$_P2(AP),ERR$_P3(AP),-
00F6 900 ERR$_P4(AP)
00F6 901 PUSHBL ERR$_P4(AP)
000100E0 9F 20 AC DD 00F9 PUSHBL ERR$_P3(AP)
000100E0 9F 1C AC DD 00FC PUSHAB aERR$_P2(AP)
000100E0 9F 18 BC 9F 00FF CALLS $$$N, a#DS$PRINTB
002C 31 0106 901 BRW PRINT_GEN X
0109 902 400$: $DS_PRINTB_S aERR$_P2(AP),ERR$_P3(AP),-
0109 903 ERR$_P4(AP),ERR$_P5(AP)
0109 904 PUSHBL ERR$_P5(AP)
000100E0 9F 24 AC DD 010C PUSHBL ERR$_P4(AP)
000100E0 9F 20 AC DD 010F PUSHBL ERR$_P3(AP)
000100E0 9F 1C AC DD 0112 PUSHAB aERR$_P2(AP)
000100E0 9F 18 BC 9F 0115 CALLS $$$N, a#DS$PRINTB
0016 31 011C 904 BRW PRINT_GEN X
011F 905 500$: $DS_PRINTB_S aERR$_P2(AP),ERR$_P3(AP),-
011F 906 ERR$_P4(AP),ERR$_P5(AP),-

```

ZZ-ECKAM-3.2
ECKAM
03-02

PRINT GENERAL ROUTINE
MEMORY DIAGNOSTIC HEADER
PRINT GENERAL ROUTINE

M 4

13-MAY-1987 Fiche 1 Frame M4 Sequence 51
23-JAN-1987 14:30:12 VAX/VMS Macro V04-00 Page 29
23-JAN-1987 14:26:49 [ZECCHINO.ECKAM]ECKAM0.MAR;4 (6)

```
          28 AC DD 011F 907
          24 AC DD 0122
          20 AC DD 0125
           1C AC DD 0128
           18 BC 9F 012B
000100E0 9F 05 FB 012E
          04 0135 908 PRINT_GEN_X:
          0135 909 RET
          0136 910
```

```
PUSHL ERR$_P6(AP)
PUSHL ERR$_P6(AP)
PUSHL ERR$_P5(AP)
PUSHL ERR$_P4(AP)
PUSHL ERR$_P3(AP)
PUSHAB aERR$_P2(AP)
CALLS $$$N, a#DS$PRINTB
```

```
00E0 0136 912 .SBTTL PRINT UNEXPECTED UNCORRECTABLE ERROR ROUTINE
0136 913 PRINT_UNX_UNC::
0136 914 .WORD ^M<R5,R6,R7>
0138 915 ;++
0138 916 ;
0138 917 ; FUNCTIONAL DESCRIPTION:
0138 918 ;
0138 919 ; THIS ROUTINE IS CALLED WHEN AN UNEXPECTED MACHINE CHECK IS
0138 920 ; DETECTED IN A TEST THAT CHECKS FOR UNEXPECTED UNCORRECTABLE
0138 921 ; ERRORS. THE CALL IS ALWAYS VIA A PRINT LINK.
0138 922 ;
0138 923 ; IMPLICIT INPUTS:
0138 924 ;
0138 925 ; ERR$_P1(AP) - ADDRESS REFERENCED
0138 926 ;
0138 927 ;--
0138 928
0138 929 $DS_PRINTB_S FMT_UNX_UNC,ERR$_P1(AP)
0138 930 PUSHL ERR$_P1(AP)
0138 931 PUSHAB FMT_UNX_UNC
0141 932 CALLS $$$N, a#DS$PRINTB
0148 933 ; Print error message and address
0148 934 ;
0148 935 ; dump the memory registers
0148 936 ;
0148 937 BICL3 #^X1F000180,CSR0(R2),R5 ; Get CSR0
0150 938 $DS_CVTREG_S #31,R5,CSR0_MNEM,-
0150 939 CVTREG_BUFF,#44
0150 940 PUSHL #0
0152 941 PUSHL #0
0154 942 PUSHL #0
0156 943 PUSHL #0
0158 944 PUSHL #0
015A 945 PUSHL #0
015C 946 PUSHL #44
015E 947 PUSHAB CVTREG_BUFF
0164 948 PUSHAB CSR0_MNEM
016A 949 PUSHL R5
016C 950 PUSHL #31
016E 951 CALLS #11, a#DS$CVTREG
0175 952 BICL3 #^XE0000180,CSR1(R2),R6
017E 953 $DS_CVTREG_S #28,R6,CSR1_MNEM,-
017E 954 CVTREG_BUFF+44,#44
017E 955 PUSHL #0
0180 956 PUSHL #0
0182 957 PUSHL #0
0184 958 PUSHL #0
0186 959 PUSHL #0
0188 960 PUSHL #0
018A 961 PUSHL #44
018C 962 PUSHAB CVTREG_BUFF+44
0192 963 PUSHAB CSR1_MNEM
0198 964 PUSHL R6
019A 965 PUSHL #28
019C 966 CALLS #11, a#DS$CVTREG
01A3 967 BICL3 #^XFF000000,CSR2(R2),R7
01AC 968 $DS_CVTREG_S #23,R7,CSR2_MNEM,-
```

14 AC DD 0138
00001C7D'EF 9F 013B
000100E0 9F 02 FB 0141

55 62 1F000180 8F CB 0148
00 DD 0150
00 DD 0152
00 DD 0154
00 DD 0156
00 DD 0158
00 DD 015A
2C DD 015C
00001078'EF 9F 015E
000010FC'EF 9F 0164
55 DD 016A
1F DD 016C
000100B0 9F 0B FB 016E
56 04 A2 E0000180 8F CB 0175
00 DD 017E
00 DD 0180
00 DD 0182
00 DD 0184
00 DD 0186
00 DD 0188
2C DD 018A
000010A4'EF 9F 018C
0000113A'EF 9F 0192
56 DD 0198
1C DD 019A
000100B0 9F 0B FB 019C
57 08 A2 FF000000 8F CB 01A3
01AC 941

		01AC	942		CVTREG_BUFF+88,#44
	00	DD	01AC		PUSHL #0
	00	DD	01AE		PUSHL #0
	00	DD	01B0		PUSHL #0
	00	DD	01B2		PUSHL #0
	00	DD	01B4		PUSHL #0
	00	DD	01B6		PUSHL #0
	2C	DD	01B8		PUSHL #44
000010D0	'EF	9F	01BA		PUSHAB CVTREG_BUFF+88
00001171	'EF	9F	01C0		PUSHAB CSR2_MNEM
	57	DD	01C6		PUSHL R7
	17	DD	01C8		PUSHL #23
000100B0	9F	0B	FB	01CA	CALLS #11, @#DS\$CVTREG
				01D1	FMT_MEM_REG,-
				01D1	R5,#CVTREG_BUFF,-
				01D1	R6,#CVTREG_BUFF+44,-
				01D1	R7,#CVTREG_BUFF+88
	000010D0	'8F	DD	01D1	#CVTREG_BUFF+88
	57	DD	01D7		PUSHL R7
000010A4	'8F	DD	01D9		PUSHL #CVTREG_BUFF+44
	56	DD	01DF		PUSHL R6
00001078	'8F	DD	01E1		PUSHL #CVTREG_BUFF
	55	DD	01E7		PUSHL R5
000023CB	'EF	9F	01E9		PUSHAB FMT_MEM_REG
000100E0	9F	07	FB	01EF	CALLS \$\$\$N, @#DS\$PRINTB
		04	01F6	947	
			01F7	948	
				RET	

```

0000 01F7 950 .SBTTL PRINT MEMORY SIZING ERROR
      01F7 951 PRINT_BAD_CONFIG::
      01F7 952 .WORD ^M<>
      01F9 953 ;++
      01F9 954 ; FUNCTIONAL DESCRIPTION
      01F9 955 ;
      01F9 956 ; THIS ROUTINE IS CALLED WHEN THE MAXLWA DOESNT AGREE WITH THE GENERATED
      01F9 957 ; TEST ADDRESS FROM THE MEMORY MAP (CSR2).
      01F9 958 ;
      01F9 959 ; CALLING SEQUENCE:
      01F9 960 ;
      01F9 961 ; Called by $DS_ERRHARD_S
      01F9 962 ;
      01F9 963 ; INPUT PARAMETERS: NONE
      01F9 964 ;
      01F9 965 ; IMPLICIT OUTPUTS:
      01F9 966 ;
      01F9 967 ; A printed ASCII information message on console
      01F9 968 ;
      01F9 969 ; COMPLETION CODES:: NONE
      01F9 970 ;
      01F9 971 ; SIDE EFFECTS:: NONE
      01F9 972 ;--
      01F9 973 $DS_PRINTB_S-
      01F9 974 FORMAT= FMT_SIZ_ERR,-
      01F9 975 P1 = TEST_ADDRESS,-
      01F9 976 P2 = MAXLWA
      DD 01F9 PUSHL MAXLWA
      DD 01FF PUSHL TEST_ADDRESS
      9F 0205 PUSHAB FMT_SIZ_ERR
00000449'EF DD 020B CALLS $$$N, a#DS$PRINTB
00000451'EF DD 0212 RET
00001E52'EF 9F 0213
000100E0 9F 03 FB

```

```

0000 0213 979 CORRUPT_TEST_LOC::
      0213 980 .WORD ^M<>
      0215 981 ;++
      0215 982 ;
      0215 983 ; FUNCTIONAL DESCRIPTION:
      0215 984 ;
      0215 985 ; THIS ROUTINE IS CALLED WHEN A MEMORY LOCATION THAT IS REFERENCED FOR
      0215 986 ; TESTING PURPOSES IS FOUND CORRUPTED. A SINGLE BIT ERROR IN THE LOCATION
      0215 987 ; WILL CAUSE THIS ROUTINE TO BE INVOKED.
      0215 988 ;
      0215 989 ; CALLING SEQUENCE:
      0215 990 ;
      0215 991 ; Called by $DS_ERRHARD_S
      0215 992 ;
      0215 993 ; INPUT PARAMETERS: ERR$_P1(AP)
      0215 994 ;
      0215 995 ; IMPLICIT OUTPUTS:
      0215 996 ;
      0215 997 ; A printed ASCII error message printed on the console
      0215 998 ;
      0215 999 ; COMPLETION CODES: NONE
      0215 1000 ;
      0215 1001 ; SIDE EFFECTS: NONE
      0215 1002 ;--

```

ZZ-ECKAM-3.2
ECKAM
03-02

PRINT CORRUPTED

TEST LOCATION
MEMORY DIAGNOSTIC HEADER
PRINT CORRUPTED TEST LOCATION

E 5

13-MAY-1987 Fiche 1 Frame E5 Sequence 56
23-JAN-1987 14:30:12 VAX/VMS Macro V04-00 Page 33
23-JAN-1987 14:26:49 [ZECCHINO.ECKAM]ECKAM0.MAR;4 (9)

			0215	1003		\$DS_PRINTB_S-		
			0215	1004		FORMAT=		FMT_BAD_LOC,-
			0215	1005		P1 =		ERR\$_P1(AP)
	14 AC	DD	0215			PUSHL		ERR\$_P1(AP)
	00001EAD'EF	9F	0218			PUSHAB		FMT_BAD_LOC
000100E0 9F	02	FB	021E			CALLS		\$\$\$N, a#DS\$PRINTB
		04	0225	1006	RET			

```

0226 1008 .SBTTL PRINT UNEXPECTED MACHINE CHECK ROUTINE
0226 1009 PRINT_UNX_TO::
0000 0226 1010 .WORD ^M<>
0228 1011 ;++
0228 1012 ;
0228 1013 ; FUNCTIONAL DESCRIPTION:
0228 1014 ;
0228 1015 ; THIS ROUTINE IS CALLED WHEN AN UNEXPECTED MACHINE CHECK IS
0228 1016 ; DETECTED.
0228 1017 ;
0228 1018 ; IMPLICIT INPUTS:
0228 1019 ;
0228 1020 ; 4(AP) - ADDRESS OF MACHINE CHECK STACK
0228 1021 ;
0228 1022 ;--
0228 1023
0228 1024 $DS_PRINTB_S FMT_UNK_MC ; Type the error message
00001C30'EF 9F 0228 PUSHAB FMT_UNK_MC
000100E0 9F 01 FB 022E CALLS $$$N, a#DS$PRINTB
0235 1025 ;
0235 1026 ; Dump the Machine Check Stack
0235 1027 ;
55 04 AC D0 0235 1028 MOVL 4(AP),R5 ; Get address of machine check stack
0239 1029 $DS_PRINTB_S FMT_MCH_STACK1,-
0239 1030 (R5),4(R5),8(R5),-
0239 1031 12(R5),16(R5),20(R5),-
0239 1032 24(R5)
18 A5 DD 0239 PUSHL 24(R5)
14 A5 DD 023C PUSHL 20(R5)
10 A5 DD 023F PUSHL 16(R5)
0C A5 DD 0242 PUSHL 12(R5)
08 A5 DD 0245 PUSHL 8(R5)
04 A5 DD 0248 PUSHL 4(R5)
65 DD 024B PUSHL (R5)
00002434'EF 9F 024D PUSHAB FMT_MCH_STACK1
000100E0 9F 08 FB 0253 CALLS $$$N, a#DS$PRINTB
025A 1033 $DS_PRINTB_S FMT_MCH_STACK2,-
025A 1034 28(R5),32(R5),-
025A 1035 36(R5),40(R5),44(R5),-
025A 1036 48(R5)
30 A5 DD 025A PUSHL 48(R5)
2C A5 DD 025D PUSHL 44(R5)
28 A5 DD 0260 PUSHL 40(R5)
24 A5 DD 0263 PUSHL 36(R5)
20 A5 DD 0266 PUSHL 32(R5)
1C A5 DD 0269 PUSHL 28(R5)
000024F1'EF 9F 026C PUSHAB FMT_MCH_STACK2
000100E0 9F 07 FB 0272 CALLS $$$N, a#DS$PRINTB
0279 1037 ; Print the stack
04 0279 1038 RET

```

```

027A 1040 .SBTTL WHICH ARRAY ROUTINE
027A 1041 WHICH_ARRAY::
027A 1042 ;++
027A 1043 ; FUNCTIONAL DESCRIPTION:
027A 1044 ;
027A 1045 ; This routine determines which array module is failing.
027A 1046 ;
027A 1047 ; CALLING SEQUENCE:
027A 1048 ;
027A 1049 ; JSB
027A 1050 ;
027A 1051 ; INPUT PARAMETERS:
027A 1052 ;
027A 1053 ; R7 - failing row number
027A 1054 ;
027A 1055 ; IMPLICIT INPUTS:
027A 1056 ;
027A 1057 ; ROWCON - table representing number of rows found in each module.
027A 1058 ;
027A 1059 ; OUTPUT PARAMETERS: NONE
027A 1060 ;
027A 1061 ; IMPLICIT OUTPUTS:
027A 1062 ;
027A 1063 ; A counted ASCII string in OWN STORAGE identifying the failing array
027A 1064 ; module.
027A 1065 ;
027A 1066 ; COMPLETION CODES:
027A 1067 ;
027A 1068 ; R0 - contains array number
027A 1069 ;
027A 1070 ; SIDE EFFECTS: NONE
027A 1071 ;--
027A 1072
57 00001048'EF40 50 D4 027A 1073 CLRL R0 ; Initialize slot counter
; If R7 LEQ the max row of this slot
; Then this slot is to be output
; Else, go back and look at next slot
; Keep track of current array
; Put failing array number into string
; And copy it to another string
; And exit
00001070'EF 04 18 0284 1075 10$: CMPB ROWCON[R0],R7
; If R7 LEQ the max row of this slot
; Then this slot is to be output
; Else, go back and look at next slot
; Keep track of current array
; Put failing array number into string
; And copy it to another string
; And exit
000011A1'EF 07 F3 0286 1076 AOBLEQ #7,R0,10$
; If R7 LEQ the max row of this slot
; Then this slot is to be output
; Else, go back and look at next slot
; Keep track of current array
; Put failing array number into string
; And copy it to another string
; And exit
00001422'EF 50 D0 028A 1077 20$: MOVL R0,L_CUR_ARRAY
; If R7 LEQ the max row of this slot
; Then this slot is to be output
; Else, go back and look at next slot
; Keep track of current array
; Put failing array number into string
; And copy it to another string
; And exit
000011A1'EF 30 81 0291 1078 ADDB3 #^X30,R0,DIGIT
; If R7 LEQ the max row of this slot
; Then this slot is to be output
; Else, go back and look at next slot
; Keep track of current array
; Put failing array number into string
; And copy it to another string
; And exit
02A4 1080
02A4 1081 WHICH_ARRAYX:
05 02A4 1082 RSB

```

```

02A5 1084 .SBTTL DECODER ROUTINE
02A5 1085 DECODER::
02A5 1086 ;**
02A5 1087 ; FUNCTIONAL DESCRIPTION:
02A5 1088 ;
02A5 1089 ; This is used in the row select bus and address bus tests.
02A5 1090 ; This routine decodes a table of possibly corrupted codewords.
02A5 1091 ; It first decodes the 4 MSBs of all of the codewords, and then decodes
02A5 1092 ; LSB. This routine decodes codewords of the Reed-Muller (32,6) coding
02A5 1093 ; space only.
02A5 1094 ;
02A5 1095 ; The basis of the Reed-Muller (32,6) coding space comprises the vectors
02A5 1096 ;
02A5 1097 ; I 11111111111111111111111111111111.
02A5 1098 ; V1 01010101010101010101010101010101
02A5 1099 ; V2 00110011001100110011001100110011
02A5 1100 ; V3 00001111000011110000111100001111
02A5 1101 ; V4 00000000111111110000000011111111
02A5 1102 ;
02A5 1103 ; A codeword is created with the following expression:
02A5 1104 ;
02A5 1105 ; codeword = G0*I XOR G1*V1 XOR G2*V2 XOR G3*V3 XOR G4*V4,
02A5 1106 ;
02A5 1107 ; where G0 is the LSB of the word to encode, and G4 is the MSB.
02A5 1108 ;
02A5 1109 ; Each codeword has 16 disjoint redundant relations due to the nature of
02A5 1110 ; the encoding vectors. Note that in V1, the redundant relations are
02A5 1111 ; the bit pairs <31:30>, <29:28>, <27:26>, and so on. In V2, the
02A5 1112 ; redundant relations are the bit pairs <31:29>, <30:28>, <27:25>, and
02A5 1113 ; so on. In V3 the redundant relations are the bit pairs <31:27>,
02A5 1114 ; <30:26>, <29:25>, and so on. In V4, the redundant relations are the
02A5 1115 ; bit pairs <31:23>, <30:22>, <29:21>, and so on. Thus, there are four
02A5 1116 ; sets of redundant relations in each codeword. Each bit is thus
02A5 1117 ; represented in a codeword by 16 disjoint relations.
02A5 1118 ;
02A5 1119 ; To decode a particular bit, we calculate the value of each of the 16
02A5 1120 ; redundant relations. The position (weight) of the bit in the decoded
02A5 1121 ; word determines which bit pairs to use. The original word (that is,
02A5 1122 ; the 'message' word) is represented as
02A5 1123 ;
02A5 1124 ;
02A5 1125 ; -----
02A5 1126 ; !G4!G3!G2!G1!G0!
02A5 1127 ; -----
02A5 1128 ;
02A5 1129 ; The LSB has no redundant relations inherent, since the value of G0 in
02A5 1130 ; the codeword is represented by the product G0*I (I, the identity
02A5 1131 ; vector, has no redundant relations). The next bit, G1, has the
02A5 1132 ; redundant relations represented by the set of 16 bit pairs, where the
02A5 1133 ; bit pairs are adjacent. We introduce here the notion of pair
02A5 1134 ; separation. The pair separation for the redundant relations
02A5 1135 ; representing G1 is 1. For G2, it is 2, for G3, it is 4, and for G4
02A5 1136 ; it is 8.
02A5 1137 ;
02A5 1138 ; After decoding the value of each redundant relation for a particular
02A5 1139 ; bit, the sum of all of these values is taken. If there was no
02A5 1140 ; corruption of the codeword, the sum of all of the values of redundant

```

```

02A5 1141 ; relations will be zero or 16. If there was some corruption of the
02A5 1142 ; codeword, then the sum will be in between 0 and 16. Therefore a
02A5 1143 ; majority decision is made: if the sum is 8 or more, then the value of
02A5 1144 ; the message bit (that is, Gn) is a one; if the sum is less than 8, it
02A5 1145 ; is a zero.
02A5 1146 ;
02A5 1147 ; After decoding the four MSBs, the LSB is decoded by eliminating each
02A5 1148 ; of the terms G1*V1, G2*V2, G3*V3, and G4*V4 from the codeword. This
02A5 1149 ; leaves simply G0*I. Therefore, with no corruption in the codeword,
02A5 1150 ; this term would either be 32 bits of zeros or 32 bits of ones.
02A5 1151 ; However, if there was corruption, then the number of bits set in this
02A5 1152 ; term would lie between 0 and 32. Again, a majority decision specifies
02A5 1153 ; the value of this term, and thus G0. Greater than 15 bits set will
02A5 1154 ; indicate that G0 is a one, and 15 or fewer bits will indicate that G0
02A5 1155 ; is a zero.
02A5 1156 ;
02A5 1157 ; CALLING SEQUENCE:
02A5 1158 ;
02A5 1159 ; JSB
02A5 1160 ;
02A5 1161 ; INPUT PARAMETERS:
02A5 1162 ;
02A5 1163 ; R4 - contains number of words to decode
02A5 1164 ;
02A5 1165 ; IMPLICIT INPUTS:
02A5 1166 ;
02A5 1167 ; NOISY_WORDS - table containing the possibly corrupted codewords
02A5 1168 ;
02A5 1169 ; OUTPUT PARAMETERS: NONE
02A5 1170 ;
02A5 1171 ; IMPLICIT OUTPUTS:
02A5 1172 ;
02A5 1173 ; DECODED_WORDS - table containing the decoded words
02A5 1174 ;
02A5 1175 ; COMPLETION CODES: NONE
02A5 1176 ;
02A5 1177 ; SIDE EFFECTS: NONE
02A5 1178 ;--
02A5 1179 ;
07EC 8F BB 02A5 1180 PUSHR #^M<R2,R3,R5,R6,R7,R8,R9,R10>; Save registers
02A9 1181 ;+
02A9 1182 ; First, decode the 4 MSBs of all of the possibly corrupted codewords.
02A9 1183 ;-
00000B3C *EF42 52 D4 02A9 1184 CLRL R2 ; Initialize word counter
53 01 D4 02AB 1185 10%: CLRL DECODED_WORDS[R2] ; Initialize decoded word storage
55 01 D0 02B2 1186 MOVL #1,R3 ; Initialize the pair separation
56 1F D0 02B5 1187 MOVL #1,R5 ; Initialize the message bit counter
58 D4 02B8 1188 20%: MOVL #31,R6 ; Initialize the codeword bit position
55 01 D1 02BB 1189 CLRL R8 ; Initialize the population counter
04 13 D4 02BD 1190 CMPL #1,R5 ; If we are decoding bit 1,
57 53 53 01 78 02C0 1191 BEQL 30% ; Then skip the following line
00000BBC *EF42 D0 02C2 1192 ASHL #1,R3,R3 ; Else, update the pair separation
05 57 56 E1 02C6 1193 30%: MOVL NOISY_WORDS[R2],R7 ; Get the possibly corrupted codeword
50 01 D0 02CE 1194 BBC R6,R7,40% ; If this bit of the codeword set,
02 11 D0 02D2 1195 MOVL #1,R0 ; Then store a one
50 D4 02D5 1196 BRB 50% ; And branch
02D7 1197 40%: CLRL R0 ; Else, store a one

```

```

05 56 53 C2 02D9 1198 50$:  SUBL2  R3,R6      ; Point to the next bit of the pair
    57 56 E1 02DC 1199      BBC      R6,R7,60$  ; If this bit of the codeword set,
    51 01 D0 02E0 1200      MOVL    #1,R1    ; Then store a one
        02 11 02E3 1201      BRB     70$      ; And branch
    51 51 D4 02E5 1202 60$:  CLRL    R1      ; Else, store a one
    51 50 CC 02E7 1203 70$:  XORL2   R0,R1    ; Compute the value of the relation
        02 13 02EA 1204      BEQL    80$      ; If it is a one, then
        58 D6 02EC 1205      INCL    R8      ; Bump the population counter
        02EE 1206 ;
        02EE 1207 ; Here the position of the first bit of the next redundant relation pair is
        02EE 1208 ; calculated.
        02EE 1209 ;
59 50 01 55 78 02EE 1210 80$:  ASHL    R5,#1,R0  ; Calculate 2**R5 (R5 is message bit)
50 00 56 01 7A 02F2 1211      EMUL    #1,R6,#0,R9 ; If the bit position
50 50 59 50 7B 02F7 1212      EDIV    R0,R9,R0,R0 ; modulo (2**R5)
        50 D5 02FC 1213      TSTL    R0      ; is not zero,
        03 13 02FE 1214      BEQL    90$      ; then
        56 53 C0 0300 1215      ADDL2   R3,R6    ; Add back the pair separation
        C0 56 F4 0303 1216 90$:  SOBGEQ  R6,30$   ; Go back for the next redundant pair
        08 58 D1 0306 1217      CMPL    R8,#8    ; Is the population greater than 8?
        0B 15 0309 1218      BLEQ    100$    ; Branch if not (the bit is zero)
00000B3C*EF42 DF 030B 1219      PUSHAL DECODED_WORDS[R2] ; Prepare to set the bit
    00 9E 55 E2 0312 1220      BBSS   R5,a(SP)+,100$ ; And set it
    9E 55 04 F3 0316 1221 100$:  AOBLEQ  #4,R5,20$  ; Do for all of the message bits
    8D 52 54 F2 031A 1222      AOBLSS R4,R2,10$  ; Do for all of the codewords
        031E 1223 ;+
        031E 1224 ; Now that bits 1 through 4 have been decoded, decode bit 0 of all of the
        031E 1225 ; codewords.
        031E 1226 ;-
50 0000BBC*EF42 52 D4 031E 1227      CLRL    R2      ; Initialize word counter
        53 01 D0 0320 1228 110$:  MOVL    NOISY_WORDS[R2],R0 ; Get a possibly corrupted codeword
56 0000B3C*EF42 53 01 D0 0328 1229      MOVL    #1,R3    ; Initialize bit counter
56 56 01 53 EF 0333 1231 120$:  MOVL    DECODED_WORDS[R2],R6 ; Get a partially decoded word
56 0000475*EF43 C4 0338 1232      EXTZV  R3,#1,R6,R6 ; Get a decoded bit
        50 56 CC 0340 1233      MULL2  CODEVECTORS[R3],R6 ; Form the product Gn*Vn
        E4 53 04 F3 0343 1234      XORL2   R6,R0    ; Remove the term Gn*Vn from the word
        55 D4 0347 1235      AOBLEQ  #4,R3,120$ ; Do for all four LSBs
        53 D4 0349 1236      CLRL    R5      ; Initialize population counter
        02 50 53 E1 034B 1237 130$:  CLRL    R3      ; Initialize bit counter
        55 D6 034F 1238      BBC     R3,R0,140$ ; If the bit is a one,
        F6 53 1F F3 0351 1239 140$:  INCL    R5      ; Then bump the population counter
        10 55 D1 0355 1240      AOBLEQ  #31,R3,130$ ; Do for the whole longword
        0B 15 0358 1241      CMPL    R5,#16  ; Make the majority decision
00000B3C*EF42 DF 035A 1242      BLEQ    150$    ; If the ones population exceeds 16
    00 9E 00 E2 0361 1243      PUSHAL DECODED_WORDS[R2] ; Prepare to set the LSB
    B7 52 54 F2 0365 1244 150$:  BBSS   #0,a(SP)+,150$ ; Set the LSB in the decoded word
        0369 1245      AOBLSS R4,R2,110$ ; Do for all words in the table
        0369 1246
07EC 8F BA 0369 1247      DECODERX:
        05 036D 1248      POPR    #^M<R2,R3,R5,R6,R7,R8,R9,R10>; Restore registers
        RSB      ; And return

```

```

036E 1250 .SBTTL INITIALIZATION ROUTINE
036E 1251 $DS_BGNINIT
036E .SAVE
00000000 .PSECT INITIALIZE, LONG
0000 0000 INITIALIZE:
0000 0000 .WORD ^M<> ; ENTRY MASK
0002 1252 :++
0002 1253 : FUNCTIONAL DESCRIPTION:
0002 1254 :
0002 1255 : This is the initialization routine. Here the memory is sized by
0002 1256 : accessing pages until a machine check occurs. Also, the Reed-Muller
0002 1257 : code words used in the address bus test and row select bus test
0002 1258 : are generated and stored.
0002 1259 :
0002 1260 : CALLING SEQUENCE:
0002 1261 :
0002 1262 : Called by the Diagnostic Supervisor at the beginning of every test
0002 1263 : sequence.
0002 1264 :
0002 1265 : INPUT PARAMETERS: NONE
0002 1266 :
0002 1267 : IMPLICIT INPUTS:
0002 1268 :
0002 1269 : CODEVECTORS - a table of vectors used to generate the Reed-Muller code
0002 1270 : words.
0002 1271 :
0002 1272 : OUTPUT PARAMETERS: NONE
0002 1273 :
0002 1274 : IMPLICIT OUTPUTS:
0002 1275 :
0002 1276 : CODEWORDS - a table of 32 Reed-Muller code words for later use.
0002 1277 : MAXLWA - maximum longword address of memory.
0002 1278 : MAXROW - number of existing rows of RAMs.
0002 1279 :
0002 1280 :
0002 1281 : COMPLETION CODES: NONE
0002 1282 :
0002 1283 : SIDE EFFECTS: NONE
0002 1284 :--
0002 1285
0002 1286 $DS_ENDPASS_S
0002 CALLS #0, a#DS$ENDPASS
0009 1287 $DS_SETVEC_S #4, MACHINECK ; Set up machine check vector
0009 PUSHL #0
000B PUSHAL MACHINECK
0011 PUSHL #4
0013 CALLS #3, a#DS$SETVEC
001A 1288 $DS_SETVEC_S #^X54, - ; Set up single-bit error vector
001A 1289 CORR_ERR_ISR
001A PUSHL #0
001C PUSHAL CORR_ERR_ISR
0022 PUSHL #^X54
0028 CALLS #3, a#DS$SETVEC
002F 1290 MOVL #CONTROLLER, R2 ; This is memory controller
0036 1291 ; address for use in all tests
0036 1292 CLRL CSR1(R2) ; Disable single bit error interrupts A2 M
0039 1293 CLRL EXP_UNC_ERR ; Clear the expected error flags A9

```

```

0000001D'EF D4 003F 1294 CLRL EXP_COR_ERR ; ... A9
      26 0F DA 0045 1295 MTPR #^XF,#^X26 ; Clear the machine check error
      0048 1296 ; summary register A9
0000000C'EF 00000071'EF DE 0048 1297 MOVAL 20$,RETURN_PC ; Set up machineck return PC
00000029'EF 00000101 8F D0 0053 1298 MOVL #^X101,EXP_NX_MEM ; Indicate nx mem reference expected M2
      57 D4 005E 1299 CLRL R7 ; Clear PFN
      58 D4 0060 1300 CLRL R8 ; Clear counter
      57 09 0F 58 F0 0062 1301 10$: INSV R8,#15,#09,R7 ; Calculate new PFN and
      67 D5 0067 1302 TSTL (R7) ; Read memory locations
F1 58 00000400 8F F3 0069 1303 AOBLEQ #1024,R8,10$ ; [3.0] M1.1 Until machineck occurs or have
      0071 1304 ; Reached highest possible PFN
      57 00000200 8F C2 0071 1305 20$: SUBL2 #512,R7 ; Back up one PFN
00000445'EF 57 D0 0078 1306 MOVL R7,MAXPFN ; Save highest PFN for future ref
      57 000001FC 8F C0 007F 1307 ADDL #^X1FC,R7 ; Calcualte highest longword address
00000449'EF 57 D0 0086 1308 MOVL R7,MAXLWA ; And store it for future reference
0000FE54'EF 01 D1 008D 1309 CMPL #1,DSA$GL_PASSNO ; Pass 1? A8
      29 1F 0094 1310 BLSSU 21$ ; Branch if no A8
      0096 1311 $DS_PRINTX_S FMT_WARN_MESS ; Print warnning message
      0001F70'EF 9F 0096 PUSHAB FMT_WARN_MESS
000100E8 9F 01 FB 009C CALLS $$$N, a#DS$PRINTX
      00A3 1312 $DS_PRINTX_S FMT_WARN_MES2 ; Print second half of message
      0002057'EF 9F 00A3 PUSHAB FMT_WARN_MES2
000100E8 9F 01 FB 00A9 CALLS $$$N, a#DS$PRINTX
      00B0 1313 $DS_PRINTX_S FMT_MEM_SIZ,R7 ; Print highest longword address
      57 DD 00B0 PUSHL R7
      000011E2'EF 9F 00B2 PUSHAB FMT_MEM_SIZ
000100E8 9F 02 FB 00B8 CALLS $$$N, a#DS$PRINTX
      00BF 1314 ;
      00BF 1315 ; Here the number of rows in the system are determine
      00BF 1316 ;
000005B9'EF 16 00BF 1317 21$: JSB READ_MAP
      51 D4 00C5 1318 CLRL R1 ; Init a counter
      50 D4 00C7 1319 CLRL R0 ; Use for indexing
0000103C'EF40 95 00C9 1320 22$: TSTB SLOT[R0] ; Test to find array type
      08 13 00D0 1321 BEQL 23$ ; If slot is empty were finish
      50 D6 00D2 1322 INCL R0 ; Look at next slot
      51 04 80 00D4 1323 ADDB #4,R1 ; The number of rows per array
      FFEF 31 00D7 1324 BRW 22$ ; Go back for all arrays
0000044D'EF 51 D0 00DA 1325 23$: MOVL R1,MAXROW ; We now have the # of rows
      00E1 1326 ; Here the Reed-Muller code words are generated:
      00E1 1327 ;
      50 D4 00E1 1328 25$: CLRL R0 ; Clear accumulator M8
      51 D4 00E3 1329 CLRL R1 ; Clear index
      53 D4 00E5 1330 CLRL R3 ; Clear working register
      54 04 D0 00E7 1331 30$: MOVL #4,R4 ; Initialize bit pointer
      53 51 01 54 EF 00EA 1332 40$: EXTZV R4,#1,R1,R3 ; Get bit R4 of row index
      53 00000475'EF44 C4 00EF 1333 MULL2 CODEVECTORS[R4],R3 ; Multiply bit R4 by vector
      50 53 CC 00F7 1334 XORL2 R3,R0 ; R0 = binary sum R0 + R3
      ED 54 F4 00FA 1335 SOBGEQ R4,40$ ; Do for bits 4 through 0
00000489'EF41 50 D0 00FD 1336 MOVL R0,CODEWORDS[R1] ; Save the code word
      50 D4 0105 1337 CLRL R0 ; And clear the accumulator
      DC 51 1F F3 0107 1338 AOBLEQ #31,R1,30$ ; Create all 32 code words
      010B 1339 ;
      010B 1340 ; Here cache is disabled:
      010B 1341 ;
      25 01 DA 010B 1342 MTPR #DISABLE_CACHE,#CADR ; Disable cache
      010E 1343

```

```

010E 1344 ;
010E 1345 ; Initialize the expected error flags
010E 1346 ;
50 0000001D'EF DE 010E 1347 MOVAL EXP_COR_ERR,R0 ; Get start address of flags A2
51 D4 0115 1348 CLRL R1 ; Set a counter A2
80 D4 0117 1349 50$: CLRL (R0)+ ; Clear the flag A2
FFF8 51 01 03 F1 0119 1350 ACBL #3,#1,R1,50$ ; 4 flags A2
011F 1351 ;
011F 1352 ; Initialize the counters for correctable errors
011F 1353 ;
00000439'EF D4 011F 1354 CLRL TOTAL_COR_ERR
00000031'EF D4 0125 1355 CLRL PGE_COR_ERR
012B 1356 ;
012B 1357 ; Initialize the error storage region for the page address for CRD errors
012B 1358 ;
50 00000C3C'EF DE 012B 1359 MOVAL COR_ERRORS,R0 ; A2
51 D4 0132 1360 CLRL R1 ; A2
80 D4 0134 1361 60$: CLRL (R0)+ ; A2
FFF4 51 01 000000FF 8F F1 0136 1362 ACBL #255,#1,R1,60$ ; A2
0140 1363 ; Initialize the error storage region for the Virtual address for CRD errors
0140 1364 ;
50 00000035'EF DE 0140 1365 MOVAL VA_COR_ADDRESS,R0 ;
51 D4 0147 1366 CLRL R1 ;
80 D4 0149 1367 62$: CLRL (R0)+ ;
FFF4 51 01 000000FF 8F F1 014B 1368 ACBL #255,#1,R1,62$ ;
0000043D'EF D4 0155 1369 CLRL SUMMARY_FLAG ; Indicate no summary message A9
015B 1370 ; CLRL FULL_FLAG ; Indicate error log not full
015B 1371 ;
015B 1372 ; Find the second free page of memory above the supervisor
015B 1373 ;
53 0000FE00 8F 00000014'EF D4 015B 1374 CLRL R3 ; Init a page counter
C3 015D 1375 SUBL3 L$A_LASTAD,#^XFE00,R3 ; Calculate free space of
0169 1376 ; program end and supervisor
0169 1377 ; begining
53 00000200 8F C6 0169 1378 DIVL #512,R3 ; Calculate the # of pages
53 01 80 0170 1379 ADDB #1,R3 ; Sets it 1 page higher for safety
0173 1380 65$:
0173 1381 $DS_GETBUF_S PAGCNT = R3,-
0173 1382 RETADR = FREE_PAGE_V,-
0173 1383 PHYADR = FREE_PAGE_P
00 DD 0173 PUSHL #0
00000465'EF 7F 0175 PUSHAQ FREE_PAGE_P
0000046D'EF 7F 017B PUSHAQ FREE_PAGE_V
53 DD 0181 PUSHL R3
00010120 9F 04 FB 0183 CALLS #4, @DS$GETBUF
018A 1384 ; Get a page of memory
018A 1385 $DS_BNERROR 100$ ; Branch if no error
50 DD 018A PUSHL R0
01 DD 018C PUSHL #SER
27 DD 018E PUSHL #DS$K_BNERROR
000100A8 9F 03 FB 0190 CALLS #3, @DS$BRANCH
0A 50 E8 0197 BLBS R0, 100$
00000465'EF 00 7D 019A 1386 MOVQ #0,FREE_PAGE_P ; Set free page to NULL
0000 31 01A1 1387 BRW 100$ ; Exit
00 DD 01A4 1388 100$: $DS_RELBUF_S R3 ; Release all pages
00 DD 01A4 PUSHL #0
00 DD 01A6 PUSHL #0

```

```

00010128 9F 53 DD 01A8          PUSHL  R3
                                CALLS  #3, a#DS$RELBUF
                                03 FB 01AA
                                01B1 1389 ;
                                01B1 1390 ;Here is where the first array type is determine and the row boundary is
                                01B1 1391 ;adjusted accordingly
                                01B1 1392 ;
                                50 D4 01B1 1393          CLRL      R0          ; Use to index
0000103C'EF40 95 01B3 1394          TSTB     SLOT[R0]     ; Check first array type
                                6B 13 01BA 1395          BEQL     115$        ; If zero were finished
0000103C'EF40 FF 8F 91 01BC 1396          CMPB     #-1,SLOT[R0] ; Check array type
                                3C 13 01C5 1397          BEQL     105$        ; Branch if 256kb array
                                01C7 1398 ;
                                0000103C'EF40 01 91 01C7 1399          CMPB     #1,SLOT[R0] ;[3.0] Check array type
                                19 19 01CF 1400          BLSS     103$        ; Branch if 1024kb array
00000465'EF 00100000 8F C0 01D1 1401          ADDL     #^X100000,FREE_PAGE_P ; Set to base of row 4Mb array
00000465'EF 000FFFFFF 8F CA 01DC 1402          BICL2   #^XFFFFFF,FREE_PAGE_P ; Round off
                                01E7 1403 ;
                                00000465'EF 002A 31 01E7 1404          BRW     110$        ; Branch to end
00000465'EF 00040000 8F C0 01EA 1405 103$: ADDL     #^X40000,FREE_PAGE_P ; Set to base of row 1024kb array
00000465'EF 0003FFFF 8F CA 01F5 1406          BICL2   #^X3FFFF,FREE_PAGE_P ; Round off
                                0011 31 0200 1407          BRW     110$        ; Branch to end
00000465'EF 00010000 8F C0 0203 1408 105$: ADDL     #^X10000,FREE_PAGE_P ; Set to base of row 256kb array
                                00000465'EF B4 020E 1409          CLRW    FREE_PAGE_P ; Round off
00000449'EF 00000465'EF D1 0214 1410 110$: CMPL   FREE_PAGE_P,MAXLWA ; Test to see if any arrays
                                06 19 021F 1411          BLSS     115$        ; Branch if so
                                00000465'EF D4 0221 1412          CLRL     FREE_PAGE_P ; Set to signal no arrays
                                0227 1413 115$: $DS_ENDINIT
                                0227 INITIALIZE_X:
00010058 9F 6E FA 0227          CALLG   (SP), a#DS$BREAK
                                04 022E          RET
                                0000036E          .RESTORE
; RETURN TO DIAGNOSTIC SUPERVISOR

```

```

036E 1415 .SBTTL CLEAN UP ROUTINE
036E 1416 $DS_BGNCLEAN
036E .SAVE
00000000 .PSECT CLEANUP, LONG
0000 CLEAN_UP: .WORD ^M<> ; ENTRY MASK
0002 1417 ;++
0002 1418 ; FUNCTIONAL DESCRIPTION:
0002 1419 ;
0002 1420 ; This is the clean up routine. The machine check vector is returned to
0002 1421 ; the Diagnostic Supervisor.
0002 1422 ;
0002 1423 ; CALLING SEQUENCE:
0002 1424 ;
0002 1425 ; The Diagnostic Supervisor calls this routine at the end of every test
0002 1426 ; sequence.
0002 1427 ;
0002 1428 ; INPUT PARAMETERS: NONE
0002 1429 ;
0002 1430 ; IMPLICIT PARAMETERS: NONE
0002 1431 ;
0002 1432 ; OUTPUT PARAMETERS: NONE
0002 1433 ;
0002 1434 ; IMPLICIT OUTPUTS: NONE
0002 1435 ;
0002 1436 ; COMPLETION CODES: NONE
0002 1437 ;
0002 1438 ; SIDE EFFECTS: NONE
0002 1439 ;--
0002 1440
0002 1441 $DS_INITSCB_S ; Return all vectors
0002 1442 CALLS #0, a#DS$INITSCB
0009 1442 MOVL #CONTROLLER, R2 ; Get address of memory controller
0010 1443 MOVL #^XE0000000, CSR0(R2) ; Clear controller errors
0017 1444 CLRL CSR1(R2) ; And turn off any diagnostic modes. M2 M
001A 1445 ; This also disables single-bit error
001A 1446 ; interrupts.
001A 1447 ;
001A 1448 CLRL HIGHROW ;[3.0] housecleaning to prevent
0020 1449 CLRL HIGHADDR ; subsequent runs from using old data
0026 1450 CLRL LOWROW
002C 1451 CLRL LOWADDR
0032 1452 CLRL TEMP
0038 1453 CLRL TEMP1 ;[3.0]
003E 1454 CLRL ERR_LOG_FULL ;[3.1]
0044 1455 ;
0044 1456 MTPR #^XF, #^X26 ; Clear mach ck error summary register M2
0047 1457 MTPR #0, #CADR ; Enable cache
004A 1458 $DS_ENDCLEAN
004A 1458 CLEAN_UP_X:
004A 1458 CALLG (SP), a#DS$BREAK
0051 1458 RET ; RETURN TO DIAGNOSTIC SUPERVISOR
0000036E .RESTORE
036E 1459

```

```

036E 1461 .SBTTL SINGLE BIT ERROR INTERRUPT SERVICE ROUTINE
036E 1462 .ALIGN LONG
0370 1463 CORR_ERR_ISR::
0370 1464 ;++
0370 1465 ; FUNCTIONAL DESCRIPTION:
0370 1466 ;
0370 1467 ; This is the single-bit error interrupt service routine. If there is
0370 1468 ; a single bit error, this routine is invoked. It saves the address of
0370 1469 ; the error, and increments the correctable error counter by 1.
0370 1470 ;
0370 1471 ; CALLING SEQUENCE:
0370 1472 ;
0370 1473 ; This routine is invoked by a single-bit error interrupt.
0370 1474 ;
0370 1475 ; INPUT PARAMETERS: NONE
0370 1476 ;
0370 1477 ; IMPLICIT PARAMETERS:
0370 1478 ;
0370 1479 ; EXP_COR_ERR - flag indicating a correctable error is expected
0370 1480 ;
0370 1481 ; OUTPUT PARAMETERS: NONE
0370 1482 ;
0370 1483 ; IMPLICIT OUTPUTS:
0370 1484 ;
0370 1485 ; COR_ERRORS - an array which stores correctable error data
0370 1486 ; NUM_COR_ERR - a longword containing the number of correctable errors
0370 1487 ; which have occurred.
0370 1488 ;
0370 1489 ; COMPLETION CODES: NONE
0370 1490 ;
0370 1491 ; SIDE EFFECTS: NONE
0370 1492 ;
0370 1493 ;--
0370 1494

```

```

52 00F20000 8F BB 0370 1495 PUSHR #^M<R0,R1,R2,R3,R4,R5> ; Push R0,R1,R2,R3,R4,R5
30 0000001D'EF D0 0372 1496 MOVL #CONTROLLER,R2 ; Get address of memory controller
50 00000439'EF E8 0379 1497 BLBS EXP_COR_ERR,2$ ; If a correctable error not expected, M2
51 62 FF000180 8F D0 0380 1498 MOVL TOTAL_COR_ERR,R0 ; Then, get number of corrected errors M8
55 56 D0 0387 1499 BICL3 #^XFF000180,CSR0(R2),R1 ; Get error address and syndrome
038F 1500 MOVL R6,R5 ; Get virtual address form R6
0392 1501 ; CMPL #255,R0 ; Error log full? A2
0392 1502 ; BLEQ 20$ ; Branch if yes A2
0392 1503 ;
0392 1504 ; This is where the page address and syndrome that caused the single bit error
0392 1505 ; is recorded in the log
0392 1506 ;
53 D4 0392 1507 CLRL R3 ; Setup to scan current log for error
54 D4 0394 1508 CLRL R4 ; Used to check for empty or end of log
54 00000C3C'EF43 D1 0396 1509 1$: CMPL COR_ERRORS[R3],R4 ; Is this location in log empty?
13 13 039E 1510 BEQL 3$ ; Branch if yes A2
51 00000C3C'EF43 D1 03A0 1511 CMPL COR_ERRORS[R3],R1 ; Is it in the log?
17 13 03A8 1512 BEQL 10$ ; Branch if yes
53 01 C0 03AA 1513 ADDL #1,R3 ; Point to next location in log
FFE6 31 03AD 1514 BRW 1$ ; Scan the log for entry
0058 31 03B0 1515 2$: BRW 15$ ; ...
00000C3C'EF43 51 D0 03B3 1516 3$: MOVL R1,COR_ERRORS[R3] ; Put this entry in the log A2
00000031'EF D6 03BB 1517 INCL PG_COR_ERR ; Count the error A7

```

```
03C1 1518 ; BRB 10$
03C1 1519 ;5$: $DS_PRINTB_S FMT_LOG_FULL ; Type error log full message [3.1]
03C1 1520 ; $DS_ABORT TEST ; Error log full abort test
03C1 1521 ;
03C1 1522 ; This is where the actual Virtual Address that caused the single bit error
03C1 1523 ; is recorded
03C1 1524 ;
53 D4 03C1 1525 10$: CLRL R3 ; Init an index to scan log
54 D4 03C3 1526 CLRL R4 ; Used to check for empty or end of log
54 00000035'EF43 D1 03C5 1527 11$: CMPL VA_COR_ADDRESS[R3],R4 ; Is this location in log empty ?
10 13 03CD 1528 BEQL 12$ ; Branch if yes
55 00000035'EF43 D1 03CF 1529 CMPL VA_COR_ADDRESS[R3],R5 ; Is it in the log ?
2B 13 03D7 1530 BEQL 14$ ; Branch if yes
53 01 C0 03D9 1531 ADDL #1,R3 ; Point to next location in the log
FFE6 31 03DC 1532 BRW 11$ ; Scan the log for entry
00000035'EF43 55 D0 03DF 1533 12$: MOVL R5,VA_COR_ADDRESS[R3] ; Put this entry in the log
00000435'EF D6 03E7 1534 INCL NUM_COR_ERR ; Count the error for each row
00000439'EF D6 03ED 1535 INCL TOTAL_COR_ERR ; Count the total single bit errors
00000439'EF FF 8F 91 03F3 1536 CMPB #255,TOTAL_COR_ERR
07 12 03FB 1537 BNEQ 14$
00000441'EF 01 90 03FD 1538 MOVB #1,ERR_LOG_FULL
62 20000000 8F C8 0404 1539 14$: BISL #^X20000000,CSR0(R2) ; Turn off the single bit error bit
50 17 DB 040B 1540 15$: MFPR #^X17,R0 ; Get contents of Bus Error Reg A6
0000002D'EF 50 FFFFFFFF0 8F CB 040E 1541 BICL3 #^XFFFFFFF0,R0,BUS_ERR_REG ; Save A6
3F BA 041A 1542 POPR #^M<R0,R1,R2,R3,R4,R5> ; Pop the registers
041C 1543
041C 1544 CORR_ERR_ISRX:
03 0000001D'EF 10 E0 041C 1545 BBS #16,EXP_COR_ERR,20$ ; Branch if inhibit resetting BUS ERROR A2
26 0F DA 0424 1546 MTPR #^XF,#^X26 ; Reset bus errors M2
0000001D'EF D4 0427 1547 20$: CLRL EXP_COR_ERR ; Cleanup the expected flag A2
02 042D 1548 REI ; Return
```

```
042E 1550      .SBTTL  MACHINE CHECK SERVICE ROUTINE
042E 1551      .ALIGN  LONG                      ; Align exception handler on longword
0430 1552
0430 1553 MACHINECK::
0430 1554 ;++
0430 1555 ; FUNCTIONAL DESCRIPTION:
0430 1556 ;
0430 1557 ;      This is the machine check service routine.  It examines the error code
0430 1558 ;      in order to ascertain what type of machine check has occurred.  For
0430 1559 ;      all but two types of machine checks, the routine merely reports
0430 1560 ;      the type of fault, cleans up the stack, and returns.
0430 1561 ;
0430 1562 ;      For bus error aborts, the routine cases on the type of bus error.
0430 1563 ;
0430 1564 ;      For non-existent memory errors, the routine checks the flag
0430 1565 ;      "EXP_NX_MEM".  If it is set, the routine clears it, sets up
0430 1566 ;      the desired return PC, clears the CPU bus error register, and
0430 1567 ;      returns.  If it is clear, the routine reports the fault and
0430 1568 ;      aborts the test sequence.
0430 1569 ;
0430 1570 ;      For uncorrectable errors, the routine checks the "EXP_UNC_ERR"
0430 1571 ;      flag.  If it is set, the routine clears it, sets up the
0430 1572 ;      desired return PC, clears the CPU bus error register, and
0430 1573 ;      returns.  If it is clear, the routine reports the fault and
0430 1574 ;      aborts the test sequence.
0430 1575 ;
0430 1576 ;      For information lost errors, the routine checks the flag
0430 1577 ;      "EXP_INF_LST".  If it is set, the routine clears it, sets up
0430 1578 ;      the desired return PC, clears the CPU bus error register, and
0430 1579 ;      returns.  If it is clear, the routine reports the fault and
0430 1580 ;      aborts the test sequence.
0430 1581 ;
0430 1582 ; CALLING SEQUENCE:
0430 1583 ;
0430 1584 ;      This routine is invoked as a result of a machine check.
0430 1585 ;
0430 1586 ; INPUT PARAMETERS:      None
0430 1587 ;
0430 1588 ; IMPLICIT PARAMTERS:
0430 1589 ;
0430 1590 ;      EXP_UNC_ERR = flag to indicate uncorrectable error expected
0430 1591 ;      EXP_INF_LST = flag to indicate information lost error expected
0430 1592 ;      EXP_NX_MEM  = flag to indicate non-existent memory reference expected
0430 1593 ;      all flags = 1 if true, = 0 if false
0430 1594 ;
0430 1595 ; OUTPUT PARAMETERS:      None
0430 1596 ;
0430 1597 ; IMPLICIT OUTPUTS:      None
0430 1598 ;
0430 1599 ; COMPLETION CODES:
0430 1600 ;
0430 1601 ;      All 'expected' flags are cleared.
0430 1602 ;
0430 1603 ; SIDE EFFECTS: NONE
0430 1604 ;
0430 1605 ;--
0430 1606
```

```

50 04 AE D0 0430 1607 MOVL 4(SP),R0 ; Get error code
07 00 50 8F 0434 1608 CASEB R0,#0,#7 ; Determine type of machine check M2
0010' 0438 1609 20$: .WORD 100$-20$ ; Displacement for error code = 0
001F' 043A 1610 .WORD 200$-20$ ; Displacement for error code = 1
0034' 043C 1611 .WORD 300$-20$ ; Displacement for error code = 2
012E' 043E 1612 .WORD 400$-20$ ; Displacement for error code = 3
0143' 0440 1613 .WORD 500$-20$ ; Displacement for error code = 4
0158' 0442 1614 .WORD 600$-20$ ; Displacement for error code = 5
0158' 0444 1615 .WORD 600$-20$ ; Displacement for error code = 6 M2
0158' 0446 1616 .WORD 600$-20$ ; Displacement for error code = 7 A2
5E DD 0448 1617 100$: PUSHL SP
FDD7 CF 01 FB 044A 1618 CALLS #1,PRINT_UNX_TO
5E 8E C0 044F 1619 ADDL (SP)+,SP ; Clean up stack
50 D4 0452 1620 CLRL R0 ;
015A 31 0454 1621 BRW 1000$ ; And branch A2
0457 1622 200$: $DS_PRINTF S FMT_CS_MC ; Report control store interrupt
00001D01'EF 9F 0457 PUSHAB FMT_CS_MC
000100F0 9F 01 FB 045D CALLS $$$N, a#DS$PRINTF
5E 8E C0 0464 1623 ADDL (SP)+,SP ; Clean up stack
50 D4 0467 1624 CLRL R0 ; A2
0145 31 0469 1625 BRW 1000$ ; And branch
50 D4 046C 1626 300$: CLRL R0 ; A2
0000002D'EF 24 AE D0 046E 1627 MOVL 36(SP),BUS_ERR_REG ; Get cpu bus error register M2
2F 00000025'EF E8 0476 1628 BLBS EXP_INF_LST,310$ ; Branch if expected information lost error
5D 00000021'EF E8 047D 1629 BLBS EXP_UNC_ERR,340$ ; Branch if expected uncorrectable error M2
03 00000029'EF E9 0484 1630 BLBC EXP_NX_MEM,302$ ; Branch if not expected non-existent M2
048B 1631 ; memory error M2
0099 31 048B 1632 BRW 370$ ; Expected non-existent mem error A2
29 0000002D'EF 01 E0 048E 1633 302$: BBS #1,BUS_ERR_REG,315$ ; Branch if info lost error A2
03 0000002D'EF 03 E1 0496 1634 BBC #3,BUS_ERR_REG,303$ ; Branch if not nxm error a2
0099 31 049E 1635 BRW 375$ ; Nxm error A2
4B 0000002D'EF 02 E0 04A1 1636 303$: BBS #2,BUS_ERR_REG,341$ ; Branch if uncorrectable error A2
FF9C 31 04A9 1637 BRW 100$ ; Unknown machine check A2
1A 00000025'EF 08 E0 04AC 1638 310$: BBS #8,EXP_INF_LST,320$ ; Branch if ignore BUS ERR reg M2
12 0000002D'EF 01 E0 04B4 1639 BBS #1,BUS_ERR_REG,320$ ; Branch if information lost error M2
FF89 31 04BC 1640 BRW 100$ ; Unknown machine check A2
5E DD 04BF 1641 315$: PUSHL SP
FD60 CF 01 FB 04C1 1642 CALLS #1,PRINT_UNX_TO
5E 8E C0 04C6 1643 ADDL (SP)+,SP ; Clean up stack
50 D4 04C9 1644 CLRL R0 ; A2
00E3 31 04CB 1645 BRW 1000$ ; And branch
00000025'EF D4 04CE 1646 320$: CLRL EXP_INF_LST ; Clear the expected flag A2
5E 8E C0 04D4 1647 ADDL (SP)+,SP ; Point SP to PC
6E 0000000C'EF D0 04D7 1648 MOVL RETURN_PC,(SP) ; Set up to return to desired location
00D0 31 04DE 1649 BRW 1000$ ; And branch
21 00000021'EF 08 E0 04E1 1650 340$: BBS #8,EXP_UNC_ERR,350$ ; Branch if ignore BUS ERROR reg A2
19 0000002D'EF 02 E0 04E9 1651 BBS #2,BUS_ERR_REG,350$ ; Branch if uncorrectable error M2
FF54 31 04F1 1652 BRW 100$ ; Unknown machine check A2
5E DD 04F4 1653 341$: PUSHL SP
FD2B CF 01 FB 04F6 1654 CALLS #1,PRINT_UNX_TO
5E 8E C0 04FB 1655 ADDL (SP)+,SP ; Point SP to PC
50 D4 04FE 1656 CLRL R0 ; A2
6E 00000010'EF D0 0500 1657 MOVL UNX_UNC_PC,(SP) ; Set the return PC A7
00A7 31 0507 1658 BRW 1000$ ; And branch
02 00000021'EF 10 E1 050A 1659 350$: BBC #16,EXP_UNC_ERR,360$ ; Branch if not inhibit clearing BUS A2
0512 1660 ; ERROR register A2
50 D6 0512 1661 INCL R0 ; A2

```

	5E	8E	C0	0514	1662	360\$:	ADDL	(SP)+,SP		; Point SP to PC	
	00000021	'EF	D4	0517	1663		CLRL	EXP_UNC_ERR		; Cleanup the expected error flag	A2
6E	0000000C	'EF	D0	051D	1664		MOVL	RETURN_PC,(SP)		; Set up to return to desired location	
		008A	31	0524	1665		BRW	1000\$; And branch	
1A	00000029	'EF	E0	0527	1666	370\$:	BBS	#8,EXP_NX_MEM,380\$; Branch if ignore BUS ERROR reg	M2
12	0000002D	'EF	E0	052F	1667		BBS	#3,BUS_ERR_REG,380\$; Branch if NXM error	A2
		FF0E	31	0537	1668		BRW	100\$; Unknown machine check	A2
		5E	DD	053A	1669	375\$:	PUSHL	SP			
	FCE5	CF	01	FB	053C	1670	CALLS	#1,PRINT_UNX_TO			
		5E	C0	0541	1671		ADDL	(SP)+,SP		; Point SP to PC	
			D4	0544	1672		CLRL	R0			A2
			31	0546	1673		BRW	1000\$; And branch	
02	00000029	'EF	E1	0549	1674	380\$:	BBC	#16,EXP_NX_MEM,390\$; Branch if not inhibit clearing BUS	A2
				0551	1675					; ERROR register	A2
			D6	0551	1676		INCL	R0			A2
		5E	C0	0553	1677	390\$:	ADDL	(SP)+,SP		; Point SP to PC	
	00000029	'EF	D4	0556	1678		CLRL	EXP_NX_MEM		; Cleanup the expected flag	A2
6E	0000000C	'EF	D0	055C	1679		MOVL	RETURN_PC,(SP)		; Set up to return to desired location	
		004B	31	0563	1680		BRW	1000\$; And branch	
				0566	1681	400\$:	\$DS_PRINTF_S	FMT_CPE_MC		; Report cache parity error	
	00001D73	'EF	9F	0566			PUSHAB	FMT_CPE_MC			
	000100F0	9F	01	FB	056C		CALLS	###N, a#DS\$PRINTF			
		5E	C0	0573	1682		ADDL	(SP)+,SP		; Clean up stack	
			D4	0576	1683		CLRL	R0			A2
			31	0578	1684		BRW	1000\$; And branch	
				057B	1685	500\$:	\$DS_PRINTF_S	FMT_WBE_MC		; Report write bus error	
	0000212D	'EF	9F	057B			PUSHAB	FMT_WBE_MC			
	000100F0	9F	01	FB	0581		CALLS	###N, a#DS\$PRINTF			
		5E	C0	0588	1686		ADDL	(SP)+,SP		; Clean up stack	
			D4	058B	1687		CLRL	R0			A2
			31	058D	1688		BRW	1000\$; And branch	
				0590	1689	600\$:	\$DS_PRINTF_S	FMT_NSBH		; Report 'not supposed to be here'	
	00001DA7	'EF	9F	0590			PUSHAB	FMT_NSBH			
	000100F0	9F	01	FB	0596		CALLS	###N, a#DS\$PRINTF			
		5E	C0	059D	1690		ADDL	(SP)+,SP		; Point SP to PC	
			D4	05A0	1691		CLRL	R0			A2
			31	05A2	1692		BRW	1000\$; And branch	
		5E	C0	05A5	1693	610\$:	ADDL	(SP)+,SP		; Prepare for REI	
6E	0000000C	'EF	D0	05A8	1694		MOVL	RETURN_PC,(SP)		; Set up to return to desired location	
			D4	05AF	1695		CLRL	R0			A2
				05B1	1696	1000\$:					
				05B1	1697	MACHINECKX:					
		5E	D5	05B1	1698		TSTL	R0		; Inhibit clearing BUS ERROR?	A2
		03	12	05B3	1699		BNEQ	10\$; Branch if yes	A2
	26	0F	DA	05B5	1700		MTPR	#^XF,#^X26		; Reset bus errors	M2
			02	05B8	1701	10\$:	REI			; And return	

05B9 1703 .SBTTL READ MEMORY CONFIGURATION MAP ROUTINE

05B9 1704 READ_MAP::

05B9 1705 ;**

05B9 1706 ; FUNCTIONAL DESCRIPTION:

05B9 1707 ;

05B9 1708 ;

05B9 1709 ;

05B9 1710 ;

05B9 1711 ;

05B9 1712 ;

05B9 1713 ;

05B9 1714 ;

05B9 1715 ;

05B9 1716 ;

05B9 1717 ;

05B9 1718 ;

05B9 1719 ;

05B9 1720 ;

05B9 1721 ;

05B9 1722 ;

05B9 1723 ;

05B9 1724 ;

05B9 1725 ;

05B9 1726 ;

05B9 1727 ;

05B9 1728 ;

05B9 1729 ;

05B9 1730 ;

05B9 1731 ;

05B9 1732 ;

05B9 1733 ;

05B9 1734 ;

05B9 1735 ;

05B9 1736 ;

05B9 1737 ;

05B9 1738 ;

05B9 1739 ;

05B9 1740 ;

05B9 1741 ;

05B9 1742 ;

05B9 1743 ;

05B9 1744 ;

05B9 1745 ;

05B9 1746 ;

05B9 1747 ;

05B9 1748 ;

05B9 1749 ;

05B9 1750 ;

05B9 1751 ;

05B9 1752 ;

05B9 1753 ;

05B9 1754 ;

05B9 1755 ;

05B9 1756 ;

05B9 1757 ;

05B9 1758 ;

05B9 1759 ;

The memory map consists of 8 2-bit cells. The code contained in these cells represents the contents of the corresponding memory subsystem backplane slot. The code differs for each type of controller, the code for the L0011 is as follows:

- 0 = empty
- 1 = 128kb array not supported in this version of ECKAM
- 2 = register error
- 3 = 256kb array

L0016:

- 0 = empty
- 1 = register error
- 2 = 1024kb array
- 3 = 256kb array

L0022:

- 0 = empty
- 1 = 4Mb array
- 2 = 1Mb array
- 3 = register error

[3.0]

For example, a L0016 controller with a 1024kb array in slot 0, and a 256kb array in slots 1 and 2, and the rest of the slots empty would be indicated by the following map:

CSR 2

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0

The register is scanned 2 bits at a time, starting at cell 0. An error is caused by any 3 following a 0 or 1, or any 2 following a 0, or any 3.

CALLING SEQUENCE:

JSB

INPUT PARAMETERS: NONE

OUTPUT PARAMETERS: NONE

IMPLICIT PARAMETERS: NONE

COMPLETION CODES: NONE

```

05B9 1760 ; SIDE EFFECTS:          NONE
05B9 1761 ;--
0FB7 8F  BB 05B9 1762          PUSHR  #^M<R0,R1,R2,R4,R5,R7,R8,R9,R10,R11> ; Save registers
05BD 1763 ;-
52 00F20000 8F  D0 05BD 1764          MOVL  #CONTROLLER,R2          ; Get controller address
50          D4 05C4 1765          CLRL  R0                    ; Register error flag = 0
51          D4 05C6 1766          CLRL  R1                    ; Slot number = 0
54          D4 05C8 1767          CLRL  R4                    ; Cell count for 1024kb arrays
55          D4 05CA 1768          CLRL  R5                    ; Cell count for 4Mb arrays
57          D4 05CC 1769          CLRL  R7                    ; Empty cell count (ecc) = 0
58          D4 05CE 1770          CLRL  R8                    ; 128kb flag (not supported)
59          D4 05D0 1771          CLRL  R9                    ; Cell count for 256kb arrays
                                ; or 4 Mb          [3.0]
                                ; Cell counter (cc) = 0
5A          D4 05D2 1772          CLRL  R10                   ; Illegal configuration flag (icf) = 0
5B          D4 05D4 1773          CLRL  R11                   ; Get a cell
53 08 A2 02 5A  EF 05D6 1775 10$:  EXTZV  R10,#2,CSR2(R2),R3    ; Test the cell
03 00 53  8F 05DC 1776          CASEB  R3,#0,#3
                                ; Displacement for cell = 0
0008' 05E0 1777 20$:  .WORD  100$-20$
001A' 05E2 1778          .WORD  200$-20$
0041' 05E4 1779          .WORD  300$-20$
005E' 05E6 1780          .WORD  400$-20$
                                ; Displacement for cell = 1
                                ; Displacement for cell = 2
                                ; Displacement for cell = 3
57          D6 05E8 1781 100$:  INCL  R7
0000103C'EF41 5A 94 05EA 1782          CLRB  SLOT[R1]
5A          95 05F1 1783          TSTB  R10
03          12 05F3 1784          BNEQ  110$
50          01 90 05F5 1785          MOVB  #1,R0
16 08 A2 65 11 05F8 1786 110$:  BRB   500$
18          E1 05FA 1787 200$:  BBC   #24,CSR2(R2),205$
                                ;
0C 08 A2 19  E1 05FF 1789          BBC   #25,CSR2(R2),203$
55          D6 0604 1790          INCL  R5
0000103C'EF41 01 90 0606 1791          MOVB  #1,SLOT[R1]
4F          11 060E 1792          BRB   500$
                                ;
50          01 90 0610 1793          ;
50          01 90 0610 1794 203$:  MOVB  #1,R0
4A          11 0613 1795          BRB   500$
                                ; Set register error
                                ; Return to normal flow
58          D6 0615 1796 205$:  INCL  R8
0000103C'EF41 01 90 0617 1797          MOVB  #1,SLOT[R1]
3E          11 061F 1798          BRB   500$
                                ; Set not supported flag
                                ; Indicate this slot not supported
13 08 A2 18  E1 0621 1799 300$:  BBC   #24,CSR2(R2),305$
                                ; Else, return to common flow
                                ; Check controller type
0000103C'EF41 54  D6 0626 1800          INCL  R4
                                ; Bump counter for 1024kb
02          90 0628 1801          MOVB  #2,SLOT[R1]
59          95 0630 1802          TSTB  R9
                                ; Indicate this slot is 1024kb
26          13 0632 1803          BEQL  410$
                                ; If 256kb count equals 0
5B          01 90 0634 1804          MOVB  #1,R11
26          11 0637 1805          BRB   500$
                                ; Then branch
                                ; Else set illegal configuration flag
50          01 90 0639 1806 305$:  MOVB  #1,R0
21          11 063C 1807          BRB   500$
                                ; Return to common flow
                                ; Set register error
                                ; Else, return to common flow
05 08 A2 19  E1 063E 1808          ;
50          01 90 0643 1810          BBC   #25,CSR2(R2),403$
17          11 0646 1811          MOVB  #1,R0
                                ; [3.0] For 4Mb array
                                ; Set register error
                                ; Return to normal flow
0000103C'EF41 59  D6 0648 1812          ;
FF 8F 90 064A 1814 403$:  INCL  R9
57          95 0653 1815          MOVB  #-1,SLOT[R1]
03          13 0655 1816          TSTB  R7
                                ; Fpc = fpc + 1
                                ; Indicate this slot 256kb
                                ; If empty cell count neq 0
                                ; Then branch

```

```

          5B 01 90 0657 1817      MOVB    #1,R11      ; Else, set illegal configuration flag
          03 11 065A 1818 410$: BRB     500$          ; Then branch
          5B 01 90 065C 1819      MOVB    #1,R11      ; Else, set illegal configuration flag
          51 D6 065F 1820 500$: INCL   R1             ; Add one to slot counter and then
FF6F 5A 02 0E F1 0661 1821      ACBL   #14,#2,R10,10$ ; Get all eight cells in turn
          53 08 A2 3C 0667 1822      MOVZWL CSR2(R2),R3 ; Get the memory map
          50 95 066B 1823      TSTB   R0          ; Check for register error
          1B 13 066D 1824      BEQL   510$        ; Branch if not
          066F 1825      SDS_PRINTB_S    FMT_REG_ERR,R3 ; This is a register error
          53 DD 066F          PUSHL   R3
          00002185'EF 9F 0671          PUSHAB FMT_REG_ERR
          000100E0 9F 02 FB 0677          CALLS  $$$N, a#DS$PRINTB
          067E 1826      SDS_CKLOOP  10$          ; Scope loop?
          00010040 9F FF54 CF FA 067E 1826      CALLG  10$, a#DS$CKLOOP
          002B 31 0687 1827      BRW    600$        ; And exit
          5B 95 068A 1828 510$: TSTB   R11          ; Check for illegal configuration set
          0F 13 068C 1829      BEQL   550$        ; Branch if not
          068E 1830      SDS_PRINTB_S    FMT_ILL_CFG,R3 ; Print the error
          53 DD 068E          PUSHL   R3
          0000213F'EF 9F 0690          PUSHAB FMT_ILL_CFG
          000100E0 9F 02 FB 0696          CALLS  $$$N, a#DS$PRINTB
          58 95 069D 1831 550$: TSTB   R8          ; Check for not supported flag
          14 13 069F 1832      BEQL   600$        ; Branch if not
          06A1 1833      SDS_PRINTB_S    FMT_NO_SUPP ; Type no support message
          000012F2'EF 9F 06A1          PUSHAB FMT_NO_SUPP
          000100E0 9F 01 FB 06A7          CALLS  $$$N, a#DS$PRINTB
          06AE 1834      SDS_ABORT   PROGRAM ; Leave program
          00010020 9F 6C FA 06AE          CALLG  (AP), a#DS$ABORT
          06B5 1835 ;
          06B5 1836 ; Here the ROW configuration, TRANSITION, and TEST_ADDRESS tables are built:
          06B5 1837 ;
          50 D4 06B5 1838 600$: CLRL   R0          ; Initialize slot counter
          52 D4 06B7 1839      CLRL   R2          ; Counter for slots with arrays
          0000103C'EF40 95 06B9 1840 610$: TSTB   SLOT[R0] ; Check slot configuration
          12 13 06C0 1841      BEQL   620$        ; Branch if slot[R0] = 0
          53 52 04 C5 06C2 1842      MULL3  #4,R2,R3 ; Multiply number of slots by 4
          00001048'EF40 53 03 81 06C6 1843      ADDB3  #3,R3,ROWCON[R0] ; Add # of rows to account for x 0
          52 D6 06CF 1844      INCL   R2          ; Add one to array counter
          00001048'EF40 94 06D4 1846 620$: CLRB   ROWCON[R0] ; Store a zero as this slot is empty
          0000 31 06DB 1847      BRW    650$        ; Then branch
          D7 50 07 F3 06DE 1848 650$: AOBLEQ #7,R0,610$ ; Do for all slots
          00400000 8F 55 C5 06E2 1849      MULL3  R5,#^X400000,- ;
          0000106C'EF          06E9 1850      TRANSITION4 ;[3.0] Used in tests 3 and 4 to find
          06EE 1851 ; end of 4Mb arrays.
          00001068'EF 00100000 8F 54 C5 06EE 1852      MULL3  R4,#^X100000,TRANSITION ; Used in tests 3 and 4 to find end of-
          06FA 1853 ; 1024kb arrays.
          00001068'EF 0000106C'EF C0 06FA 1854      ADDL2  TRANSITION4,TRANSITION ; Calculate total address of 1024kb and-
          0705 1855 ; 256kb arrays.
          0705 1856 ;
          00000451'EF 00040000 8F 59 C5 0705 1857      MULL3  R9,#^X40000,TEST_ADDRESS ; Calculate total address of 256kb arrays
          00000451'EF 00001068'EF C0 0711 1858      ADDL2  TRANSITION,TEST_ADDRESS ; Calculate total address of 1024kb and-
          071C 1859 ; 256kb arrays.
          00000451'EF 04 C2 071C 1860      SUBL2  #4,TEST_ADDRESS ; Calculate a longword address used in-
          0723 1861 ; Test 1
          00001068'EF 01 C2 0723 1862      SUBL2  #1,TRANSITION ; Subtract one to stay in last array-
          072A 1863 ; Used in Test 3,4 when there is a mixture

```

ZZ-ECKAM-3.2
ECKAM
03-02

READ MEMORY CONFIGURATION MAP ROUTINE
MEMORY DIAGNOSTIC HEADER
READ MEMORY CONFIGURATION MAP ROUTINE

K 6

13-MAY-1987

Fiche 1 Frame K6

Sequence 75

23-JAN-1987 14:30:12

23-JAN-1987 14:26:49

VAX/VMS Macro V04-00

Page 52

[ZECCHINO.ECKAM]ECKAM0.MAR;4

(17)

```
0000106C'EF 01 C2 072A 1864          SUBL2  #1,TRANSITION4          ;[3.0] Subtract one to stay in last array-  
              0731 1865          ; Used in Test 3,4 when there is a mixture  
              0731 1866 ;  
              55 95 0731 1867          TSTB  R5          ;[3.0] Set flag if 4Mb arrays  
              07 13 0733 1868          BEQL  670$         ;  
00001060'EF 01 90 0735 1869          MOVB  #1,FLAG4         ; are present  
              073C 1870 ;  
              54 95 073C 1871 670$: TSTB  R4          ;[3.0] Set flag if 1024kb arrays  
              07 13 073E 1872          BEQL  680$         ;  
0000105C'EF 02 90 0740 1873          MOVB  #2,FLAG3         ; are present  
              0747 1874 ;  
              59 95 0747 1875 680$: TSTB  R9          ; Is there any 256kb arrays  
              08 13 0749 1876          BEQL  READ_MAPX    ; Branch to end if not  
00001058'EF FF 8F 90 074B 1877          MOVB  #-1,FLAG2       ; Used in test 3 to flag 256kb arrays  
              0753 1878  
              0753 1879 READ_MAPX:  
0FB7 8F BA 0753 1880          POPR  #^M<R0,R1,R2,R4,R5,R7,R8,R9,R10,R11> ;Restore registers  
              0757 1881 ;-  
              05 0757 1882          RSB          ; Return
```

```

0758 1884 .SBTTL GET ARRAY CARDS TO TEST ROUTINE
0758 1885 GET_ARRAYS::
0758 1886 ;++
0758 1887 ; FUNCTIONAL DESCRIPTION:
0758 1888 ;
0758 1889 ; This routine asks the operator which array cards to test in the
0758 1890 ; moving inversions (manual array select) test. The routine first
0758 1891 ; determines which arrays are available to test. The operator is
0758 1892 ; informed of his choices, and then asked to enter the low array
0758 1893 ; and high array numbers. Only contiguous array cards may be tested.
0758 1894 ; The operator will receive an error message if he chooses a
0758 1895 ; non-existent array card to test, or if he selects for the high card
0758 1896 ; a number lower than that for the low card.
0758 1897 ;
0758 1898 ; CALLING SEQUENCE:
0758 1899 ;
0758 1900 ; JSB
0758 1901 ;
0758 1902 ; INPUT PARAMETERS: NONE
0758 1903 ;
0758 1904 ; IMPLICIT INPUTS:
0758 1905 ;
0758 1906 ; ROWCON - an eight-byte vector containing the highest row number
0758 1907 ; contained on each array card.
0758 1908 ;
0758 1909 ; SLOT - an eight-byte vector containing information about the
0758 1910 ; array card residing in each slot. Each vector element
0758 1911 ; contains a zero if the slot is empty, a one if it is a
0758 1912 ; 4Mb array, a two if it is a 1024kb array or a -1 if it
0758 1913 ; is a 256kb array.
0758 1914 ;
0758 1915 ; OUTPUT PARAMETERS: NONE
0758 1916 ;
0758 1917 ; IMPLICIT OUTPUTS:
0758 1918 ;
0758 1919 ; LOWROW - longword containing the lowest row number to test
0758 1920 ; LOWADDR - longword containing the lowest address to test
0758 1921 ; HIGHROW - longword containing the highest row number to test
0758 1922 ; HIGHADDR - longword containing the highest longword address to test
0758 1923 ;
0758 1924 ; COMPLETION CODES:
0758 1925 ;
0758 1926 ; R2 - 1 if success, 0 if failure (no arrays to test)
0758 1927 ;
0758 1928 ; SIDE EFFECTS: NONE
0758 1929 ;--

```

```

00F8 8F BB 0758 1931 PUSHR #^M<R3,R4,R5,R6,R7>
53 D4 075C 1932 CLRL R3 ; Initialize slot counter
52 D4 075E 1933 CLRL R2 ; Status pointer
00000824'EF D4 0760 1934 CLRL TEMP ; Used for storage and status
00001064'EF D4 0766 1935 CLRL COUNTER ; Used for counting 256kb arrays
0000103C'EF43 95 076C 1936 10$: TSTB SLOT[R3] ; Is this the highest slot
04 13 0773 1937 BEQL 20$ ; Branch if so
F3 53 07 0775 1938 AOBLEQ #7,R3,10$ ; Bump the slot counter and try again
53 D7 0779 1939 20$: DECL R3 ; Back up count to point to high array
52 00000465'EF D0 077B 1940 MOVL FREE_PAGE_P,R2 ; Get address of first free row

```

```

      55 D4 0782 1941          CLRL R5          ; Initialize array index
      56 D4 0784 1942          CLRL R6
0000103C'EF45 FF 8F 91 0786 1943      CMPB #-1,SLOT[R5]      ; Find array type in slot
      24 13 078F 1944          BEQL 27$              ; Branch if 256kb array
      0791 1945 ;
0000103C'EF45 02 91 0791 1946      CMPB #2,SLOT[R5]      ;[3.0] Find array type in slot
      0D 13 0799 1947          BEQL 25$
54 52 03 16 EF 079B 1948      EXTZV #22,#3,R2,R4    ; Get lowest array possible 4MB
52 52 02 14 EF 07A0 1949      EXTZV #20,#2,R2,R2    ; Get row number of base address
      0017 31 07A5 1950          BRW 28$              ; Continue flow
      07A8 1951 ;
54 52 03 14 EF 07A8 1952 25$:    EXTZV #20,#3,R2,R4    ; Get lowest array possible
52 52 02 12 EF 07AD 1953      EXTZV #18,#2,R2,R2    ; Get row number of base address
      000A 31 07B2 1954          BRW 28$              ; Continue flow
54 52 03 12 EF 07B5 1955 27$:    EXTZV #18,#3,R2,R4    ; Get lowest array 1024kb
52 52 02 10 EF 07BA 1956      EXTZV #16,#2,R2,R2    ; Get row number of base address
      07BF 1957 28$:          $DS_PRINTF S FMT_AVAILABLE,R4,R3 ; Report on the available arrays
      53 DD 07BF          PUSHL R3
      54 DD 07C1          PUSHL R4
00001339'EF 9F 07C3          PUSHAB FMT_AVAILABLE
000100F0 9F 03 FB 07C9          CALLS $$$N, a#DS$PRINTF
      07D0 1958          $DS_PRINTF S FMT_ASK_LOW ; This is the low question message
0000139B'EF 9F 07D0          PUSHAB FMT_ASK_LOW
000100F0 9F 01 FB 07D6          CALLS $$$N, a#DS$PRINTF
      07DD 1959          $DS_ASKDATA S ASK,TEMP,#10,,R4,R3,R4
      00 DD 07DD          PUSHL #0
      00 DD 07DF          PUSHL #0
      54 DD 07E1          PUSHL R4
      53 DD 07E3          PUSHL R3
      54 DD 07E5          PUSHL R4
      FFFFFFFF 8F DD 07E7          PUSHL #-1
      0A DD 07ED          PUSHL #10
00000824'EF DF 07EF          PUSHAL TEMP
000013B1'EF DF 07F5          PUSHAL ASK
00010080 9F 09 FB 07FB          CALLS #9, a#DS$ASKDATA
      0802 1960          ; Ask him for the low one M4
54 00000824'EF D0 0802 1961          MOVL TEMP,R4        ; Put the low array number in R4
      58 54 D0 0809 1962          MOVL R4,R8          ; Put the low array number in R8
00000824'EF D4 080C 1963          CLRL TEMP          ; Initialize temp storage again
0000103C'EF44 FF 8F 91 0812 1964      CMPB #-1,SLOT[R4]    ; Find out what kind of board here
      35 13 081B 1965          BEQL 29$            ; Branch if 256KB array
      081D 1966 ;
0000103C'EF46 01 91 081D 1967      CMPB #1,SLOT[R6]    ;[3.0] Branch if no 4Mb arrays
      56 12 0825 1968          BNEQ 34$
      0827 1969 ;
      0827 1970 ; Case of 4Mb ar 4Mb/1024kb combination:
      0827 1971 ;
00000455'EF 00001048'EF44 9A 0827 1972          MOVZBL ROWCON[R4],LOWROW ;[3.0] Get starting row
00000455'EF 03 C2 0833 1973          SUBL2 #3,LOWROW
      54 95 083A 1974          TSTB R4
      17 12 083C 1975          BNEQ 30$
      00000465'EF D0 083E 1976          MOVL FREE_PAGE_P,- ; Lowest possible address
0000045D'EF 0844 1977          LOWADDR ; when startin at slot 0
00000455'EF 52 D0 0849 1978          MOVL R2,LOWROW
      5B 11 0850 1979          BRB 37$
      0065 31 0852 1980 29$:    BRW 40$
      0855 1981 ;
```

```

0000103C'EF46 01 91 0855 1982 ; Get starting row for 4Mb or 4Mb/1024kb
0000045D'EF 00400000 8F C0 0855 1983 ;
0000045D'EF 00100000 8F C0 0855 1984 30$: CMPB #1,SLOT[R6] ; If not 4Mb then branch
DA 56 54 F2 085D 1985 BNEQ 32$
0000045D'EF 00000455'EF 08 11 085F 1986 ADDL2 #^X400000,LOWADDR ; Case of 4mb array
0000045D'EF 0000045D'EF 0B 11 086A 1987 BRB 33$
0000045D'EF 00000465'EF 08 11 086C 1988 32$: ADDL2 #^X100000,LOWADDR ; Case of 1024kb array
0000045D'EF 00000465'EF 03 15 0877 1989 33$: AOBLS R4,R6, 30$
0000045D'EF 00000465'EF 03 15 087B 1990 BRB 37$
0000045D'EF 00000465'EF 03 15 087D 1991 ;
0000045D'EF 00000465'EF 03 15 087D 1992 ; Case of 1024kb array:
0000045D'EF 00000465'EF 03 15 087D 1993 ;
0000045D'EF 00000465'EF 03 15 087D 1994 34$: MOVZBL ROWCON[R4],R7 ; Get highest row of slot
0000045D'EF 00000465'EF 03 15 0885 1995 SUBL3 #3,R7,LOWROW ; Move to first row of array
0000045D'EF 00000465'EF 03 15 088D 1996 35$: MULL3 LOWROW,#^X40000,LOWADDR ; Calculate low address
0000045D'EF 00000465'EF 03 15 089D 1997 36$: CMPL FREE_PAGE_P,LOWADDR ; Find out where to start testing
0000045D'EF 00000465'EF 03 15 08A8 1998 BLEQ 37$ ; Branch if free page is < or =
0000045D'EF 00000465'EF 03 15 08AA 1999 BRW 38$ ; . . .
0000045D'EF 00000465'EF 03 15 08AD 2000 37$: BRW 55$ ; . . .
0000045D'EF 00000465'EF 03 15 08B0 2001 38$: ADDL2 R2,LOWROW ; Add the # of rows used by free page
0000045D'EF 00000465'EF 03 15 08B7 2002 BRW 35$ ; Branch to calculate a new address
0000045D'EF 00000465'EF 03 15 08BA 2003 40$: MOVZBL ROWCON[R4],R7 ; Get the highest row #in this slot
0000045D'EF 00000465'EF 03 15 08C2 2004 SUBL3 #3,R7,LOWROW ; Move to first row in the array
0000045D'EF 00000465'EF 03 15 08CA 2005 ;
0000045D'EF 00000465'EF 03 15 08CA 2006 42$: CMPB #2,SLOT[R5] ; Any 1024kb arrays ?
0000045D'EF 00000465'EF 03 15 08D2 2007 BEQL 50$ ; Branch if so
0000045D'EF 00000465'EF 03 15 08D4 2008 TSTB R5 ; Is there ONLY 256kb arrays ?
0000045D'EF 00000465'EF 03 15 08D6 2009 BEQL 45$ ; Branch if yes
0000045D'EF 00000465'EF 03 15 08D8 2010 ;
0000045D'EF 00000465'EF 03 15 08D8 2011 MOVZBL ROWCON[R5],R7 ; Get the rows for 1024kb arrays
0000045D'EF 00000465'EF 03 15 08E0 2012 SUBL2 #3,R7 ; Go back one array to allow for 0
0000045D'EF 00000465'EF 03 15 08E3 2013 SUBL3 R7,LOWROW,R9 ; Extract the 1024kb rows
0000045D'EF 00000465'EF 03 15 08EB 2014 MULL3 R7,#^X40000,TEMP ; Calculate max address of 1024kb array
0000045D'EF 00000465'EF 03 15 08F7 2015 MULL3 R9,#^X10000,LOWADDR ; Calculate lowest address 256kb arrays
0000045D'EF 00000465'EF 03 15 0903 2016 BRW 47$ ; Continue.....
0000045D'EF 00000465'EF 03 15 0906 2017 45$: MULL3 LOWROW,#^X10000,LOWADDR ; Calculate lowest address 256kb arrays
0000045D'EF 00000465'EF 03 15 0916 2018 47$: TSTL TEMP ; Is this the first time here ?
0000045D'EF 00000465'EF 03 15 091C 2019 BEQL 48$ ; Branch if so
0000045D'EF 00000465'EF 03 15 091E 2020 ADDL2 TEMP,LOWADDR ; Calculate the address
0000045D'EF 00000465'EF 03 15 0929 2021 48$: CMPL FREE_PAGE_P,LOWADDR ; Find out where to start testing
0000045D'EF 00000465'EF 03 15 0934 2022 BLEQ 55$ ; Branch if free page is < or =
0000045D'EF 00000465'EF 03 15 0936 2023 ADDL2 R2,LOWROW ; Add the # of rows used by free page
0000045D'EF 00000465'EF 03 15 093D 2024 BRW 45$ ; Branch to calculate a new address
0000045D'EF 00000465'EF 03 15 0940 2025 50$: INCL R5 ; Bump the 1024kb array counter
0000045D'EF 00000465'EF 03 15 0942 2026 BRW 42$ ; Go see if there is any more
0000045D'EF 00000465'EF 03 15 0945 2027 55$: CMPL R4,R3 ; See if he has selected highest one
0000045D'EF 00000465'EF 03 15 0948 2028 BEQL 60$ ; If so, don't ask any more questions
0000045D'EF 00000465'EF 03 15 094A 2029 $DS_PRINTF_S FMT_ASK_HIGH ; This is the high question message
0000045D'EF 00000465'EF 03 15 094A 2029 PUSHAB FMT_ASK_HIGH
0000045D'EF 00000465'EF 03 15 0950 CALLS $$$N, a#DS$PRINTF
0000045D'EF 00000465'EF 03 15 0957 2030 $DS_ASKDATA_S ASK,TEMP1,#10,,R4,R3,R3 ; Ask him for the high one
0000045D'EF 00000465'EF 03 15 0957 2030 PUSHL #0
0000045D'EF 00000465'EF 03 15 0959 2030 PUSHL #0
0000045D'EF 00000465'EF 03 15 095B 2030 PUSHL R3
0000045D'EF 00000465'EF 03 15 095D 2030 PUSHL R3

```

```

          54 DD 095F PUSHL R4
          FFFFFFFF 8F DD 0961 PUSHL #-1
          0A DD 0967 PUSHL #10
          00000AA4'EF DF 0969 PUSHAL TEMP1
          000013B1'EF DF 096F PUSHAL ASK
          00010080 9F 09 FB 0975 CALLS #9, a#DS$ASKDATA
          54 00000AA4'EF D0 097C 2031 MOVL TEMP1,R4 ; Put the high array number in R4
00000459'EF 00001048'EF44 9A 0983 2032 60$: MOVZBL ROWCON[R4],HIGHROW ; Get the high row number
          098F 2033 ;
          56 D4 098F 2034 CLRL R6 ;[3.0] If first array is not 4MB, then
          0000103C'EF46 01 91 0991 2035 CMPB #1,SLOT[R6] ; branch to 1024kb/256kb loop
          30 12 0999 2036 BNEQ 64$
          099B 2037 ;
          099B 2038 ; Case of 4Mb or 4Mb/1024kb combination
          099B 2039 ;
          0000103C'EF46 01 91 099B 2040 61$: CMPB #1,SLOT[R6] ;[3.0] If 4Mb present, then there are
          0D 12 09A3 2041 BNEQ 62$ ; no 256kb arrays
          00000461'EF 00400000 8F C0 09A5 2042 ADDL2 #^X400000,HIGHADDR ; Calculate high address
          0B 11 09B0 2043 BRB 63$
          00000461'EF 00100000 8F C0 09B2 2044 62$: ADDL2 #^X100000,HIGHADDR ; Calculate high address
          DA 56 54 F3 09BD 2045 63$: AOBLEQ R4,R6,61$
          00000461'EF 04 C2 09C1 2046 SUBL2 #4,HIGHADDR ; Backup to last longword address
          00EF 31 09C8 2047 BRW 85$
          09CB 2048 ;
          55 95 09CB 2049 64$: TSTB R5 ; If 0 says no mix arrays
          22 13 09CD 2050 BEQL 65$ ; Branch if so
          51 00000459'EF 57 C3 09CF 2051 SUBL3 R7,HIGHROW,R1 ; Subtract # of rows for 1024kb arrays
00000461'EF 00010000 8F 51 C5 09D7 2052 MULL3 R1,#^X10000,HIGHADDR ; Calculate address
          00000461'EF 00000824'EF C0 09E3 2053 ADDL2 TEMP,HIGHADDR ; Calculate max address
          00BE 31 09EE 2054 BRW 83$ ; Continue...
          0000103C'EF45 FF 8F 91 09F1 2055 65$: CMPB #-1,SLOT[R5] ; Is there only 256kb arrays ?
          03 13 09FA 2056 BEQL 67$ ; Branch if so
          0003 31 09FC 2057 BRW 70$ ; . . .
          008D 31 09FF 2058 67$: BRW 80$ ; . . .
          0000103C'EF44 FF 8F 91 0A02 2059 70$: CMPB #-1,SLOT[R4] ; Is there a mixtute ?
          2E 13 0A0B 2060 BEQL 75$ ; Branch if so
          00000455'EF 00040000 8F C5 0A0D 2061 MULL3 #^X40000,LOWROW,LOWADDR ; Calculate 1024kb array address (low)
          0000045D'EF 0A18
          00000459'EF 00040000 8F C5 0A1D 2062 MULL3 #^X40000,HIGHROW,HIGHADDR ; Calculate the high address
          00000461'EF 0A28
          00000461'EF 003FFFFC 8F C0 0A2D 2063 ADDL2 #^X3FFFFC,HIGHADDR ; . . .
          007F 31 0A38 2064 BRW 85$ ; End . . .
          00001064'EF D6 0A3B 2065 75$: INCL COUNTER ; Increment the 256kb array counter
          54 D7 0A41 2066 DECL R4 ; Decrement the slot counter
          54 58 D1 0A43 2067 CMPL R8,R4 ; Is R4 = to low array ?
          BA 19 0A46 2068 BLSS 70$ ; Branch if less
          00001064'EF 04 C4 0A48 2069 MULL2 #4,COUNTER ; Calculate the # of rows for 256kb arrays
          00000459'EF 00001064'EF C3 0A4F 2070 SUBL3 COUNTER,HIGHROW,R11 ; Subtract from # of rows of 1024kb arrays
          5B 01 C0 0A5B 2071 ADDL2 #1,R11 ; Add 1 to allow for row 0
          00010000 8F 00001064'EF C5 0A5E 2072 MULL3 COUNTER,#^X10000,TEMP ; Calculate 256kb array address
          00000824'EF 0A69
          00000461'EF 00040000 8F 5B C5 0A6E 2073 MULL3 R11,#^X40000,HIGHADDR ; Calculate 1024kb array address
          00000461'EF 00000824'EF C0 0A7A 2074 ADDL2 TEMP,HIGHADDR ; Get max address
          00000461'EF 04 C2 0A85 2075 SUBL2 #4,HIGHADDR ; Adjust to a longword address
          002B 31 0A8C 2076 BRW 85$ ; Branch . . .
          00000455'EF 00010000 8F C5 0A8F 2077 80$: MULL3 #^X10000,LOWROW,LOWADDR ; Calculate the low address
          0000045D'EF 0A9A

```

ZZ-ECKAM-3.2
ECKAM
03-02

GET ARRAY CARDS TO TEST ROUTINE
MEMORY DIAGNOSTIC HEADER
GET ARRAY CARDS TO TEST ROUTINE

C 7

13-MAY-1987

Fiche 1 Frame C7

Sequence 80

23-JAN-1987 14:30:12

VAX/VMS Macro V04-00

Page 57

23-JAN-1987 14:26:49

[ZECCHINO.ECKAM]ECKAM0.MAR;4

(18)

```
00000459'EF 00010000 8F C5 0A9F 2078 MULL3 #^X10000,HIGHROW,HIGHADDR ; Calculate the high address
00000461'EF 00000461'EF 0AAA
00000461'EF 0000FFFC 8F C0 0AAF 2079 83$: ADDL2 #^XFFFC,HIGHADDR ;
52 01 D0 0ABA 2080 85$: MOVL #1,R2 ; Indicate success
0ABD 2081 1000$:
0ABD 2082 GET_ARRAYSX:
00F8 8F BA 0ABD 2083 POPR #^M<R3,R4,R5,R6,R7> ; Unsave the registers
05 0AC1 2084 RSB ; Return
0AC2 2085 $DS_ENDMOD
0AC2 2086 .END
```

MEMORY DIAGNOSTIC HEADER

\$\$ARGS	=	0000000A			CK	000005D1	RG	08
\$\$E	=	00000001			CK0	000005D1	RG	08
\$\$N	=	00000001			CK1	000005D7	RG	08
\$\$S	=	FFFFFFFF			CK2	000005DE	RG	08
\$\$T1	=	0000002C			CK3	000005E6	RG	08
\$ENV	=	00000001	G		CK4	000005ED	RG	08
\$ER	=	00000001			CK5	000005F5	RG	08
\$MO	=	00000001	G		CK6	000005FC	RG	08
\$ST	=	00000000			CKALL	00000603	RG	08
\$TN	=	00000000			CKBIT	00000509	RG	08
ALL	=	00000001	G		CKBIT0	00000509	RG	08
ALLMOD	=	000011A3	RG	08	CKBIT1	00000521	RG	08
AL_DEVTYP	=	00000039	R	01	CKBIT2	0000053D	RG	08
ARRAY	=	0000119A	RG	08	CKBIT3	0000055D	RG	08
ARRAY_LEN	=	00000008			CKBIT4	00000579	RG	08
ARRAY_LEN1	=	0000001D			CKBIT5	00000599	RG	08
ARRAY_SGL_BIT	=	000011D4	RG	08	CKBIT6	000005B5	RG	08
ASK	=	000013B1	RG	08	CLEAN_UP	00000000	R	0C
A_HEADEND	=	00000058	R	02	CLEAN_UP_X	0000004A	R	0C
BIT...	=	00000028			CODEVECTORS	00000475	RG	08
BIT0	=	00000001			CODEWORDS	00000489	RG	08
BIT1	=	00000002			CONTROL	000011B5	RG	08
BIT10	=	00000400			CONTROLLER	=	00F20000	
BIT11	=	00000800			CORRUPT_TEST_LOC	00000213	RG	09
BIT12	=	00001000			CORR_ERR_ISR	00000370	RG	09
BIT13	=	00002000			CORR_ERR_ISRX	0000041C	R	09
BIT14	=	00004000			COR_ERRORS	00000C3C	RG	08
BIT15	=	00008000			COUNTER	00001064	RG	08
BIT16	=	00010000			CPU	000011C8	RG	08
BIT17	=	00020000			CS0	=	00000000	
BIT18	=	00040000			CS0_MNEM	000010FC	R	08
BIT19	=	00080000			CS1	=	00000004	
BIT2	=	00000004			CS1_MNEM	0000113A	R	08
BIT20	=	00100000			CS2	=	00000008	
BIT21	=	00200000			CS2_MNEM	00001171	R	08
BIT22	=	00400000			CVTRÉG_BUFF	00001078	R	08
BIT23	=	00800000			DATA	00000000	RG	08
BIT24	=	01000000			DCM	=	04000000	
BIT25	=	02000000			DECODED_WORDS	00000B3C	RG	08
BIT26	=	04000000			DECODER	000002A5	RG	09
BIT27	=	08000000			DECODERX	00000369	R	09
BIT28	=	10000000			DEFAULT	=	00000000	G
BIT29	=	20000000			DEVICE_TYPE	00000018	RG	08
BIT3	=	00000008			DEV_RÉG	00000000	R	01
BIT30	=	40000000			DIGIT	000011A1	RG	08
BIT31	=	80000000			DIGIT1	00001422	RG	08
BIT4	=	00000010			DISABLE_CACHE	=	00000001	
BIT5	=	00000020			DISPATCH	00000000	R	06
BIT6	=	00000040			DS\$ABORT	00010020	G	
BIT7	=	00000080			DS\$ASKADR	00010090	G	
BIT8	=	00000100			DS\$ASKDATA	00010080	G	
BIT9	=	00000200			DS\$ASKLGCL	00010098	G	
BUS_ERR_REG	=	0000002D	RG	08	DS\$ASKSTR	000100A0	G	
CADR	=	00000025			DS\$ASKVLD	00010088	G	
CE	=	20000000			DS\$ATTACH	000101A8	G	
CEP_FUNCTIONAL	=	00000000			DS\$BGNSUB	00010030	G	
CEP_REPAIR	=	00000001			DS\$BRANCH	000100A8	G	

DS\$BREAK 00010058 G
DS\$CANWAIT 00010070 G
DS\$CHANNEL 00010180 G
DS\$CKLOOP 00010040 G
DS\$CLRVEC 00010168 G
DS\$CNTRLC 00010078 G
DS\$CVTREG 000100B0 G
DS\$ENDPASS 00010010 G
DS\$ENDSUB 00010038 G
DS\$ERRDEV 000100C8 G
DS\$ERRHARD 000100D0 G
DS\$ERRPREP 00010108 G
DS\$ERRSOFT 000100D8 G
DS\$ERRSYS 000100C0 G
DS\$ESCAPE 00010050 G
DS\$FREEDBGSYM 000101B8 G
DS\$GETBUF 00010120 G
DS\$GETMEM 00010130 G
DS\$GETTERM 000101C8 G
DS\$GPHARD 00010018 G
DS\$HELP 000101B0 G
DS\$INITSCB 00010170 G
DS\$INLOOP 00010048 G
DS\$K_BBC = 0000001D G
DS\$K_BBCC = 00000021 G
DS\$K_BBCCI = 00000023 G
DS\$K_BBCS = 0000001F G
DS\$K_BBS = 0000001C G
DS\$K_BBSC = 00000020 G
DS\$K_BBSS = 0000001E G
DS\$K_BBSSI = 00000022 G
DS\$K_BCC_M = 0000001B G
DS\$K_BCS_M = 0000001A G
DS\$K_BEQLU_B = 00000005 G
DS\$K_BEQLU_M = 00000011 G
DS\$K_BEQL_B = 00000004 G
DS\$K_BEQL_M = 00000010 G
DS\$K_BERROR = 00000026 G
DS\$K_BGEQU_B = 00000007 G
DS\$K_BGEQU_M = 00000013 G
DS\$K_BGEQ_B = 00000006 G
DS\$K_BGEQ_M = 00000012 G
DS\$K_BGTRU_B = 00000009 G
DS\$K_BGTRU_M = 00000015 G
DS\$K_BGTR_B = 00000008 G
DS\$K_BGTR_M = 00000014 G
DS\$K_BLBC = 00000025 G
DS\$K_BLBS = 00000024 G
DS\$K_BLEQU_B = 00000003 G
DS\$K_BLEQU_M = 0000000F G
DS\$K_BLEQ_B = 00000002 G
DS\$K_BLEQ_M = 0000000E G
DS\$K_BLSSU_B = 00000001 G
DS\$K_BLSSU_M = 0000000D G
DS\$K_BLSS_B = 00000000 G
DS\$K_BLSS_M = 0000000C G
DS\$K_BNEQU_B = 0000000B G

DS\$K_BNEQU_M = 00000017 G
DS\$K_BNEQ_B = 0000000A G
DS\$K_BNEQ_M = 00000016 G
DS\$K_BNERROR = 00000027 G
DS\$K_BVC_M = 00000019 G
DS\$K_BVS_M = 00000018 G
DS\$LOAD 00010198 G
DS\$LOADPCS 000101C0 G
DS\$MAPDBGBLOCK 00010118 G
DS\$MEMSIZE 000101D8 G
DS\$MMOFF 00010158 G
DS\$MMON 00010150 G
DS\$MOVPHY 00010148 G
DS\$MOVVRT 00010140 G
DS\$PARSE 000100B8 G
DS\$PRINTB 000100E0 G
DS\$PRINTF 000100F0 G
DS\$PRINTS 000100F8 G
DS\$PRINTSIG 00010100 G
DS\$PRINTX 000100E8 G
DS\$PROBE 000101A0 G
DS\$RELBUF 00010128 G
DS\$RELMEM 00010138 G
DS\$SERVICE_DISPATCHER 000101D0 G
DS\$SETIPL 00010178 G
DS\$SETMAP 00010188 G
DS\$SETPRIEXV 00010110 G
DS\$SETVEC 00010160 G
DS\$SHOCHAN 00010190 G
DS\$SUMMARY 00010028 G
DS\$WAITMS 00010060 G
DS\$WAITUS 00010068 G
DSA\$AL_APTMAIL 0000FE00
DSA\$AT_APTTXT 0000FA00
DSA\$GL_APTCOM 0000FE04
DSA\$GL_DEVLEN 0000FE58
DSA\$GL_ERRNO 0000FE44
DSA\$GL_EVENT 0000FE48
DSA\$GL_FLAGS 0000FE00
DSA\$GL_MSGTYP 0000FE40
DSA\$GL_PASSES 0000FE08
DSA\$GL_PASSNO 0000FE54
DSA\$GL_SECTNO 0000FE10
DSA\$GL_SID 0000FE14
DSA\$GL_SUBTNO 0000FE4C
DSA\$GL_TESTNO 0000FE50
DSA\$GL_UNITS 0000FE0C
DSA\$GQ_MSGPTR 0000FE68
DSA\$GT_DEVNAM 0000FE5C
EDM = 02000000
ENV\$M_DOMAIN = 00000002
ENV\$M_LEVEL = 00000001
ENV\$M_SUPER = 000003FC
ENV\$S_DOMAIN = 00000001
ENV\$S_LEVEL = 00000001
ENV\$S_SUPER = 00000008
ENV\$V_DOMAIN = 00000001

ENV\$V_LEVEL	= 00000000			FMT_ECC_15	00001BB0	RG	08
ENV\$V_SUPER	= 00000002			FMT_ECC_16	00001BE4	RG	08
ENV\$CPU	= 00000000			FMT_ECC_2	00001730	RG	08
ENV\$FUNCTIONAL	= 00000000			FMT_ECC_3	0000178F	RG	08
ENV\$REPAIR	= 00000001			FMT_ECC_4	00001802	RG	08
ENV\$SUPER	= 00000001			FMT_ECC_5	00001864	RG	08
ENV\$SYSTEM	= 00000001			FMT_ECC_6	000018C5	RG	08
ERCE	= 10000000			FMT_ECC_7	00001921	RG	08
ERR\$MSGADR	= 0000000C			FMT_ECC_8	00001980	RG	08
ERR\$NARGS	= 0000000A			FMT_ECC_9	000019DD	RG	08
ERR\$NUM	= 00000004			FMT_EMPTY	00001324	RG	08
ERR\$P1	= 00000014			FMT_ILL_CFG	0000213F	RG	08
ERR\$P2	= 00000018			FMT_INF_LST	00001D4B	RG	08
ERR\$P3	= 0000001C			FMT_LOG_FULL	000023B5	RG	08
ERR\$P4	= 00000020			FMT_MCH_STACK1	00002434	RG	08
ERR\$P5	= 00000024			FMT_MCH_STACK2	000024F1	RG	08
ERR\$P6	= 00000028			FMT_MEM_MAP	00001209	RG	08
ERR\$POINTER	= 00000010			FMT_MEM_MAP2	0000123A	RG	08
ERR\$UNIT	= 00000008			FMT_MEM_MAP3	0000126B	RG	08
ERROR_TYPE	00000014	RG	08	FMT_MEM_REG	000023CB	RG	08
ERR_LOG_FULL	= 00000441	RG	08	FMT_MEM_SIZ	000011E2	RG	08
EXHAUSTIVE	= 00000002	G		FMT_NO_ARRAYS	000013B3	RG	08
EXP_COR_ERR	0000001D	RG	08	FMT_NO_SUPP	000012F2	RG	08
EXP_INF_LST	00000025	RG	08	FMT_NSBH	00001DA7	RG	08
EXP_MSG	000021DC	RG	08	FMT_NX_MEM	00001DC1	RG	08
EXP_NX_MEM	00000029	RG	08	FMT_RAM	00001DDD	RG	08
EXP_UNC_ERR	00000021	RG	08	FMT_REG_ERR	00002185	RG	08
FLAG1	00001054	RG	08	FMT_ROM_1	000013E1	RG	08
FLAG2	00001058	RG	08	FMT_ROM_2	00001630	RG	08
FLAG3	0000105C	RG	08	FMT_ROM_3	00001405	RG	08
FLAG4	00001060	RG	08	FMT_ROM_4	000016A3	RG	08
FMT_1024_KB	000012B9	RG	08	FMT_ROW	00001E03	RG	08
FMT_256_KB	0000129C	RG	08	FMT_ROW_MSG	00001437	RG	08
FMT_4_MB	000012D7	RG	08	FMT_SA0	00001E24	RG	08
FMT_ADR_BUS	00001D1B	RG	08	FMT_SA1	00001E32	RG	08
FMT_ARRAY	00001419	RG	08	FMT_SGL_BIT_ERR	00002581	RG	08
FMT_ASK_HIGH	000013A6	RG	08	FMT_SHORT	00001E3F	RG	08
FMT_ASK_LOW	0000139B	RG	08	FMT_SIZ_ERR	00001E52	RG	08
FMT_AVAILABLE	00001339	RG	08	FMT_SUMMARY1	000021F4	R	08
FMT_BAD_LOC	00001EAD	RG	08	FMT_SUMMARY2	00002275	R	08
FMT_BODY	00001CC3	RG	08	FMT_SUMMARY3	0000229E	R	08
FMT_COMP_ERR	00001CDC	RG	08	FMT_SUMMARY4	000022C3	R	08
FMT_CPE_MC	00001D73	RG	08	FMT_SUMMARY5	000022EC	R	08
FMT_CPU_0	0000144D	RG	08	FMT_SUMMARY6	00002316	R	08
FMT_CPU_1	000014AD	RG	08	FMT_SUMMARY7	000023AA	R	08
FMT_CPU_2	00001511	RG	08	FMT_UNK_ET	00001C1B	RG	08
FMT_CPU_3	00001579	RG	08	FMT_UNK_MC	00001C30	RG	08
FMT_CPU_4	000015CE	RG	08	FMT_UNX_TO	00001C59	RG	08
FMT_CS_BUS	00001D88	RG	08	FMT_UNX_UNC	00001C7D	RG	08
FMT_CS_MC	00001D01	RG	08	FMT_WARN_MES2	00002057	RG	08
FMT_DATA_BUS	00001D37	RG	08	FMT_WARN_MESS	00001F70	RG	08
FMT_ECC_1	000016E0	RG	08	FMT_WBE_MC	0000212D	RG	08
FMT_ECC_10	00001A4E	RG	08	FREE_PAGE_P	00000465	RG	08
FMT_ECC_11	00001ABA	RG	08	FREE_PAGE_V	0000046D	RG	08
FMT_ECC_12	00001ADA	RG	08	FROW	00001050	RG	08
FMT_ECC_13	00001B19	RG	08	GET_ARRAYS	00000758	RG	09
FMT_ECC_14	00001B6D	RG	08	GET_ARRAYSX	00000ABD	R	09

MEMORY DIAGNOSTIC HEADER

HIGHADDR	00000461	RG	08	SEQUENCE	00000004	RG	08
HIGHROW	00000459	RG	08	SIZ...	= 00000001		
INITIALIZE	00000000	R	0B	SLOT	0000103C	RG	08
INITIALIZE_X	00000227	R	0B	SUMMARY	00000000	R	0A
L\$A_CCP	00000040	R	02	SUMMARY_FLAG	0000043D	RG	08
L\$A_DEVP	0000001C	R	02	SUMMARY_X	000000EC	R	0A
L\$A_DREG	00000024	R	02	SYNDROME	00000604	RG	08
L\$A_DTP	00000018	R	02	SYS\$ALLOC	00010238	G	
L\$A_ICP	0000003C	R	02	SYS\$ASCTIM	00010248	G	
L\$A_LASTAD	00000014	R	02	SYS\$ASSIGN	00010250	G	
L\$A_NAME	00000008	R	02	SYS\$BINTIM	00010258	G	
L\$A_REPP	00000044	R	02	SYS\$CANCEL	00010260	G	
L\$A_SECNAM	00000050	R	02	SYS\$CANTIM	00010268	G	
L\$A_STATAB	00000048	R	02	SYS\$CLOSE	000105B8	G	
L\$A_TSTCNT	00000054	R	02	SYS\$CLREF	00010298	G	
L\$L_ENVIRON	00000004	R	02	SYS\$CONNECT	000105C0	G	
L\$L_ERRTYP	0000004C	R	02	SYS\$DALLOC	000102D8	G	
L\$L_HEADLENGTH	00000000	R	02	SYS\$DASSGN	000102E0	G	
L\$L_REV	0000000C	R	02	SYS\$DISCONNECT	000105D0	G	
L\$L_UNIT	00000020	R	02	SYS\$FAO	00010350	G	
L\$L_UPDATE	00000010	R	02	SYS\$FAOL	00010358	G	
LASTAD	00000000	R	03	SYS\$GET	00010580	G	
LOWADDR	0000045D	RG	08	SYS\$GETCHN	000104C8	G	
LOWROW	00000455	RG	08	SYS\$GETTIM	00010378	G	
L_CUR_ARRAY	00001070	RG	08	SYS\$LKWSET	000103A0	G	
L_TSTCNT	00000000	R	04	SYS\$NUMTIM	000103B8	G	
MACHINECK	00000430	RG	09	SYS\$OPEN	00010608	G	
MACHINECKX	000005B1	R	09	SYS\$QIO	000103C8	G	
MANUAL	= 00000003	G		SYS\$QIOW	00010200	G	
MAXLWA	00000449	RG	08	SYS\$READ	00010590	G	
MAXPFN	00000445	RG	08	SYS\$READEF	000103D0	G	
MAXROW	0000044D	RG	08	SYS\$SETAST	000103F8	G	
MOVI_PAT_0	00000B24	RG	08	SYS\$SETEF	00010400	G	
MOVI_PAT_1	00000B2C	RG	08	SYS\$SETIMR	00010420	G	
MS750	= 00000001			SYS\$SETPRI	00010428	G	
NOISY_WORDS	00000BBC	RG	08	SYS\$SETPRT	00010430	G	
NUM_COR_ERR	00000435	RG	08	SYS\$SETRWM	00010438	G	
PATCNT	00000008	RG	08	SYS\$SETSWM	00010448	G	
PATTERN	00000B34	RG	08	SYS\$ULKPAG	00010460	G	
PGE_COR_ERR	00000031	RG	08	SYS\$ULWSET	00010468	G	
PM	= 08000000			SYS\$UNWIND	00010470	G	
PRINT	00000000	RG	09	SYS\$WAITFR	00010478	G	
PRINTX	000000C7	R	09	SYS\$WFLAND	00010488	G	
PRINT_BAD_CONFIG	000001F7	RG	09	SYS\$WFLOR	00010490	G	
PRINT_GENERAL	000000C8	RG	09	S_ALL	00000008	R	01
PRINT_GEN_X	00000135	R	09	S_DEFAULT	00000000	R	01
PRINT_UNX_TO	00000226	RG	09	S_EXHAUSTIVE	0000000C	R	01
PRINT_UNX_UNC	00000136	RG	09	S_MANUAL	00000017	R	01
READ_MAP	000005B9	RG	09	TEMP	00000824	RG	08
READ_MAPX	00000753	R	09	TEMP1	00000AA4	RG	08
REC_MSG	000021D0	RG	08	TEST_ADDRESS	00000451	RG	08
RETURN_PC	0000000C	RG	08	TIME_LEFT	00001074	RG	08
ROM_TYPE	0000001B	RG	08	TOTAL_COR_ERR	00000439	RG	08
ROWCON	00001048	RG	08	TRANSITION	00001068	RG	08
SECTION	0000001E	R	01	TRANSITION4	0000106C	RG	08
SEP_FUNCTIONAL	= 00000002			T_MS750	00000032	R	01
SEP_REPAIR	= 00000003			T_NAME	00000058	R	02

UCE = 80000000
 UCEIL = 40000000
 UNX_UNC_PC 00000010 RG 08
 VA_COR_ADDRESS 00000035 RG 08
 WHICH_ARRAY 0000027A RG 09
 WHICH_ARRAYX 000002A4 R 09
 XOR_MSG 000021E8 RG 08

 ! Psect synopsis !

PSECT name	Allocation	PSECT No.	Attributes
. ABS	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
HEADER	00000041 (65.)	01 (1.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC LONG
\$HEADER	00000080 (128.)	02 (2.)	NOPIC USR CON REL LCL NOSHR NOEXE RD NOWRT NOVEC PAGE
_LAST	00000000 (0.)	03 (3.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC PAGE
\$TSTCNT	00000000 (0.)	04 (4.)	NOPIC USR OVR REL LCL NOSHR NOEXE RD NOWRT NOVEC LONG
\$ABS\$	00010610 (67088.)	05 (5.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
DISPATCH	00000000 (0.)	06 (6.)	NOPIC USR CON REL LCL NOSHR NOEXE RD NOWRT NOVEC LONG
DISPATCH_X	00000018 (24.)	07 (7.)	NOPIC USR CON REL LCL NOSHR NOEXE RD NOWRT NOVEC LONG
DATA	000025B3 (9651.)	08 (8.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC LONG
ROUTINES	00000AC2 (2754.)	09 (9.)	NOPIC USR CON REL LCL NOSHR EXE RD NOWRT NOVEC PAGE
SUMMARY	000000F4 (244.)	0A (10.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC LONG
INITIALIZE	0000022F (559.)	0B (11.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC LONG
CLEANUP	00000052 (82.)	0C (12.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC LONG

 ! Performance indicators !

Phase	Page faults	CPU Time	Elapsed Time
Initialization	40	00:00:00.17	00:00:00.67
Command processing	910	00:00:00.60	00:00:02.07
Pass 1	744	00:00:12.01	00:00:25.10
Symbol table sort	0	00:00:00.64	00:00:00.65
Pass 2	134	00:00:04.85	00:00:07.20
Symbol table output	2	00:00:00.24	00:00:00.59
Psect synopsis output	2	00:00:00.04	00:00:00.04
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	1833	00:00:18.55	00:00:36.33

The working set limit was 2048 pages.
 244183 bytes (477 pages) of virtual memory were used to buffer the intermediate code.
 There were 30 pages of symbol table space allocated to hold 520 non-local and 144 local symbols.
 2086 source lines were read in Pass 1, producing 69 object records in Pass 2.
 35 pages of virtual memory were used to define 33 macros.

! Macro library statistics !

Macro library name	Macros defined
-----	-----
SYS\$COMMON:[SYSLIB]DIAG.MLB;1	38
SYS\$COMMON:[SYSLIB]STARLET.MLB;2	7
TOTALS (all libraries)	45

701 GETS were required to define 45 macros.

There were no errors, warnings or information messages.

MAC/LIS ECKAM0

(2)	142	DECLARATIONS
(3)	201	TEST 1: MEMORY MAP VERIFICATION TEST
(4)	304	TEST 2: DATA BUS TEST
(5)	398	TEST 3: ROW SELECT BUS TEST
(7)	587	TEST 4: ADDRESS BUS TEST
(8)	755	TEST 5: ECC LOGIC TEST
(9)	1418	TEST 6: CSRO TEST
(10)	1730	TEST 7: BOOTSTRAP ROM TEST
(11)	1807	TEST 8: CPU LOST ERROR TEST
(12)	1944	TEST 9: CPU XB ERROR BIT TEST
(13)	2066	TEST 10: MOVING INVERSIONS TEST
(14)	2376	TEST 11: MOVI WITH MANUAL ARRAY SELECT
(15)	2641	TEST 12: MEMORY QUICK VERIFY TEST

```
0000 1 .TITLE ECKAM MEMORY SUBSYSTEM TESTS
0000 2 .IDENT /03-02/
0000 3
0000 4 :
0000 5 : COPYRIGHT (C) 1980, 1981, 1982, 1986
0000 6 : DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS 01754
0000 7 :
0000 8 : THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY ON A SINGLE
0000 9 : COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH THE INCLUSION OF THE
0000 10 : ABOVE COPYRIGHT NOTICE. THIS SOFTWARE, OR ANY OTHER COPIES THEREOF,
0000 11 : MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY OTHER PERSON
0000 12 : EXCEPT FOR USE ON SUCH SYSTEM AND TO ONE WHO AGREES TO THESE LICENSE
0000 13 : TERMS. TITLE TO AND OWNERSHIP OF THE SOFTWARE SHALL AT ALL TIMES
0000 14 : REMAIN IN DEC.
0000 15 :
0000 16 : THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 17 : AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 18 : CORPORATION.
0000 19 :
0000 20 : DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 21 : SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.
0000 22 :
0000 23 :
0000 24 :++
0000 25 : FACILITY: COMET Diagnostic Library
0000 26 :
0000 27 : ABSTRACT: This program detects and isolates memory subsystem failures.
0000 28 :
0000 29 : ENVIRONMENT: Diagnostic Supervisor
0000 30 :
0000 31 : AUTHOR: Rand Gray, CREATION DATE: 23-JUN-78
0000 32 :
0000 33 : MODIFIED BY: Don Monroe
0000 34 :
0000 35 : 00-02 May 1979
0000 36 : Added a subtest to the ECC Logic Test to check that the
0000 37 : Controller Interrupts correctly, and a subtest to check
0000 38 : that inhibiting the interrupt also inhibited the setting of
0000 39 : bit 29 and changing of bits <23:9> in CSR0.
0000 40 :
0000 41 : Added a subtest to ECC Logic Test to ensure that a BYTE write
0000 42 : while forcing a single bit error does not report the error
0000 43 : and does correct the error.
0000 44 :
0000 45 : Fixed a bug in ECC Logic Test Subtest 5. Single bit error
0000 46 : correcting of a floating zero did not float the zero.
0000 47 :
0000 48 : Added a test for the error address bits in CSR0 and other
0000 49 : miscellaneous functions of CSR0.
0000 50 :
0000 51 : Added a test for the bootstrap ROMS.
0000 52 :
0000 53 : Added a test for the LOST ERROR bit in the CPU BUS ERROR
0000 54 : register.
0000 55 :
0000 56 : Added a test for the XB bit in the MACHINE CHECK ERROR
0000 57 : SUMMARY REGISTER.
```

0000	58 ;		
0000	59 ;	00-03	June 1979
0000	60 ;		Renamed the IRCE bit to ERCE in CSR1. Changed tests to enable
0000	61 ;		the interrupt when the bit is set.
0000	62 ;		
0000	63 ;	00-04	July 1979
0000	64 ;		Modified to support the new stack image for single bit error
0000	65 ;		interrupts.
0000	66 ;		
0000	67 ;	00-05	July 1979 Don Monroe
0000	68 ;		Added a check for more than one array to tests that cannot
0000	69 ;		use array 0.
0000	70 ;		
0000	71 ;	00-06	August 1979 Don Monroe
0000	72 ;		Added test of Bus Error Register on Single Bit Interrupts.
0000	73 ;		
0000	74 ;	00-07	September 1979 Don Monroe
0000	75 ;		Test 5 Subtest 1 did not clear ERCE bit. Fixed it.
0000	76 ;		Added a forced loop back to the beginning of a test or subtest
0000	77 ;		if an unexpected uncorrectable error occurs.
0000	78 ;		Added a subtest to test 6 to check that CSR0 is not overwritten
0000	79 ;		by a single bit error if it contains a double bit error.
0000	80 ;	00-08	October 1, 1979 Don Monroe
0000	81 ;		Fixed bug in test 6 subtest 5.
0000	82 ;		
0000	83 ;	00-09	October 15, 1979 Don Monroe
0000	84 ;		Inhibited typeout of bootstrap ROM messages if not pass 1.
0000	85 ;		Moving Inversions tests now set the SUMMARY_FLAG to enable
0000	86 ;		the summary module typeout.
0000	87 ;	00-10	March 3, 1980 Don Monroe
0000	88 ;		Changed MOVING INVERSIONS to do a \$DS_BREAK more often.
0000	89 ;		Added PRINT LINKS to all ERRHARD calls. Added ERRHARDS for
0000	90 ;		unexpected double bit errors.
0000	91 ;	01-00	June 16, 1980 Don Monroe
0000	92 ;		Initial Release.
0000	93 ;	01-01	July 10, 1980 Don Monroe
0000	94 ;		Fixed bug in Test 5 Subtest 6. The mask on CSR0 did not
0000	95 ;		clear unused bits.
0000	96 ;		Fixed bugs in Test 6 Subtest 5. Was not setting up the
0000	97 ;		correct expected data. Stack would also overflow if looping
0000	98 ;		on error.
0000	99 ;	01-02	September 17, 1980 Don Monroe
0000	100 ;		Modified tests 6 and 10 so they don't clobber the Diagnostic
0000	101 ;		Supervisor.
0000	102 ;	01-03	May 27, 1981 Paul Hakala
0000	103 ;		Changed the INIT code to calculate the correct Max Address
0000	104 ;	02-01	Dec 31, 1981 Paul Hakala
0000	105 ;		Modified to accommodate both types of memory controllers
0000	106 ;		(L0011, L0016) and arrays (MS750AA-16kb RAMS, MS750CA-64k RAMS).
0000	107 ;	02-02	March 1, 1982 Paul Hakala
0000	108 ;		Modified tests 10 and 11 to provide better Ram coverage.
0000	109 ;	02-03	March 8, 1982 Paul Hakala
0000	110 ;		Added a quick verify for memory to detect any gross
0000	111 ;		memory errors with the RAMS. (Test 12)
0000	112 ;	02-04	May 1, 1982 Paul Hakala
0000	113 ;		1. Fixed manipulation of Reed-Muller scheme in tests 3 and 4
0000	114 ;		to function properly.

ZZ-ECKAM-3.2
ECKAM
03-02

ECKAM MEMORY SUBSYSTEM TESTS
MEMORY SUBSYSTEM TESTS

M 7

13-MAY-1987

Fiche 1 Frame M7

Sequence 90

23-JAN-1987 14:30:50 VAX/VMS Macro V04-00 Page 3
23-JAN-1987 14:29:55 [ZECCHINO.ECKAM]ECKAM1.MAR;6 (1)

0000 115 ;
0000 116 ;
0000 117 ;
0000 118 ;
0000 119 ;
0000 120 ;
0000 121 ;
0000 122 ;
0000 123 ;
0000 124 ;
0000 125 ;
0000 126 ;
0000 127 ;
0000 128 ;
0000 129 ;
0000 130 ;
0000 131 ;
0000 132 ;
0000 133 ;
0000 134 ;
0000 135 ;
0000 136 ;
0000 137 ;
0000 138 ;
0000 139 ;
0000 140 ;--

2. Corrected misleading tests results in test 5, 6, and 8 to reflect what actually happens in the error messages.
3. Added a check to test 1 to determine if there was a memory sizing error.
4. Added a memory warning message to inform users of the mode (exhaustive) the program defaults to. Also a warning message to inform that only memory above the Diagnostic Supervisor is tested.
5. Modified the single bit error routine to log the actual VA that caused the CRD interrupt.

03-00 March 31, 1986 Susan E. Hutcheson
Added for support for M199 4Mb array and L0022 controller.
Search on [3.0]

03-01 June 20, 1986 Susan E. Hutcheson Version 3.1
Removed test for full error log from CORR_ERR_ISR and put it in tests 10,11, and 12 so that ERRHARD for APT would not be skipped over. Search on [3.1]

03-02 January 23,1987 Richard Zecchino Version 3.2
Fixed Problem in Test 12 with Flag1 not being set to indicate 1024kb array with 256kb rows.

```
0000 142      .SBTTL  DECLARATIONS
0000 143
0000 144 ;
0000 145 ; INCLUDE FILES:
0000 146 ;
0000 147      .LIBRARY      /SYS$LIBRARY:DIAG/
0000 148
0000 149      $DS_SECDEF      <ALL,EXHAUSTIVE,MANUAL>
0000 150
0000 151 ;
0000 152 ; MACROS:
0000 153 ;
0000 154
0000 155 ;
0000 156 ; EQUATED SYMBOLS:
0000 157 ;
0000 158      $DS_QADEF
0000 159      $DS_BITDEF
0000 160 ;+
0000 161 ; This is the memory controller physical address:
0000 162 ;-
00F20000 0000 164 CONTROLLER      =      ^XF20000
0000 165
0000 166 ;+
0000 167 ; These are PC offsets for accessing memory controller registers:
0000 168 ;-
00000000 0000 170 CSR0      =      0
00000004 0000 171 CSR1      =      4
00000008 0000 172 CSR2      =      8
0000 173
0000 174 ;+
0000 175 ; These are bit definitions for single bit fields in the memory controller
0000 176 ; registers.
0000 177 ;-
0000 178
0000 179 ;
0000 180 ; For CSR0
0000 181 ;
0000 182
00000000 0000 183 UCE      =      BIT31
00000000 0000 184 UCEIL    =      BIT30
00000000 0000 185 CE      =      BIT29
0000 186
0000 187 ;
0000 188 ; For CSR1
0000 189 ;
0000 190
00000000 0000 191 ERCE      =      BIT28
00000000 0000 192 PM      =      BIT27
00000000 0000 193 DCM      =      BIT26
00000000 0000 194 EDM      =      BIT25
0000 195
0000 196      $DS_DSADEF
0000 197
0000 198      $DS_BGNMOD      CEP_REPAIR,1
```

ZZ-ECKAM-3.2 DECLARATIONS
ECKAM
03-02

MEMORY SUBSYSTEM TESTS
DECLARATIONS

B 8

13-MAY-1987 Fiche 1 Frame B8 Sequence 92
23-JAN-1987 14:30:50 VAX/VMS Macro V04-00 Page 5
23-JAN-1987 14:29:55 [ZECCHINO.ECKAM]ECKAM1.MAR;6 (2)

0000 199 ::: Macro-32 Diagnostic macro library Version 10.0 "DIAG.MLB (271)"
0000 \$DS_PAGE

```

0000          .SBTTL TEST 1: MEMORY MAP VERIFICATION TEST
00000000      .PSECT TEST_001, PAGE, NOWRT
0020          $DS_BGNTTEST <DEFAULT,ALL,EXHAUSTIVE>
0020 202      DATA_001:
00000000 0020          .LONG 0 ; TEST ARGUMENT TABLE TERMINATOR
0024          TEST_001::
0000 0024          .WORD ^M<> ; ENTRY MASK
0026 203 ;++
0026 204 ; Functional description:
0026 205 ;
0026 206 ; In this test the memory map (bits <15:00> of Control and Status
0026 207 ; Register 2) is verified. The memory map consists of 8 2-bit cells,
0026 208 ; each of which contains a code representing the contents of each of the
0026 209 ; eight array slots of the memory subsystem backplane. The map code
0026 210 ; differs for L0016 controllers, because half-populated (128kb) boards
0026 211 ; are illegal due to design differences of the M8750 array. This test
0026 212 ; will determine what type of controller is present and take the
0026 213 ; appropriate action. The test also checks the max longword address
0026 214 ; found by reading memory, to the max longword address found as
0026 215 ; indicated by the map in CSR2. If they disagree a hard error will
0026 216 ; occur. The code for the L0011 and the L0016 is as follows.
0026 217 ;
0026 218 ; L0011:
0026 219 ;
0026 220 ; 1 - half populated (128kb), is no longer supported
0026 221 ; 2 - register error
0026 222 ; 3 - 256kb array
0026 223 ; 0 - empty
0026 224 ; L0016:
0026 225 ;
0026 226 ; 1 - register error
0026 227 ; 2 - 1024kb array
0026 228 ; 3 - 256kb array
0026 229 ; 0 - empty
0026 230 ;
0026 231 ; For example, a L0016 with a 1024kb array in slot 0, and a 256kb array
0026 232 ; in slots 1 and 2, and the rest of the slots empty would be indicated
0026 233 ; by the following map:
0026 234 ;
0026 235 ; CSR 2
0026 236 ;
0026 237 ; |15 14|13 12|11 10|09 08|07 06|05 04|03 02|01 00|
0026 238 ; |-----|-----|-----|-----|-----|-----|-----|-----|
0026 239 ; | 0 0 | 0 0 | 0 0 | 0 0 | 0 0 | 1 1 | 1 1 | 1 0 |
0026 240 ; |-----|-----|-----|-----|-----|-----|-----|-----|
0026 241 ;
0026 242 ; Test algorithm:
0026 243 ;
0026 244 ; The register is scanned 2 bits at a time, starting at cell 0. An
0026 245 ; error is caused by any 3 following a 0 or 1, or any 2 following any 0,
0026 246 ; or 3.
0026 247 ;
0026 248 ; The contents of the map are always printed whether there were any
0026 249 ; errors or not.
0026 250 ;--
0026 251

```

0000FE54'EF	01	D1	0026	252		CMPL	#1,DSA\$GL_PASSNO	; Pass number 1?	A8
	03	1F	002D	253		BLSSU	5\$; Branch if no	A8
	0003	31	002F	254		BRW	8\$; Continue	
	00DE	31	0032	255	5\$:	BRW	60\$; Branch to end	
52	00000000'EF	16	0035	256	8\$:	JSB	READ_MAP	; Read the memory map	
	00F20000	8F	003B	257		MOVL	#CONTROLLER,R2	; Get address of controller	
	2F 08 A2	18	0042	258		BBC	#24,CSR2(R2),10\$; Check controller type	
			0047	259					
	15 08 A2	19	0047	260		BBC	#25,CSR2(R2),9\$;[3.0] Check controller type	
	53 08 A2	3C	004C	261		MOVZWL	CSR2(R2),R3	; Get the map contents	
			0050	262		\$DS_PRINTB_S	FMT_MEM_MAP3,R3	; Print the memory map (10022)	
		53	DD	0050		PUSHL	R3		
	00000000'EF	9F	0052			PUSHAB	FMT_MEM_MAP3		
	00000000'9F	02	FB	0058		CALLS	\$\$\$N, a#DS\$PRINTB		
		28	11	005F	263	BRB	12\$		
			0061	264					
	53 08 A2	3C	0061	265	9\$:	MOVZWL	CSR2(R2),R3	; Get the map contents	
			0065	266		\$DS_PRINTB_S	FMT_MEM_MAP,R3	; Print the memory map (10016)	
		53	DD	0065		PUSHL	R3		
	00000000'EF	9F	0067			PUSHAB	FMT_MEM_MAP		
	00000000'9F	02	FB	006D		CALLS	\$\$\$N, a#DS\$PRINTB		
			0074	267					
		13	11	0074	268	BRB	12\$; Return to normal flow	
	53 08 A2	3C	0076	269	10\$:	MOVZWL	CSR2(R2),R3	; Get the map contents	
			007A	270		\$DS_PRINTB_S	FMT_MEM_MAP2,R3	; Print the memory map (10011)	
		53	DD	007A		PUSHL	R3		
	00000000'EF	9F	007C			PUSHAB	FMT_MEM_MAP2		
	00000000'9F	02	FB	0082		CALLS	\$\$\$N, a#DS\$PRINTB		
		5B	D4	0089	271	CLRL	R11	; Clear slot counter	
	00000000'EF4B	95	008B	272	20\$:	TSTB	SLOT(R11)	; Check type of slot	
		2E	13	0092	273	BEQL	30\$; Branch if slot empty	
			0094	274					
		3D	19	0094	275	BLSS	40\$; Branch if slot is 256KB	
	00000000'EF4B	01	91	0096	276	CMPB	#1,SLOT(R11)	; if not 256K, slot must be	
		11	19	009E	277	BLSS	25\$; 1024KB or 4Mb	
			00A0	278		\$DS_PRINTB_S	FMT_4_MB,-	; Report this slot is 4MB	
			00A0	279			R11	; Slot number	
		5B	DD	00A0		PUSHL	R11		
	00000000'EF	9F	00A2			PUSHAB	FMT_4_MB		
	00000000'9F	02	FB	00A8		CALLS	\$\$\$N, a#DS\$PRINTB		
		31	11	00AF	280	BRB	50\$		
			00B1	281					
			00B1	282	25\$:	\$DS_PRINTB_S	FMT_1024_KB,-	; Report this slot is 1024KB	
			00B1	283			R11	; Slot number	
		5B	DD	00B1		PUSHL	R11		
	00000000'EF	9F	00B3			PUSHAB	FMT_1024_KB		
	00000000'9F	02	FB	00B9		CALLS	\$\$\$N, a#DS\$PRINTB		
		20	11	00C0	284	BRB	50\$; Branch to common flow	
			00C2	285	30\$:	\$DS_PRINTB_S	FMT_EMPTY,R11	; Report this slot is empty	
		5B	DD	00C2		PUSHL	R11		
	00000000'EF	9F	00C4			PUSHAB	FMT_EMPTY		
	00000000'9F	02	FB	00CA		CALLS	\$\$\$N, a#DS\$PRINTB		
		0F	11	00D1	286	BRB	50\$; And branch to common flow	
			00D3	287					
			00D3	288	40\$:	\$DS_PRINTB_S	FMT_256_KB,-	; Report this slot is 256kb	
			00D3	289			R11	; Slot number	
		5B	DD	00D3		PUSHL	R11		


```

00000000 011E .SBTTL TEST 2: DATA BUS TEST
00000000 0010 .PSECT TEST_002, PAGE, NOWRT
00000000 0010 305 $DS_BGNTST <DEFAULT,ALL,EXHAUSTIVE>
00000000 0010 DATA_002:
00000000 0010 .LONG 0 ; TEST ARGUMENT TABLE TERMINATOR
0000 0014 TEST_002::
0000 0014 .WORD ^M<> ; ENTRY MASK
0016 306 ;**
0016 307 ; Functional description:
0016 308 ;
0016 309 ; In this test the data bus is checked for bits stuck at one, stuck at
0016 310 ; zero, and shorted together. The test ignores individual RAM faults.
0016 311 ; The reason for this is that at this time we are only interested in the
0016 312 ; the data bus. Stuck or shorted RAMs will be detected in a later test.
0016 313 ; This partitioning of the tests is necessary to provide automatic fault
0016 314 ; isolation.
0016 315 ;
0016 316 ; Test algorithm:
0016 317 ;
0016 318 ; 1. Write a pattern of all zeros to the longword 0 of each row.
0016 319 ;
0016 320 ; 2. Read bit zero of each written longwords and store its value.
0016 321 ;
0016 322 ; 3. Write a pattern of all ones to longword 0 of each row.
0016 323 ;
0016 324 ; 4. Read bit zero of each written longword and store its value. Check
0016 325 ; that bit 0 of the written longwords is able to toggle (change state
0016 326 ; from 0 to 1). If bit 0 never toggles, then bit 0 of the data bus
0016 327 ; is stuck.
0016 328 ;
0016 329 ; 5. Repeat steps 1 through 4 for bits 1 through 31 of the data bus.
0016 330 ;
0016 331 ; Note: This test uses the smallest of the rows (64k boundarys) for
0016 332 ; test alogorithms.
0016 333 ;
0016 334 ; The algorithm for this test is based on an idea by Srini, from his
0016 335 ; paper, "Fault Location in a Semiconductor Random-Access Memory Unit,"
0016 336 ; IEEE TRANSACTIONS on COMPUTERS, Vol. C-27, No. 4, April, 1978.
0016 337 ;--
0016 338
00000000'EF 00000021'EF DE 0016 339 MOVAL 2$,UNX_UNC_PC ; Save PC for UNEX UNC ERROR A7
00000000'EF 00000000'EF D5 0021 340 2$: TSTL FREE_PAGE_P ; Enough memory to test?
10 12 0027 341 BNEQ 5$ ; Branch if yes A5
00000000'EF 9F 0029 342 $DS_PRINTB_S FMT_NO_ARRAYS ; Type no arrays to test message A5
00000000'9F 01 FB 002F 343 PUSHAB FMT_NO_ARRAYS
015C 31 0036 344 CALLS $$$N, @#DS$PRINTB
57 D4 0039 345 5$: BRW 120$ ; Exit test A5
58 D4 003B 346 10$: CLRL R7 ; Initialize pointer operand
56 00000000'EF D0 003D 347 MOVL FREE_PAGE_P,R6 ; Initialize address operand
66 D4 0044 348 20$: CLRL (R6) ; Write all zero's
56 00010000 8F 00000000'EF F1 0046 349 ACBL MAXPFN,#^X10000,R6,20$ ; Do for all existing rows
FFFO 0052
56 00000000'EF D0 0054 350 MOVL FREE_PAGE_P,R6 ; Initialize address operand
00000000'EF 01 D0 005B 351 30$: MOVL #1,EXP_UNC_ERR ; Set flag to catch uncorrectable error A5

```

```
00000000'EF 00000076'EF DE 0062 352 MOVAL 35$,RETURN_PC ; Set the return PC A5
                25 66 57 E0 006D 353 BBS R7,(R6),40$ ; Check for 1 or 0
                58 01 C8 0071 354 BISL #1,R8 ; Indicate bit was zero
                2B 11 0074 355 BRB 50$ ; Branch
                0076 356 35$: $DS_ERRHARD_S ,#0,ALLMOD,-
                0076 357 PRINT_UNX_UNC,- ; Error message A5
                0076 358 R6 ; Print unexpected message A5
                56 DD 0076 PUSHL R6
                00000000'EF DF 0078 PUSHAL PRINT_UNX_UNC
                00000000'EF DF 007E PUSHAL ALLMOD
                00 DD 0084 PUSHL #0
                01 DD 0086 PUSHL #$ER
                0088 ::; TEST 2, SUBTEST 0, ERROR 1
                00000000'9F 05 FB 0088 CALLS $$$M, a#DS$ERRHARD
                62 E0000000 8F D0 008F 359 MOVL #^XE0000000,CSRO(R2) ; Cleanup the controller
                360 40$: $DS_CKLOOP 30$ ; Scope Loop? A5
                00000000'9F C2 AF FA 0096 CALLG 30$, a#DS$CKLOOP
                58 02 C8 009E 361 BISL #2,R8 ; Indicate bit was one
                00A1 362 50$: $DS_CKLOOP 30$ ; Scope Loop? A5
                00000000'9F B7 AF FA 00A1 CALLG 30$, a#DS$CKLOOP
56 00010000 8F 00000000'EF F1 00A9 363 ACBL MAXPFN,#^X10000,R6,30$ ; Do for all rows
                FFA4 00B5
                56 00000000'EF D0 00B7 364 MOVL FREE_PAGE_P,R6 ; Initialize address pointer
                66 FFFFFFFF 8F D0 00BE 365 60$: MOVL #-1,(R6) ; Write all one's
56 00010000 8F 00000000'EF F1 00C5 366 ACBL MAXPFN,#^X10000,R6,60$ ; Do for all existing rows
                FFEB 00D1
                56 00000000'EF D0 00D3 367 MOVL FREE_PAGE_P,R6 ; Initialize address pointer
                00000000'EF 01 D0 00DA 368 70$: MOVL #1,EXP_UNC_ERR ; Set expected error flag A5
                00000000'EF 000000F5'EF DE 00E1 369 MOVAL 75$,RETURN_PC ; Set the return PC A5
                1E 66 57 E0 00EC 370 BBS R7,(R6),80$ ; Check for 1 or 0
                58 01 C8 00F0 371 BISL #1,R8 ; Indicate bit was zero
                24 11 00F3 372 BRB 90$ ; And branch
                00F5 373 75$: $DS_ERRHARD_S ,#0,ALLMOD,-
                00F5 374 PRINT_UNX_UNC,- ; Report error A5
                00F5 375 R6 ; Report Uncorrectable Error A5
                56 DD 00F5 PUSHL R6
                00000000'EF DF 00F7 PUSHAL PRINT_UNX_UNC
                00000000'EF DF 00FD PUSHAL ALLMOD
                00 DD 0103 PUSHL #0
                02 DD 0105 PUSHL #$ER
                0107 ::; TEST 2, SUBTEST 0, ERROR 2
                00000000'9F 05 FB 0107 CALLS $$$M, a#DS$ERRHARD
                010E 376 80$: $DS_CKLOOP 70$ ; Scope Loop? A5
                00000000'9F C9 AF FA 010E CALLG 70$, a#DS$CKLOOP
                58 02 C8 0116 377 BISL #2,R8 ; Indicate bit was one
                0119 378 90$: $DS_CKLOOP 70$ ; Scope Loop? A5
                00000000'9F BE AF FA 0119 CALLG 70$, a#DS$CKLOOP
56 00010000 8F 00000000'EF F1 0121 379 ACBL MAXPFN,#^X10000,R6,70$ ; Do for all existing rows
                FFAB 012D
                58 03 D1 012F 380 CMPL #3,R8 ; Did this bit toggle?
                53 13 0132 381 BEQL 110$ ; Branch if so
                56 00000000'EF D0 0134 382 MOVL FREE_PAGE_P,R6 ; Initialize address pointer
                66 D4 013B 383 95$: CLRL (R6) ; Write a zero to this location
                66 FFFFFFFF 8F D0 013D 384 MOVL #-1,(R6) ; Then write a one
                0144 385 $DS_ERRHARD_S ,#0,ALLMOD ; Report the error
                00 DD 0144 PUSHL #0
                00000000'EF DF 0146 PUSHAL ALLMOD
```



```

00000000 01A0 .SBTTL TEST 3: ROW SELECT BUS TEST
00000000 0000 .PSECT TEST_003, PAGE, NOWRT
00000000 0018 399 $DS_BGNTST <DEFAULT,ALL,EXHAUSTIVE>
00000000 0018 DATA_003: .LONG 0 ; TEST ARGUMENT TABLE TERMINATOR
00000000 001C TEST_003:: .WORD ^M<> ; ENTRY MASK
00000000 001C
00000000 001E 400 ;**
00000000 001E 401 ; Functional description:
00000000 001E 402 ;
00000000 001E 403 ; In this test the row select (bank select) is checked for stuck at
00000000 001E 404 ; zero or one faults. The memory array boards are organized in rows
00000000 001E 405 ; (banks) of 39 by 1 bit Ram Ic's. These Rams are either 16k or 64k
00000000 001E 406 ; depending upon the array type. Eack row (bank) is either 64kb (10000)
00000000 001E 407 ; hex or 256kb (40000) hex in density. The address selection circuit of
00000000 001E 408 ; the memory controller applies INT BUS MA <15:14> to the row select
00000000 001E 409 ; circuit of the memory array. The row select circuit decodes INT BUS MA
00000000 001E 410 ; <15:14>, thus generating the row address strobes RAS SEL <A:D> L which
00000000 001E 411 ; selects one of the four banks of rams. If there is a stuck or shorted
00000000 001E 412 ; bit in this row select, it would show up as a dual addressing
00000000 001E 413 ; syndrome. For instance, consider the case of row select 0 stuck at
00000000 001E 414 ; zero. Writing a unique pattern to address 0, row 0 (the 24 bit CMI
00000000 001E 415 ; address would be 000000), and then writing a second unique pattern
00000000 001E 416 ; to address 0, row 1 (CMI address 010000) would overwrite the first
00000000 001E 417 ; pattern. One way of detecting a stuck or shorted bit in this bus
00000000 001E 418 ; would be to write unique patterns in the same 14/16 bit address of
00000000 001E 419 ; each row, and then verify that all of the unique patterns were intact.
00000000 001E 420 ; A convenient set of patterns would be the sequence 0, 1,2, 3, ...
00000000 001E 421 ;
00000000 001E 422 ; The trouble with using this sequence of patterns is that a faulty RAM
00000000 001E 423 ; chip might cause the results of the test to be misleading. Therefore,
00000000 001E 424 ; to make the test tolerant to RAM faults, the values in the sequence
00000000 001E 425 ; are encoded into an error-correcting code. In this test (and in the
00000000 001E 426 ; following test) a Reed-Muller (32,6) code is employed. This code
00000000 001E 427 ; provides for a Hamming distance of 7 between codewords. Thus, the
00000000 001E 428 ; test tolerates a maximum of 6 faulty RAMs without invalidating the
00000000 001E 429 ; test results.
00000000 001E 430 ;
00000000 001E 431 ; Test algorithm:
00000000 001E 432 ;
00000000 001E 433 ; 1. Write a unique code word in each row (bank) of all present arrays
00000000 001E 434 ; under test. The code word is the boolean product of the row number
00000000 001E 435 ; and a Reed-Muller code word of (32,6). The assignment of the code-
00000000 001E 436 ; word to a row (bank) is arbitrary.
00000000 001E 437 ;
00000000 001E 438 ; 2. Read each word written in step 1, and decode them. A stuck or
00000000 001E 439 ; shorted row select bit will be indicated if the decoded word does
00000000 001E 440 ; not match its indexed value.
00000000 001E 441 ;--
00000000 001E 442 ;
00000000 *EF 00000029 *EF DE 001E 443 MOVAL 2$,UNX_UNC_PC ; Save PC for UNEX UNC ERROR A7
00000000 0029 444 2$:
00000000 *EF 00000000 *EF D5 0029 445 TSTL FREE_PAGE_P ; Enough memory to test?
00000000 10 12 002F 446 BNEQ 5$ ; Branch if yes A5
00000000 *EF 00000000 *EF 9F 0031 447 $DS_PRINTB_S FMT_NO_ARRAYS ; Type no arrays to test message A5
00000000 0031 447 PUSHAB FMT_NO_ARRAYS

```

```

00000000'9F 01 FB 0037 CALLS $$$N, a#DS$PRINTB ; Exit test
029A 31 003E 448 BRW 120$
0041 449 ;
0041 450 ; First, each row is written with its distinct code word. Also, each written
0041 451 ; word is retrieved and stored in a table for convenience in analysis.
0041 452 ; Note..TRANSITION and FLAG2 are loaded as result of JSB READ_MAP.
0041 453 ;
56 00000000'EF D0 0041 454 5$: MOVL FREE_PAGE_P,R6 ; Initialize row pointer
57 D4 0048 455 CLRL R7 ; Initialize row counter
59 D4 004A 456 CLRL R9 ; Initialize slot counter
00000000'EF D4 004C 457 CLRL FLAG1 ; Initialize 256kb array counter
00000000'EF D4 0052 458 CLRL FLAG2 ; Initialize 256kb array at TRANSITION
00000000'EF D4 0058 459 CLRL FLAG3 ;[3.0] Initialize
00000000'EF D4 005E 460 CLRL FLAG4 ; Initialize
00000000'EF 16 0064 461 JSB READ_MAP ; Find array type
00000000'EF49 FF 8F 91 006A 462 CMPB #-1,SLOT[R9] ; Check array type
57 13 0073 463 BEQL 8$ ; Branch if 256kb array
0075 464 ;
00000000'EF49 01 91 0075 465 CMPB #1,SLOT[R9] ;[3.0] Check array type
22 19 007D 466 BLSS 6$ ; Branch if 1024kb array
54 00000000'EF 56 05 14 EF 007F 467 EXTZV #20,#5,R6,FROW ; The # of rows used 4Mb arrays
00000000'EF 00000000'EF C3 0088 468 SUBL3 FROW,MAXROW,R4 ; Store the rows left
58 54 D0 0094 469 MOVL R4,R8 ; Save for later use
00000000'EF 01 D0 0097 470 MOVL #1,FLAG4
004B 31 009E 471 BRW 10$ ; Continue flow
00A1 472 ;
54 00000000'EF 56 05 12 EF 00A1 473 6$: EXTZV #18,#5,R6,FROW ; The # of rows used 1024kb arrays
00000000'EF 00000000'EF C3 00AA 474 SUBL3 FROW,MAXROW,R4 ; Store the rows left
58 54 D0 00B6 475 MOVL R4,R8 ; Save for later use
00B9 476 ;
00000000'EF 01 91 00B9 477 CMPB #1,FLAG4 ;[3.0] If first array not 4Mb then
2A 12 00C0 478 BNEQ 10$ ; it must be 1024kb
00000000'EF 02 D0 00C2 479 MOVL #2,FLAG3 ; Set flag
00C9 480 ;
0020 31 00C9 481 BRW 10$ ; Continue flow
54 00000000'EF 56 05 10 EF 00CC 482 8$: EXTZV #16,#5,R6,FROW ; The # of rows used for 256kb arrays
00000000'EF 00000000'EF C3 00D5 483 SUBL3 FROW,MAXROW,R4 ; Stores the rows left
58 54 D0 00E1 484 MOVL R4,R8 ; Save for later use
00000000'EF FF 8F 90 00E4 485 MOVB #-1,FLAG1 ; Set flag to indicate 256kb array
00EC 486 ; This section is where the codewords are written and any house keeping
00EC 487 ; that needs to be done.
00EC 488 ;
66 00000000'EF47 D0 00EC 489 10$: MOVL CODEWORDS[R7],(R6) ; Store a code word in a row
00000000'EF 01 D0 00F4 490 MOVL #1,EXP_UNC_ERR ; Set expected error flag
00000000'EF 00000108'EF DE 00FB 491 MOVAL 15$,RETURN_PC ; Setup the return PC
19 11 0106 492 BRB 20$ ;
0108 493 15$: $DS_ERRHARD_S ,#0,ALLMOD,-
0108 494 PRINT_UNX_UNC,-; Report error
0108 495 R6 ; Report failing address
56 DD 0108 PUSHL R6
00000000'EF DF 010A PUSHAL PRINT_UNX_UNC
00000000'EF DF 0110 PUSHAL ALLMOD
00 DD 0116 PUSHL #0
01 DD 0118 PUSHL #SER
011A ;::: TEST 3, SUBTEST 0, ERROR 1
00000000'9F 05 FB 011A CALLS $$$M, a#DS$ERRHARD
0121 496 20$: $DS_CKLOOP 10$ ; Scope Loop?

```

```
00000000'9F C8 AF FA 0121 CALLG 10$, a#DS$CKLOOP
                    57 D6 0129 497 INCL R7 ; Bump row counter once
00000000'EF FF 8F 91 012B 498 CMPB #-1,FLAG1 ; Find array type
                    51 13 0133 499 BEQL 25$ ; Branch if 256kb array else
                    0135 500 ;
00000000'EF 02 91 0135 501 CMPB #2,FLAG3 ;[3.0] If first array not 1024KB
                    25 13 013C 502 BEQL 23$ ; and not 256KB then must be
00000000'EF F1 013E 503 ACBL TRANSITION4,- ; 4Mb.
FFA0 56 00100000 8F 0144 504 #^X100000,R6,10$ ; Do all 4MB arrays
00000000'EF 00000000'EF D1 014C 505 CMPL TRANSITION4,TRANSITION
                    3B 13 0157 506 BEQL 30$
00000000'EF 02 D0 0159 507 MOVL #2,FLAG3
                    FF89 31 0160 508 BRW 10$
                    0163 509 ;
56 00040000 8F 00000000'EF F1 0163 510 23$: ACBL TRANSITION,#^X40000,R6,10$; Do for all existing rows 1024kb
                    FF7B 016F
00000000'EF FF 8F 91 0171 511 CMPB #-1,FLAG2 ; Find if there are any 256kb arrays
                    19 12 0179 512 BNEQ 30$ ; Branch if all 1024kb arrays
00000000'EF FF 8F 90 017B 513 MOVB #-1,FLAG1 ; Set flag for 1024/256kb mixture
                    FF66 31 0183 514 BRW 10$ ; Go write next codeword
56 00010000 8F 00000000'EF F1 0186 515 25$: ACBL MAXPFN,#^X10000,R6,10$ ; Do for all existing rows 256kb
                    FF58 0192
                    0194 516 ;
                    0194 517 ; Now the codewords written are retrieved and saved in a table called
                    0194 518 ; NOISY_WORDS. All pointers and counters are initialize again.
                    57 D4 0194 519 30$: CLRL R7 ; Initialize the row counter
                    59 D4 0196 520 CLRL R9 ; Initialize slot counter
00000000'EF D4 0198 521 CLRL FLAG1 ; Initialize 256kb array flag
00000000'EF D4 019E 522 CLRL FLAG2 ; Initialize 256kb array at TRANSITION
00000000'EF D4 01A4 523 CLRL FLAG3 ;[3.0] Initialize 256kb array flag
00000000'EF D4 01AA 524 CLRL FLAG4 ; Initialize 256kb array at TRANSITION
56 00000000'EF D0 01B0 525 MOVL FREE_PAGE_P,R6 ; Initialize the row pointer
00000000'EF 16 01B7 526 JSB READ_MAP ; Find array type in first slot
00000000'EF49 FF 8F 91 01BD 527 CMPB #-1,SLOT[R9] ; Check array type
                    14 13 01C6 528 BEQL 33$ ; Branch if 256kb array otherwise
                    01C8 529 ;
00000000'EF49 01 91 01C8 530 CMPB #1,SLOT[R9] ;[3.0] If not 4Mb then
                    12 13 01D0 531 BEQL 34$ ; must be 1024kb
00000000'EF 02 90 01D2 532 MOVB #2,FLAG3 ; Set flag
                    01D9 533 ;
                    0008 31 01D9 534 BRW 34$ ; Continue
00000000'EF FF 8F 90 01DC 535 33$: MOVB #-1,FLAG1 ; Set 256kb array flag
00000000'EF47 66 D0 01E4 536 34$: MOVL (R6),NOISY_WORDS[R7] ; Retrieve and save the codeword
00000000'EF 01 D0 01EC 537 MOVL #1,EXP_UNC_ERR ; Set expected error flag
00000000'EF 00000200'EF DE 01F3 538 MOVAL 35$,RETURN_PC ; Set up return PC
                    19 11 01FE 539 BRB 36$ ;
                    0200 540 35$: $DS_ERRHARD_S ,#0,ALLMOD,-
                    0200 541 PRINT_UNX_UNC,-; Report error
                    0200 542 R6 ; Report failing address
                    56 DD 0200 PUSHL R6
00000000'EF DF 0202 PUSHAL PRINT_UNX_UNC
00000000'EF DF 0208 PUSHAL ALLMOD
                    00 DD 020E PUSHL #0
                    02 DD 0210 PUSHL #SER
                    0212 ;:; TEST 3, SUBTEST 0, ERROR 2
00000000'9F 05 FB 0212 CALLS $$$M, a#DS$ERRHARD
                    0219 543 36$: $DS_CKLOOP 34$ ; Scope loop ?
```

```
00000000'9F C8 AF FA 0219 CALLG 34$, a#DS$CKLOOP
00000000'EF FF 8F 91 0221 544 INCL R7 ; Bump the row counter
57 D6 0222 545 CMPB #-1,FLAG1 ; Check array type 256k ?
00000000'EF 51 13 022B 546 BEQL 39$ ; Branch if 256kb array
022D 547 ;
00000000'EF 02 91 022D 548 CMPB #2,FLAG3 ;[3.0] If first array not 1024KB
25 13 0234 549 BEQL 37$ ; and not 256KB then must be
00000000'EF F1 0236 550 ACBL TRANSITION4,- ; Do all 4MB arrays
FFA0 56 00100000 8F 023C 551 #^X100000,R6,34$
00000000'EF 00000000'EF D1 0244 552 CMPL TRANSITION4,TRANSITION
3B 13 024F 553 BEQL 40$
00000000'EF 02 D0 0251 554 MOVL #2,FLAG3
FF89 31 0258 555 BRW 34$ ; Go read next codeword
025B 556 ;
56 00040000 8F 00000000'EF F1 025B 557 37$: ACBL TRANSITION,#^X40000,R6,34$; Do for all existing rows 1024kb
FF7B 0267
00000000'EF FF 8F 91 0269 558 CMPB #-1,FLAG2 ; Is ther any 256kb arrays ?
19 12 0271 559 BNEQ 40$ ; Branch if all 1024kb arrays
00000000'EF FF 8F 90 0273 560 MOVB #-1,FLAG1 ; Set flag for mixture 1024/256kb
FF66 31 027B 561 BRW 34$ ; Go read next codeword
56 00010000 8F 00000000'EF F1 027E 562 39$: ACBL MAXPFN,#^X10000,R6,34$ ; Do for all existing rows 256kb
FF58 028A
028C 563 ;
028C 564 ; Next, the retrieved codewords, now stored in the table 'NOISY_WORDS', are
028C 565 ; decoded.
028C 566 ;
00000000'EF 16 028C 567 40$: JSB DECODER ; Decode the codewords
0292 568 ;
0292 569 ; Now we have a table of decoded words. The table is scanned and any errors
0292 570 ; will reveal a stuck or shorted row-select bus.
0292 571 ;
0292 572 ;
00000000'EF47 57 D4 0292 572 CLRL R7 ; Initialize counter
57 D1 0294 573 50$: CMPL R7,DECODED_WORDS[R7] ; Does the decoded word = the index?
39 13 029C 574 BEQL 60$ ; Branch if so
54 00000000'EF47 D0 029E 575 MOVL DECODED_WORDS[R7],R4 ; This is actual data
53 57 D0 02A6 576 MOVL R7,R3 ; This is expected data
00000000'EF 57 C0 02A9 577 ADDL2 R7,FROW ; Point to actual row in failure
00000000'EF 02 90 02B0 578 MOVB #2,ERROR_TYPE ; Point to row select bus failure
02B7 579 $DS_ERRHARD_S ,#0,ALLMOD,- ; Report the error
02B7 580 PRINT
00000000'EF DF 02B7 PUSHAL PRINT
00000000'EF DF 02BD PUSHAL ALLMOD
00 DD 02C3 PUSHL #0
03 DD 02C5 PUSHL #SER
02C7 ;:: TEST 3, SUBTEST 0, ERROR 3
00000000'9F 04 FB 02C7 CALLS $$$M, a#DS$ERRHARD
02CE 581 $DS_CKLOOP 5$ ; Scope loop ?
00000000'9F FD6F CF FA 02CE CALLG 5$, a#DS$CKLOOP ; Do for all rows
B9 57 58 F2 02D7 582 60$: AOBLSS R8,R7,50$
02DB 583 120$: $DS_ENDTEST
50 01 D0 02DB MOVL #1, R0 ; NORMAL EXIT
02DE TEST_003_X::
00000000'9F 6E FA 02DE CALLG (SP), a#DS$BREAK
04 02E5 RET ; RETURN TO TEST SEQUENCER
02E6 584 $DS_PAGE
```

```

00000000 02E6 586
00000000 02E6 .SBTTL TEST 4: ADDRESS BUS TEST
0014 0014 .PSECT TEST_004, PAGE, NOWRT
0014 588 $DS_BGNTTEST <DEFAULT,ALL,EXHAUSTIVE>
00000000 0014 DATA_004:
0018 0018 .LONG 0 ; TEST ARGUMENT TABLE TERMINATOR
0000 0018 TEST_004:: .WORD ^M<> ; ENTRY MASK
001A 589 ;++
001A 590 ; Functional description:
001A 591 ;
001A 592 ; In this test the address bus is checked for bits stuck at one, stuck
001A 593 ; at zero, and shorted together. Each row of an array board consists of
001A 594 ; 39 16k by 1 bit rams for 256kb arrays or 64k by 1 for 1024kb arrays. A
001A 595 ; fourteen/sixteen bit address bus runs in parallel to all of the RAMs
001A 596 ; in a row. To ascertain there are any problems with this address bus, a
001A 597 ; scheme similar to that of Test 3 is employed.
001A 598 ;
001A 599 ; Test algorithm:
001A 600 ;
001A 601 ; 1. Write a unique pattern into each location of the address sequence
001A 602 ; 0, 1, 2, 4, 8, ..., 16384 for 16k rams or 65536 for 64k rams of each
001A 603 ; row of all present arrays. The patterns are Reed-Muller (32,6)
001A 604 ; codewords representing the values in the sequence 0, 1, 2, 3, ...
001A 605 ;
001A 606 ; 2. Read each word written in step 1 and decode. A stuck or shorted
001A 607 ; address bit will result in a mismatch of the decoded word and its
001A 608 ; expected value.
001A 609 ;
001A 610 ; This test tolerates a mazimum of 6 RAM failures per row without
001A 611 ; invalidating the test results.
001A 612 ;--
00000000'EF 00000025'EF DE 001A 614 MOVAL 1$,UNX_UNC_PC ; Save PC for UNEX UNC ERROR A7
00000000'EF 00000000'EF D5 0025 615 1$: TSTL FREE_PAGE_P ; Enough memory to test?
10 12 002B 616 BNEQ 2$ ; Branch if yes A5
00000000'EF 9F 002D 618 $DS_PRINTB_S FMT_NO_ARRAYS ; Type no arrays to test message A5
00000000'9F 01 FB 0033 PUSLAB FMT_NO_ARRAYS
01F0 31 003A 619 CALLS #$$N, a#DS$PRINTB
00000000'EF D4 003D 620 2$: BRW 80$ ; Exit test A5
00000000'EF D4 0043 621 CLRL FLAG1 ; Initialize 256kb array flag
0049 622 ; CLRL FLAG2 ; Initialize 256kb flag at TRANSITION
00000000'EF D4 0049 623 CLRL FLAG3 ;[3.0]
00000000'EF D4 004F 624 CLRL FLAG4
0055 625 ;
00000000'EF D4 0055 626 CLRL TEMP ; Initialize temporary storage
52 D4 005B 627 CLRL R2 ; If set tells us to go get codewords
5A D4 005D 628 CLRL R10 ; Initialize row counter
5B D4 005F 629 CLRL R11 ; Initialize slot counter
00000000'EF 00000000'EF D0 0061 630 MOVL FREE_PAGE_P,TEMP ; Set up to be used as a row counter
00000000'EF 16 006C 631 JSB READ_MAP ; Set up slot table for arrays
00000000'EF4B FF 8F 91 0072 632 CMPB #-1,SLOT[R11] ; Look at slot zero
1D 13 007B 633 BEQL 6$ ; Branch if 256kb array
007D 634 ;

```

```
00000000'EF4B 01 91 007D 635 CMPB #1,SLOT[R11] ;[3.0] If slot zero is 4Mb
                07 13 0085 636 BEQL 5$ ; then branch
00000000'EF 02 90 0087 637 4$: MOVB #2,FLAG3 ; else it is 1024kb - set flag
                57 D4 008E 638 ;
56 00000000'EF 0011 31 0090 639 5$: CLRL R7 ; Initialize address bit counter
                D0 0090 640 MOVL TEMP,R6 ; Initialize address pointer
00000000'EF FF 8F 90 009A 641 BRW 8$ ; Continue flow
                57 D4 00A2 642 6$: MOVB #-1,FLAG1 ; Set 256kb array flag
56 00000000'EF 58 04 D0 00A4 643 7$: CLRL R7 ; Initialize address bit counter
                59 D4 00AE 644 MOVL TEMP,R6 ; Initialize address pointer
                52 95 00B0 645 8$: MOVL #4,R8 ; Initialize address counter
                1D 12 00B2 646 CLRL R9 ; Initialize counter
66 00000000'EF47 D0 00B4 647 10$: TSTB R2 ; Time to read codewords written ?
00000000'EF 000000DC'EF 01 D0 00BC 648 BNEQ 11$ ; Branch if so otherwise
                0024 31 00CE 649 MOVL CODEWORDS[R7],(R6) ; Store a code word
00000000'EF47 66 D0 00D1 650 MOVL #1,EXP_UNC_ERR ; Set expected error flag
                0019 31 00D9 651 MOVAL 12$,RETURN_PC ; Setup the return PC
                00DC 652 BRW 14$ ;
                00DC 653 11$: MOVL (R6),NOISY_WORDS[R7] ; And immediately retrieve it
                00DC 654 BRW 14$ ;
                00DC 655 12$: $DS_ERRHARD_S ,#0,ALLMOD,- ;
                656 PRINT_UNX_UNC,- ; Report the error
                657 R6 ; Report the address
                56 DD 00DC PUSHL R6
00000000'EF DF 00DE PUSHAL PRINT_UNX_UNC
00000000'EF DF 00E4 PUSHAL ALLMOD
                00 DD 00EA PUSHL #0
                01 DD 00EC PUSHL #SER
                00EE ::: TEST 4, SUBTEST 0, ERROR 1
00000000'9F 05 FB 00EE CALLS $$$M, a#DS$ERRHARD
                00F5 658 14$: $DS_CKLOOP 10$ ; Scope Loop?
00000000'9F B8 AF FA 00F5 CALLG 10$, a#DS$CKLOOP
                57 D6 00FD 659 INCL R7 ; Bump address bit counter
                57 01 D1 00FF 660 CMPL #1,R7 ; Is this the first time?
                04 13 0102 661 BEQL 15$ ; Skip next two lines if so
58 58 01 78 0104 662 ASHL #1,R8,R8 ; Shift left once
                56 58 C0 0108 663 15$: ADDL R8,R6 ; And calculate new address
00000000'EF FF 8F 91 010B 664 CMPB #-1,FLAG1 ; Is slot a 256kb array ?
                39 13 0113 665 BEQL 25$ ; Branch if so
                0115 666 ;
00000000'EF 02 91 0115 667 CMPB #2,FLAG3 ;[3.0] Is slot a 1024kb array ?
                06 13 011C 668 BEQL 16$ ; Branch if so
8E 59 10 F3 011E 669 AOBLEQ #16,R9,10$ ; Do for all address bits 4Mb arrays
                04 11 0122 670 BRB 17$
                0124 671 ;
88 59 0E F3 0124 672 16$: AOBLEQ #14,R9,10$ ; Do for all address bits 1024kb arrays
                0128 673 ;
                0128 674 ; Now the codewords are all written, here testing is done to see if
                0128 675 ; the codewords written were retrieved and put in the table called
                0128 676 ; NOISY_WORDS.
                0128 677 ;
                52 95 0128 678 17$: TSTB R2 ; Time to retrieve the codewords ?
                06 12 012A 679 BNEQ 18$ ; Branch if not otherwise
52 01 D0 012C 680 MOVL #1,R2 ; Set flag
                FF5C 31 012F 681 BRW 5$ ; GO get codewords written
                0132 682 ;
00000000'EF 02 91 0132 683 18$: CMPB #2,FLAG3 ; Check array type
```

```
0139 684 ;
0139 685 ; Case of 4mb array: [3.0]
0139 686 ;
54 05 13 0139 687 BEQL 19$
11 013B 688 MOVL #17,R4 ; Number of codewords to decode
03 11 013E 689 BRB 20$
0140 690 ;
0140 691 ; Case of 1024kb array:
0140 692 ;
54 0F D0 0140 693 19$: MOVL #15,R4 ; Indicate # of words to decode 1024kb
00000000'EF 16 0143 694 20$: JSB DECODER ; Decode the codewords
0149 695 ;
0149 696 ; Now we have a table of decoded words. The table is scanned and any errors
0149 697 ; will reveal a stuck or shorted address bus bit. But first we have to find out
0149 698 ; what type of array we are testing.
0149 699 ;
03 57 D4 0149 700 CLRL R7 ; Initialize counter
0022 31 014B 701 BRW 50$ ;
03 59 0C F3 014E 702 25$: AOBLEQ #12,R9,30$ ; Do for all address bit 256kb arrays
0152 703 ;
0152 704 ; Now the codewords are written for 256kb arrays, testing is done to determine
0152 705 ; if the codewords written, were retrieved and put in the table called
0152 706 ; NOISY_WORDS.
0152 707 ;
0003 31 0152 708 BRW 35$ ; Go decode the codewords
FF58 31 0155 709 30$: BRW 10$ ; Store more codewords
52 95 0158 710 35$: TSTB R2 ; Is it time to retrieve?
06 12 015A 711 BNEQ 40$ ; Branch if already done
52 01 D0 015C 712 MOVL #1,R2 ; Set flag to indicate time to retrieve
FF40 31 015F 713 BRW 7$ ; Go get codewords
54 0D D0 0162 714 40$: MOVL #13,R4 ; # of codewords to decode for 256kb
00000000'EF 16 0165 715 JSB DECODER ; Decode the codewords 256kb arrays
57 D4 016B 716 CLRL R7 ; Initialize counter
0000 31 016D 717 45$: BRW 50$ ;
00000000'EF47 57 D1 0170 718 50$: CMLP R7,DECODED_WORDS[R7] ; Decoded word = address bit number?
2D 13 0178 719 BEQL 60$ ; Branch if so
54 00000000'EF47 D0 017A 720 MOVL DECODED_WORDS[R7],R4 ; This is actual data
53 57 D0 0182 721 MOVL R7,R3 ; This is expected data
00000000'EF 94 0185 722 CLRB ERROR_TYPE ; Point to address bus failure
018B 723 $DS_ERRHARD_S ,#0,ALLMOD,- ; Report the error
018B 724 PRINT
00000000'EF DF 018B PUSHAL PRINT
00000000'EF DF 0191 PUSHAL ALLMOD
00 DD 0197 PUSHL #0
02 DD 0199 PUSHL #SER
00000000'9F 04 FB 019B ::: TEST 4, SUBTEST 0, ERROR 2
03 11 01A2 725 BRB 60$
FFC9 31 01A4 726 58$: BRW 50$
00000000'EF FF 8F 91 01A7 727 60$: CMPB #-1,FLAG1 ; Is the array a 256kb ?
5F 13 01AF 728 BEQL 65$ ; If so branch otherwise
01B1 729 ;
00000000'EF 02 91 01B1 730 CMPB #2,FLAG3 ;[3.0] Is this 1024kb array?
29 13 01B8 731 BEQL 61$ ;
B2 57 10 F3 01BA 732 AOBLEQ #16,R7,50$ ; Do for all address bits 1024kb
00100000 8F 00000000'EF F1 01BE 733 ACBL TRANSITION,#^X100000,TEMP,62$ ; Keeps count of 256kb rows
002C 00000000'EF 01C9
```



```

0238 755 $DS_SBTTL <ECC LOGIC TEST>
0238 .SBTTL TEST 5: ECC LOGIC TEST
0238 $D2_SBTTL 005,<ECC LOGIC TEST>,<PAGE>
00000000 .PSECT TEST_005, PAGE, NOWRT
0014
0014 756 $DS_BGNTST <DEFAULT,ALL,EXHAUSTIVE>
0014 DATA_005:
00000000 0014 .LONG 0 ; TEST ARGUMENT TABLE TERMINATOR
0018 TEST_005::
0000 0018 .WORD ^M<> ; ENTRY MASK
001A 757 ;++
001A 758 ; Functional description:
001A 759 ;
001A 760 ; In this test the ECC logic is functionally verified.
001A 761 ;
001A 762 ; Test algorithm:
001A 763 ;
001A 764 ; 1. Check first that the diagnostic bits ECC Disable Mode, Page Mode,
001A 765 ; Diagnostic Check Mode, and Enable Interrupt which are all
001A 766 ; in CSR1, are not stuck at 0 or 1.
001A 767 ;
001A 768 ; 2. Next, check that page mode works. To do this, we put the
001A 769 ; controller in ECC Disable Mode and Page Mode, write a location in
001A 770 ; the given page with a pattern. This stores the check bits into
001A 771 ; CSR1<06:00>. The check bits are read back again, checking that
001A 772 ; they are as expected. Now turn off Page Mode and read the check
001A 773 ; bits in CSR1. With page mode off, the pattern will change with any
001A 774 ; read from memory, including an instruction-stream fetch.
001A 775 ;
001A 776 ; 3. Determine if disabling single-bit error reporting works. Turn off
001A 777 ; the report single-bit error bit in CSR1. Force a single-bit error,
001A 778 ; as in step 5. Check that no interrupt occurred, and that the
001A 779 ; single-bit error flag did not set in CSRO.
001A 780 ;
001A 781 ; 4. Write a series of patterns into memory in ECC Disable Mode. For
001A 782 ; each write, check the contents of CSR1<6:0> (the check bits). Read
001A 783 ; each location, and verify the check bits for each read cycle. The
001A 784 ; patterns are chosen to verify each gate in each checkbit generator.
001A 785 ;
001A 786 ; 5. Put the controller into ECC Disable Mode and write a pattern into
001A 787 ; memory. This stores the generated checkbits into CSR1. Turn off
001A 788 ; ECC disable Mode and write a new pattern into the same location
001A 789 ; (the pattern is chosen to simulate a single bit error in the first
001A 790 ; pattern written). Put the controller into Diagnostic Check Mode
001A 791 ; and read the pattern. This should cause the correctable error flag
001A 792 ; to set, and the page address of the error to be logged. Check that
001A 793 ; the error syndrome is correct. Check also that the failing bit was
001A 794 ; corrected. Check also that bit 0 in the CPU BUS ERROR register sets.
001A 795 ; Repeat for all 32 bits for a one and a zero.
001A 796 ;
001A 797 ; 6. Turn off Diagnostic Check Mode, and write a third pattern into the
001A 798 ; same location which will cause a double bit error when Diagnostic
001A 799 ; Check Mode is turned on again. Check that the uncorrectable error
001A 800 ; flag is set, and that the error is properly logged in CSRO. Check
001A 801 ; that the error syndrome is correct. Check that memory did not
001A 802 ; change.
001A 803 ;

```

```

001A 804 ;
001A 805 ;
001A 806 ;
001A 807 ;
001A 808 ;
001A 809 ;
001A 810 ;
001A 811 ;
001A 812 ;
001A 813 ;
001A 814 ;
001A 815 ;
001A 816 ;
001A 817 ;
001A 818 ;
001A 819 ;
001A 820 ;
001A 821 ;
001A 822 ;
001A 823 ;
001A 824 ;
001A 825 ;
001A 826 ;
001A 827 ;
001A 828 ;
001A 829 ;
001A 830 ;
001A 831 ;
001A 832 ;
001A 833 ;
001A 834 ;
001A 835 ;
001A 836 ;
001A 837 ;
001A 838 ;
001A 839 ;
001A 840 ;
001A 841 ;
0021 842 ;
0021 843 ;
002C 844 ;
002C 845 ;
002C 846 ;
002C 847 ;
002C 848 ;
002C 849 ;
0037 850 ;
003E 851 ;
0041 852 ;
0045 853 ;
0047 854 ;
004B 855 ;
004D 856 ;
004D 857 ;
004D 858 ;

```

7. With the uncorrectable error flag still set, force an additional error in a different page and check to see if the uncorrectable error, information lost flag is set. Check that the page address did not change. Also check that BUS ERROR REGISTER has the LOST ERROR bit set (bit 1). M2 A2

8. Force a single bit error, as in step 5, with the single bit error interrupt enabled. Check that the interrupt occurred. A2 A2

9. Determine if disabling single bit error reporting also inhibits the setting of the error bit in CSRO and latching of the address in CSRO. Perform step 3 and check that bit <29> is clear and that bits <23:9> are unchanged in CSRO. A2 A2 A2 A2

10. Check all combinations of two bits in error in a field of zero's and a field of one's. The data patterns used will be as follows:

	Field of Zero's	Field of One's	
00000003	00000003	FFFFFFFFC	A2
00000005	00000005	FFFFFFFA	A2
80000001	80000001	7FFFFFFE	A2
00000006	00000006	FFFFFFF9	A2
00000009	00000009	FFFFFFF5	A2
80000002	80000002	7FFFFFFD	A2
C0000000	C0000000	3FFFFFFF	A2

11. Force single-bit errors by changing the check bits for a given pattern. Check that the error is detected and that the syndrome is correct. This is done by loading a data pattern whose check bits are all zero (or one) and floating a zero (or one) thru the check bits. The syndrome bit ascertained will correspond to the check bit be tested. A2 A2 A2 A2 A2 A2

```

--
52 00F20000 8F D0 001A 841 MOVL #CONTROLLER,R2 ; Get memory controller address
00000000'9F 00000000'EF FA 0021 842 $DS_BGNSUB
0021 T5_S1:: CALLG $$$, a#DS$BGNSUB
002C 843 ;
002C 844 ; Part 1
002C 845 ;
002C 846 ; Check that ECC Disable Mode, Diagnostic Check Mode, and Page Mode bits are
002C 847 ; not stuck at zero or one in CSR1.
002C 848 ;
00000000'EF 00000037'EF DE 002C 849 MOVAL 10$,UNX_UNC_PC ; Save PC for UNEX UNCORRECTABLE ERROR A7
62 E0000000 8F D0 0037 850 MOVL #^XE0000000,CSR0(R2) ; Cleanup the controller
04 A2 D4 003E 851 CLRL CSR1(R2) ; Make all bits in CSR1 zero
54 04 A2 D0 0041 852 MOVL CSR1(R2),R4 ; Get contents of register
53 D4 0045 853 CLRL R3 ; Expect all bits clear
55 54 53 CD 0047 854 XORL3 R3,R4,R5 ; Are all bits clear?
25 13 004B 855 BEQL 20$ ; Branch if so
004D 856 $DS_ERRHARD_S ,#0,CONTROL,-
004D 857 PRINT_GENERAL,#4,-; Otherwise, report the error
004D 858 #FMT_ECC_8,R3,R4,R5

```

```

      55 DD 004D          PUSHL R5
      54 DD 004F          PUSHL R4
      53 DD 0051          PUSHL R3
00000000'8F DD 0053          PUSHL #FMT_ECC_8
      04 DD 0059          PUSHL #4
00000000'EF DF 005B          PUSHAL PRINT_GENERAL
00000000'EF DF 0061          PUSHAL CONTROL
      00 DD 0067          PUSHL #0
      01 DD 0069          PUSHL #SER
00000000'9F 09 FB 006B          ::: TEST 5, SUBTEST 1, ERROR 1
      09 FB 006B          CALLS $$$M, a#DS$ERRHARD
00000000'9F C2 AF FA 0072 859 20$: $DS_CKLOOP 10$ ; Scope loop?
04 A2 0C000000 8F C8 007A 860 30$: BISL #PM!DCM,CSR1(R2) ; Set Diag Ck Mode and Page Mode
      54 04 A2 D0 0082 861 MOVL CSR1(R2),R4 ; Get contents of register
      04 A2 D4 0086 862 CLRL CSR1(R2) ; Turn the bits off again
      53 0C000000 8F C8 0089 863 BISL #PM!DCM,R3 ; Expect DCM and PM set
      55 54 53 CD 0090 864 XORL3 R3,R4,R5 ; Are they set?
      25 13 0094 865 BEQL 40$ ; Branch if so
      0096 866 $DS_ERRHARD_S ,#0,CONTROL,-
      0096 867 PRINT_GENERAL,#4,-; Otherwise, report the error
      0096 868 #FMT_ECC_9,R3,R4,R5
      55 DD 0096          PUSHL R5
      54 DD 0098          PUSHL R4
      53 DD 009A          PUSHL R3
00000000'8F DD 009C          PUSHL #FMT_ECC_9
      04 DD 00A2          PUSHL #4
00000000'EF DF 00A4          PUSHAL PRINT_GENERAL
00000000'EF DF 00AA          PUSHAL CONTROL
      00 DD 00B0          PUSHL #0
      02 DD 00B2          PUSHL #SER
00000000'9F 09 FB 00B4          ::: TEST 5, SUBTEST 1, ERROR 2
      09 FB 00B4          CALLS $$$M, a#DS$ERRHARD
00000000'9F BC AF FA 00BB 869 40$: $DS_CKLOOP 30$ ; Scope loop?
04 A2 1A000000 8F C8 00C3 870 50$: BISL #ERCE!PM!EDM,CSR1(R2) ; Set ECC disable mode and page mode M2 M
      54 04 A2 D0 00CB 871 ; and Enable Interrupt A2 M
04 A2 1A000000 8F CA 00CF 872 MOVL CSR1(R2),R4 ; Get contents of register
      53 1A000000 8F C8 00D7 873 BICL #ERCE!PM!EDM,CSR1(R2) ; And turn off the bits M2 M
      55 54 53 CD 00E0 874 CLRL R3 ; Clear expected results register
      25 13 00D9 875 BISL #ERCE!PM!EDM,R3 ; Expect EDM and PM and ERCE set M2 M
      00E6 876 XORL3 R3,R4,R5 ; Are they set?
      00E6 877 BEQL 60$ ; Branch if so
      00E6 878 $DS_ERRHARD_S ,#0,CONTROL,-
      00E6 879 PRINT_GENERAL,#4,-; Else, report the error
      880 #FMT_ECC_10,R3,R4,R5
      55 DD 00E6          PUSHL R5
      54 DD 00E8          PUSHL R4
      53 DD 00EA          PUSHL R3
00000000'8F DD 00EC          PUSHL #FMT_ECC_10
      04 DD 00F2          PUSHL #4
00000000'EF DF 00F4          PUSHAL PRINT_GENERAL
00000000'EF DF 00FA          PUSHAL CONTROL
      00 DD 0100          PUSHL #0
      03 DD 0102          PUSHL #SER
00000000'9F 09 FB 0104          ::: TEST 5, SUBTEST 1, ERROR 3
      09 FB 0104          CALLS $$$M, a#DS$ERRHARD
```



```
04 A2 D4 01E3 912 CLRL CSR1(R2) ; Turn off ECC disable mode
01E6 913 $DS_ENDSUB
01E6 T5_S2_X:
00000000'9F 0000000C'EF FA 01E6 CALLG $$$, a#DS$ENDSUB
01F1 914 $DS_BGNSUB
00000000'9F 00000018'EF FA 01F1 T5_S3:: CALLG $$$, a#DS$BGNSUB
01F1
01FC 915 ;
01FC 916 ; Part 3
01FC 917 ;
01FC 918 ; Check that disabling single bit error interrupts works.
01FC 919 ;
00000000'EF 00000212'EF DE 01FC 920 MOVAL 5$,UNX_UNC_PC ; Save PC for UNEX UNCORRECTABLE ERROR A7
00000000'EF 00000276'EF DE 0207 921 MOVAL 15$,RETURN_PC
0212 922 5$:
04 A2 62 E0000000 8F D0 0212 923 10$: MOVL #^XE0000000,CSR0(R2) ; Clean up controller
50 00000000'EF DE 0219 924 MOVAL TEMP,R0 ; Get address of location to write
04 A2 50 000001FF 8F CB 0220 925 BICL3 #^X1FF,R0,CSR1(R2) ; Put page address into pm address
04 A2 0A000000 8F C8 0229 926 BISL #PM!EDM,CSR1(R2) ; Turn on ECC disable mode
00000000'EF 01 D0 0231 927 MOVL #1,TEMP ; Write a pattern in this page
50 00000000'EF D0 0238 928 MOVL TEMP,R0 ; Read it
04 A2 02000000 8F CA 023F 929 BICL #EDM,CSR1(R2) ; Turn off ECC disable mode
00000000'EF D4 0247 930 CLRL TEMP ; Simulate a single bit error
04 A2 10000000 8F CA 024D 931 BICL #ERCE,CSR1(R2) ; Disable the interrupt A3
00000000'EF 01 90 0255 932 MOVVB #1,EXP_COR_ERR ; Indicate correctable error expected
00000000'EF 00000101 8F D0 025C 933 MOVL #^X101,EXP_UNC_ERR
04 A2 0C000000 8F C8 0267 934 BISL #PM!DCM,CSR1(R2) ; Turn on Diagnostic Check mode
54 00000000'EF D0 026F 935 MOVL TEMP,R4 ; Read the 'corrupted' word
04 A2 0C000000 8F CA 0276 936 15$: BICL #PM!DCM,CSR1(R2) ; Turn off DCM
29 00000000'EF 00 E4 027E 937 BBSC #0,EXP_UNC_ERR,17$ ; Branch if no double bit
0286 938 $DS_ERRHARD_S ,#0,ALLMOD,-
0286 939 CORRUPT_TEST_LOC,-
0286 940 #TEMP
PUSHL #TEMP
PUSHAL CORRUPT_TEST_LOC
PUSHAL ALLMOD
PUSHL #0
PUSHL #SER
PUSHL #SER
::: TEST 5, SUBTEST 3, ERROR 1
00000000'9F 05 FB 029C 941 CALLS $$$M, a#DS$ERRHARD
00000000'9F FF6B CF FA 02A3 941 $DS_CKLOOP 10$
02AC 942 $DS_ABORT 10$, a#DS$CKLOOP
50 D4 02AC 942 CLRL R0 ; Abort test corrupted test location
04 02AE RET ; SEND A WARNING TO THE SUPERVISOR
00000000'EF 95 02AF 943 17$: TSTB EXP_COR_ERR ; TERMINATE TEST
1F 12 02B5 944 BNEQ 20$ ; Expect no interrupt occurred M2
02B7 945 $DS_ERRHARD_S ,#0,CONTROL,- ; Branch if so M2
02B7 946 PRINT_GENERAL,#1,-; Report the error
02B7 947 #FMT_ECC_12
#FMT_ECC_12
PUSHL #FMT_ECC_12
PUSHL #1
PUSHAL PRINT_GENERAL
PUSHAL CONTROL
PUSHL #0
PUSHL #SER
::: TEST 5, SUBTEST 3, ERROR 2
```

```
00000000'9F 06 FB 02CF          CALLS   $$$M, a#DS$ERRHARD
00000000'9F FF38 CF FA 02D6 948 20$: $DS_CKLOOP 10$ ; Scope loop?
00000000'9F 00000018'EF FA 02D6          CALLG   10$, a#DS$CKLOOP
00000000'9F 00000018'EF FA 02DF 949          $DS_ENDSUB
00000000'9F 00000018'EF FA 02DF          T5_S3_X:
00000000'9F 00000024'EF FA 02EA 950          CALLG   $$$, a#DS$ENDSUB
00000000'9F 00000024'EF FA 02EA          $DS_BGNSUB
00000000'9F 00000024'EF FA 02EA          CALLG   $$$, a#DS$BGNSUB
00000000'EF 00000300'EF DE 02F5 951 ;
00000000'EF 00000300'EF DE 02F5 952 ; Part 4
00000000'EF 00000300'EF DE 02F5 953 ;
00000000'EF 00000300'EF DE 02F5 954 ; This verifies the parity tree networks in the ECC gate arrays.
00000000'EF 00000300'EF DE 02F5 955 ;
00000000'EF 00000300'EF DE 02F5 956          MOVAL   5$, UNX_UNC_PC ; Save PC for UNEX UNCORRECTABLE ERROR A7
00000000'EF 00000300'EF DE 0300 957 5$:
00000000'EF 00000300'EF DE 0300 958          CLRL   R6 ; Initialize index
00000000'EF 00000300'EF DE 0302 959          MOVAL  TEMP, R0 ; Get address of 'TEMP'
00000000'EF 00000300'EF DE 0309 960          BICL3  #^X1FF, R0, CSR1(R2) ; Put page address in pm address
00000000'EF 00000300'EF DE 0312 961          BISL   #PM!EDM, CSR1(R2) ; Put memory into ECC disable mode
00000000'EF 00000300'EF DE 031A 962 10$:          MOVL   CKBIT[R6], TEMP ; Write a pattern to memory
00000000'EF 00000300'EF DE 0326 963          MOVZBL CK[R6], R3 ; Get the expected value
00000000'EF 00000300'EF DE 032E 964          EXTZV #0, #7, CSR1(R2), R4 ; Get the actual value
00000000'EF 00000300'EF DE 0334 965          XORL3  R3, R4, R5 ; Expected = actual?
00000000'EF 00000300'EF DE 0338 966          BEQL   20$ ; Branch if so
00000000'EF 00000300'EF DE 033A 967          $DS_ERRHARD_S , #0, CONTROL, -
00000000'EF 00000300'EF DE 033A 968          PRINT_GENERAL, #4, - ; Report the error
00000000'EF 00000300'EF DE 033A 969          #FMT_ECC_1, - ; And all the parameters
00000000'EF 00000300'EF DE 033A 970          R3, R4, R5
00000000'EF 00000300'EF DE 033A 971          PUSHL  R5
00000000'EF 00000300'EF DE 033A 972          PUSHL  R4
00000000'EF 00000300'EF DE 033A 973          PUSHL  R3
00000000'EF 00000300'EF DE 033A 974          PUSHL  #FMT_ECC_1
00000000'EF 00000300'EF DE 033A 975          PUSHL  #4
00000000'EF 00000300'EF DE 033A 976          PUSHAL PRINT_GENERAL
00000000'EF 00000300'EF DE 033A 977          PUSHAL CONTROL
00000000'EF 00000300'EF DE 033A 978          PUSHL  #0
00000000'EF 00000300'EF DE 033A 979          PUSHL  #SER
00000000'9F 09 FB 0358          ::: TEST 5, SUBTEST 4, ERROR 1
00000000'9F 09 FB 0358          CALLS   $$$M, a#DS$ERRHARD
00000000'9F B8 AF FA 035F 971 20$: $DS_CKLOOP 10$ ; Scope loop?
00000000'9F B8 AF FA 035F          CALLG   10$, a#DS$CKLOOP
00000000'9F B8 AF FA 0367 972          MOVL   TEMP, R0 ; Read it
00000000'9F B8 AF FA 036E 973          BICL3  #^C<^X7F>, CSR1(R2), R4 ; Get the actual value
00000000'9F B8 AF FA 0377 974          XORL3  R3, R4, R5 ; Expected = actual?
00000000'9F B8 AF FA 037B 975          BEQL   30$ ; Branch if so
00000000'9F B8 AF FA 037D 976          $DS_ERRHARD_S , #0, CONTROL, -
00000000'9F B8 AF FA 037D 977          PRINT_GENERAL, #4, -
00000000'9F B8 AF FA 037D 978          #FMT_ECC_1, - ; And all the parameters
00000000'9F B8 AF FA 037D 979          R3, R4, R5
00000000'9F B8 AF FA 037D          PUSHL  R5
00000000'9F B8 AF FA 037D          PUSHL  R4
00000000'9F B8 AF FA 037D          PUSHL  R3
00000000'9F B8 AF FA 037D          PUSHL  #FMT_ECC_1
00000000'9F B8 AF FA 037D          PUSHL  #4
00000000'9F B8 AF FA 037D          PUSHAL PRINT_GENERAL
00000000'9F B8 AF FA 037D          PUSHAL CONTROL
00000000'9F B8 AF FA 0381          DD 0381
00000000'9F B8 AF FA 0383          DD 0383
00000000'9F B8 AF FA 0389          DD 0389
00000000'9F B8 AF FA 038B          DD 038B
00000000'9F B8 AF FA 0391          DD 0391
```

```
00 DD 0397          PUSHL #0
02 DD 0399          PUSHL #SER
00000000'9F 09 FB 039B          ::: TEST 5, SUBTEST 4, ERROR 2
00000000'9F  FF74 CF  FA 03A2 980 30$:  $DS_CKLOOP 10$          ; Scope loop?
02 56 31  F3 03AB 981          AOBLEQ #49,R6,40$          ; Do for all ECC patterns
03 11 03AF 982          BRB 50$          ; Done
04 A2 0A000000 8F  CA 03B1 983 40$:  BRW 10$          ; Go back and do the next one
04 A2 0A000000 8F  CA 03B4 984 50$:  BICL #PM!EDM,CSR1(R2) ; Turn off ECC disable mode
03BC 985          $DS_ENDSUB
00000000'9F 00000024'EF FA 03BC          T5_S4_X:
03C7 986          $DS_BGNSUB          CALLG $$$, a#DS$ENDSUB
00000000'9F 00000030'EF FA 03C7          T5_S5::          CALLG $$$, a#DS$BGNSUB
03D2 987          ;
03D2 988          ; Part 5
03D2 989          ;
03D2 990          ; Check that the controller can correct any single bit error.
03D2 991          ;
00000000'EF 000003E8'EF DE 03D2 992          MOVAL 5$,UNX_UNC_PC          ; Save PC for UNEX UNCORRECTABLE ERROR A7
00000000'EF 00000449'EF DE 03DD 993          MOVAL 15$,RETURN_PC
03E8 994 5$:
03E8 995          CLRL R6          ; Initialize counter
03EA 996          MOVL #1,R7          ; Initialize test operand
04 A2 62 E0000000 8F  D0 03ED 997 10$:  MOVL #^XE0000000,CSR0(R2) ; Cleanup the controller
04 A2 50 00000000'EF DE 03F4 998          MOVAL TEMP,R0          ; Get address of location to write
04 A2 50 000001FF 8F  CB 03FB 999          BICL3 #^X1FF,R0,CSR1(R2) ; And put page address into pm address
04 A2 0A000000 8F  C8 0404 1000          BISL #PM!EDM,CSR1(R2) ; Turn on ECC disable mode
00000000'EF 57  D0 040C 1001          MOVL R7,TEMP          ; Write a location in memory
04 A2 50 00000000'EF D0 0413 1002          MOVL TEMP,R0          ; Read it
04 A2 02000000 8F  CA 041A 1003          BICL #EDM,CSR1(R2) ; Turn off ECC disable mode
00000000'EF D4 0422 1004          CLRL TEMP          ; Simulate a single-bit failure
00000000'EF 00000101 8F D0 0428 1005          MOVL #^X101,EXP_UNC_ERR
00000000'EF 01  D0 0433 1006          MOVL #1,EXP_COR_ERR          ; Indicate correctable error expected M2 M
04 A2 1C000000 8F  C8 043A 1007          BISL #ERCE!PM!DCM,CSR1(R2) ; Turn on page mode/diag check mode M3
04 A2 54 00000000'EF D0 0442 1008          MOVL TEMP,R4
04 A2 0C000000 8F  CA 0449 1009 15$:  BICL #PM!DCM,CSR1(R2) ; Turn off page mode/diag check mode
29 00000000'EF 00  E4 0451 1010          BBSC #0,EXP_UNC_ERR,17$ ; Branch if no double bit
0459 1011          $DS_ERRHARD_S ,#0,ALLMOD,-
0459 1012          CORRUPT_TEST_LOC,-
0459 1013          #TEMP
00000000'8F DD 0459          PUSHL #TEMP
00000000'EF DF 045F          PUSHAL CORRUPT_TEST_LOC
00000000'EF DF 0465          PUSHAL ALLMOD
00 DD 046B          PUSHL #0
01 DD 046D          PUSHL #SER
00000000'9F 05 FB 046F          ::: TEST 5, SUBTEST 5, ERROR 1
00000000'9F  FF73 CF  FA 0476 1014          $DS_CKLOOP 10$
047F 1015          $DS_ABORT          CALLG $$$, a#DS$CKLOOP
047F 1015          TEST          ; Abort test corrupted test location
047F 1015          CLRL R0          ; SEND A WARNING TO THE SUPERVISOR
0481          RET          ; TERMINATE TEST
0482 1016 17$:  MOVL R7,R3          ; Expect correction to have occurred
0485 1017          XORL3 R3,R4,R5          ; Expected = received?
```

```

                25    13    0489    1018    BEQL    20$                ; Branch if so
                048B    1019    $DS_ERRHARD_S    ,#0,CONTROL,-
                048B    1020    PRINT_GENERAL,#4,-        ; Report the error
                048B    1021    #FMT_ECC_2,R3,R4,R5
                55    DD    048B    PUSHL    R5
                54    DD    048D    PUSHL    R4
                53    DD    048F    PUSHL    R3
                00000000'8F    DD    0491    PUSHL    #FMT_ECC_2
                04    DD    0497    PUSHL    #4
                00000000'EF    DF    0499    PUSHAL   PRINT_GENERAL
                00000000'EF    DF    049F    PUSHAL   CONTROL
                00    DD    04A5    PUSHL    #0
                02    DD    04A7    PUSHL    #SER
                04A9    ;;; TEST 5, SUBTEST 5, ERROR 2
                00000000'9F    09    FB    04A9    CALLS    $$$M, a#DS$ERRHARD
                04B0    1022    20$:    $DS_CKLOOP    10$                ; Scope loop?
                00000000'9F    FF39    CF    FA    04B0    CALLG    10$, a#DS$CKLOOP
                54    62    1F000180    8F    CB    04B9    1023    BICL3    #^X1F000180,CSR0(R2),R4 ; Read CSR 0, masking unused bits
                53    00000000'EF    DE    04C1    1024    MOVAL    TEMP,R3                ; Get address of 'error'
                53    000001FF    8F    CA    04C8    1025    BICL2    #^X1FF,R3                ; Mask all but page address
                50    00000000'EF46    9A    04CF    1026    MOVZBL   SYNDROME[R6],R0        ; Get expected syndrome bits
                53    50    C8    04D7    1027    BISL    R0,R3                ; Put syndrome bits in expected result
                53    20000000    8F    C8    04DA    1028    BISL    #CE,R3                ; Also, expect correctable error flag
                55    54    53    CD    04E1    1029    XORL3   R3,R4,R5                ; Expected = actual?
                25    13    04E5    1030    BEQL    25$                ; Branch if so
                04E7    1031    $DS_ERRHARD_S    ,#0,CONTROL,-
                04E7    1032    PRINT_GENERAL,#4,-        ; Report the error
                04E7    1033    #FMT_ECC_13,R3,R4,R5
                55    DD    04E7    PUSHL    R5
                54    DD    04E9    PUSHL    R4
                53    DD    04EB    PUSHL    R3
                00000000'8F    DD    04ED    PUSHL    #FMT_ECC_13
                04    DD    04F3    PUSHL    #4
                00000000'EF    DF    04F5    PUSHAL   PRINT_GENERAL
                00000000'EF    DF    04FB    PUSHAL   CONTROL
                00    DD    0501    PUSHL    #0
                03    DD    0503    PUSHL    #SER
                0505    ;;; TEST 5, SUBTEST 5, ERROR 3
                00000000'9F    09    FB    0505    CALLS    $$$M, a#DS$ERRHARD
                050C    1034    25$:    $DS_CKLOOP    10$                ; Scope loop?
                00000000'9F    FEDD    CF    FA    050C    CALLG    10$, a#DS$CKLOOP
                58    00000000'EF    01    8D    0515    1035    XORB3   #1,BUS_ERR_REG,R8        ; Corrected data bit set in BUS ERR reg? A2
                29    13    051D    1036    BEQL    27$                ; Branch if yes
                051F    1037    $DS_ERRHARD_S    ,#0,CPU,-
                051F    1038    PRINT_GENERAL,#4,-        ; Report the error
                051F    1039    #FMT_CPU_1,#1,BUS_ERR_REG,R8 ; A2 D3
                58    DD    051F    PUSHL    R8
                00000000'EF    DD    0521    PUSHL    BUS_ERR_REG
                01    DD    0527    PUSHL    #1
                00000000'8F    DD    0529    PUSHL    #FMT_CPU_1
                04    DD    052F    PUSHL    #4
                00000000'EF    DF    0531    PUSHAL   PRINT_GENERAL
                00000000'EF    DF    0537    PUSHAL   CPU
                00    DD    053D    PUSHL    #0
                04    DD    053F    PUSHL    #SER
                0541    ;;; TEST 5, SUBTEST 5, ERROR 4
                00000000'9F    09    FB    0541    CALLS    $$$M, a#DS$ERRHARD
```

```

00000000'9F  FEA1 CF  FA 0548 1040 27$:  $DS_CKLOOP 10$ ; Scope loop? A2 D
                                CALLG 10$, a#DS$CKLOOP
                                0551 1041
                                0551 1042 ROTL #1,R7,R7 ; Prepare to check next bit
                                FE92 56 01 1F F1 0555 1043 ACBL #31,#1,R6,10$ ; Do for all 32 bits
                                055B 1044
                                055B 1045 ;
                                055B 1046 ; Float a zero thru the word, forcing a single bit error.
                                055B 1047 ;
00000000'EF  000005C0'EF  DE 055B 1048 MOVAL 35$,RETURN_PC
                                56 D4 0566 1049 CLRL R6 ; Initialize counter
                                57 FFFFFFFE 8F D0 0568 1050 MOVL #-2,R7 ; Initialize test operand M2
                                62 E0000000 8F D0 056F 1051 30$: MOVL #^XE0000000,CSR0(R2) ; Cleanup the controller A2
04 A2 0A000000 8F C8 0576 1052 BISL #PM!EDM,CSR1(R2) ; Turn on ECC disable mode M2
                                00000000'EF  57 D0 057E 1053 MOVL R7,TEMP ; Write a location in memory
                                50 00000000'EF  D0 0585 1054 MOVL TEMP,R0 ; Read it
04 A2 02000000 8F CA 058C 1055 BICL #EDM,CSR1(R2) ; Turn off ECC disable mode
00000000'EF  FFFFFFFF 8F D0 0594 1056 MOVL #-1,TEMP ; Simulate a single-bit failure
00000000'EF  00000101 8F D0 059F 1057 MOVL #^X101,EXP_UNC_ERR
                                00000000'EF  01 D0 05AA 1058 MOVL #1,EXP_COR_ERR ; Indicate correctable error expected M2 M
04 A2 1C000000 8F C8 05B1 1059 BISL #ERCE!PM!DCM,CSR1(R2) ; Turn on page mode/diag check mode M3
                                54 00000000'EF  D0 05B9 1060 MOVL TEMP,R4 ; Read the corrupted word
04 A2 0C000000 8F CA 05C0 1061 35$: BICL #PM!DCM,CSR1(R2) ; Turn off page mode/diag check mode
29 00000000'EF  00 E4 05C8 1062 BBSC #0,EXP_UNC_ERR,37$ ; Branch if no double bit
                                05D0 1063 $DS_ERRHARD_S ,#0,ALLMOD,-
                                05D0 1064 CORRUPT_TEST_LOC,-
                                05D0 1065 #TEMP
                                DD 05D0 PUSHL #TEMP
                                DF 05D6 PUSHAL CORRUPT_TEST_LOC
                                DF 05DC PUSHAL ALLMOD
                                DD 05E2 PUSHL #0
                                DD 05E4 PUSHL #SER
                                05E6 ::: TEST 5, SUBTEST 5, ERROR 5
                                00000000'9F  05 FB 05E6 CALLS $$$M, a#DS$ERRHARD
                                00000000'9F  FF7E CF  FA 05ED 1066 $DS_CKLOOP 30$
                                50 D4 05F6 1067 $DS_ABORT CALLG 30$, a#DS$CKLOOP
                                04 05F8 CLRL R0 ; Abort test corrupted test location
                                53 57 D0 05F9 1068 37$: MOVL R7,R3 ; SEND A WARNING TO THE SUPERVISOR
                                55 54 53 CD 05FC 1069 XORL3 R3,R4,R5 ; TERMINATE TEST
                                25 13 0600 1070 BEQL 40$ ; Expect word to be corrected
                                0602 1071 $DS_ERRHARD_S ,#0,CONTROL,- ; Expected = received?
                                0602 1072 PRINT GENERAL,#4,- ; Branch if so
                                0602 1073 #FMT_ECC_2,R3,R4,R5 ; Report the error
                                55 DD 0602 PUSHL R5
                                54 DD 0604 PUSHL R4
                                53 DD 0606 PUSHL R3
                                00000000'8F  DD 0608 PUSHL #FMT_ECC_2
                                04 DD 060E PUSHL #4
                                00000000'EF  DF 0610 PUSHAL PRINT_GENERAL
                                00000000'EF  DF 0616 PUSHAL CONTROL
                                00 DD 061C PUSHL #0
                                06 DD 061E PUSHL #SER
                                0620 ::: TEST 5, SUBTEST 5, ERROR 6
                                00000000'9F  09 FB 0620 CALLS $$$M, a#DS$ERRHARD
                                0627 1074 40$: $DS_CKLOOP 30$ ; Scope loop?

```

```
00000000'9F  FF44 CF  FA 0627          CALLG 30$, a#DS$CKLOOP
      57 57 01 9C 0630 1075          ROTL #1,R7,R7 ; Prepare to do for next bit
      FF35 56 01 1F F1 0634 1076          ACBL #31,#1,R6,30$ ; Do for all 32 bits
      62 E0000000 8F D0 063A 1077          MOVL #^XE0000000,CSR0(R2) ; Clean up the controller
      0641 1078          $DS_ENDSUB
00000000'9F  00000030'EF  FA 0641          T5_S5_X:          CALLG $$$, a#DS$ENDSUB
      064C 1079          $DS_BGNSUB
00000000'9F  0000003C'EF  FA 064C          T5_S6::          CALLG $$$, a#DS$BGNSUB
      0657 1080 ;
      0657 1081 ; Part 6
      0657 1082 ;
      0657 1083 ; Check that the controller correctly reports uncorrectable error.
      0657 1084 ; Also check that memory is not modified and that the uncorrectable error
      0657 1085 ; bit in the BUS ERROR register set.
      0657 1086 ;
00000000'EF  00000662'EF  DE 0657 1087          MOVAL 5$,UNX_UNC_PC ; Save PC for UNEX UNCORRECTABLE ERROR A7
      0662 1088 5$:
04 A2 50 00000000'EF  DE 0662 1089 10$:          MOVAL TEMP,R0 ; Get address of location to write
      04 A2 50 000001FF 8F  CB 0669 1090          BICL3 #^X1FF,R0,CSR1(R2) ; Put page address into CSR1
      04 A2 0A000000 8F  C8 0672 1091          BISL #PM!EDM,CSR1(R2) ; Turn on ECC disable mode
      00000000'EF  03  D0 067A 1092          MOVL #3,TEMP ; Write a location in memory
      50 00000000'EF  D0 0681 1093          MOVL TEMP,R0 ; Read it
      04 A2 02000000 8F  CA 0688 1094          BICL #EDM,CSR1(R2) ; Turn off ECC disable mode
      00000000'EF  D4 0690 1095          CLRL TEMP ; Corrupt two bits of the word
00000000'EF  000006BB'EF  DE 0696 1096          MOVAL 20$,RETURN_PC ; Set up the return PC
00000000'EF  00000101 8F  D0 06A1 1097          MOVL #^X101,EXP_UNC_ERR ; Indicate uncorrectable err expected M2
      04 A2 0C000000 8F  C8 06AC 1098          BISL #PM!DCM,CSR1(R2) ; Turn on page mode/diag check mode
      54 00000000'EF  D0 06B4 1099          MOVL TEMP,R4 ; Read the location
      04 A2 0C000000 8F  CA 06BB 1100 20$:          BICL #PM!DCM,CSR1(R2) ; Turn off page mode/diag check mode
      50 00000000'EF  DE 06C3 1101          MOVAL TEMP,R0 ; Get address of 'TEMP'
      53 50 000001FF 8F  CB 06CA 1102          BICL3 #^X1FF,R0,R3 ; Discard byte within page
      53 80000000 8F  C8 06D2 1103          BISL #UCE,R3 ; Expect uncorrectable error flag set
      54 62 1F0001FF 8F  CB 06D9 1104          BICL3 #^X1F0001FF,CSR0(R2),R4 ; Get CSR0, masking syndrome bits & unused
      55 54 53  CD 06E1 1105          XORL3 R3,R4,R5 ; Expected = received?
      25 13 06E5 1106          BEQL 30$ ; Branch if so
      06E7 1107          $DS_ERRHARD_S ,#0,CONTROL,-
      06E7 1108          PRINT_GENERAL,#4,- ; Report the error
      06E7 1109          #FMT_ECC_5,R3,R4,R5
      55 DD 06E7          PUSHL R5
      54 DD 06E9          PUSHL R4
      53 DD 06EB          PUSHL R3
      00000000'8F  DD 06ED          PUSHL #FMT_ECC_5
      04 DD 06F3          PUSHL #4
      00000000'EF  DF 06F5          PUSHAL PRINT_GENERAL
      00000000'EF  DF 06FB          PUSHAL CONTROL
      00 DD 0701          PUSHL #0
      01 DD 0703          PUSHL #SER
      0705          ;;: TEST 5, SUBTEST 6, ERROR 1
00000000'9F  09  FB 0705          CALLS $$$M, a#DS$ERRHARD
      62 E0000000 8F  D0 070C 1110 30$:          MOVL #^XE0000000,CSR0(R2) ; Clean up the controller
      0713 1111          $DS_CKLOOP 10$ ; Scope loop?
      00000000'9F  FF4B CF  FA 0713          CALLG 10$, a#DS$CKLOOP
00000000'EF  00000739'EF  9E 071C 1112          MOVAB 35$,RETURN_PC
00000000'EF  00000101 8F  D0 0727 1113 32$:          MOVL #^X101,EXP_UNC_ERR
      54 00000000'EF  D0 0732 1114          MOVL TEMP,R4 ; Read the memory location again
      A2
```

```
29 00000000'EF 00 E4 0739 1115 35$: BBSC #0,EXP_UNC_ERR,37$ ; Branch if no double bit
                                0741 1116 $DS_ERRHARD_S ,#0,ALLMOD,-
                                0741 1117 CORRUPT_TEST_LOC,-
                                0741 1118 #TEMP
                                DD 0741 PUSHL #TEMP
                                DF 0747 PUSHAL CORRUPT_TEST_LOC
                                DF 074D PUSHAL ALLMOD
                                DD 0753 PUSHL #0
                                DD 0755 PUSHL #SER
                                0757 ::: TEST 5, SUBTEST 6, ERROR 2
00000000'9F 05 FB 0757 CALLS $$$M, a#DS$ERRHARD
00000000'9F FF00 CF FA 075E 1119 $DS_CKLOOP 10$
                                0767 1120 $DS_ABORT TEST ; Abort test corrupted test location
                                DD 0767 CLRL R0 ; SEND A WARNING TO THE SUPERVISOR
                                DD 0769 RET ; TERMINATE TEST
                                DD 076A 1121 37$: CLRL R3 ; Get expected data A2
55 54 53 CD 076C 1122 XORL3 R3,R4,R5 ; Check that memory was not modified A2
                                DD 0770 1123 BEQL 40$ ; Branch if ok A2
                                DD 0772 1124 $DS_ERRHARD_S ,#0,CONTROL,-
                                DD 0772 1125 PRINT_GENERAL,#4,- ; Report the error
                                DD 0772 1126 #FMT_ECC_4,R3,R4,R5 ; A2
                                DD 0772 PUSHL R5
                                DD 0774 PUSHL R4
                                DD 0776 PUSHL R3
                                DD 0778 PUSHL #FMT_ECC_4
                                DD 077E PUSHL #4
                                DF 0780 PUSHAL PRINT_GENERAL
                                DF 0786 PUSHAL CONTROL
                                DD 078C PUSHL #0
                                DD 078E PUSHL #SER
                                0790 ::: TEST 5, SUBTEST 6, ERROR 3
00000000'9F 09 FB 0790 CALLS $$$M, a#DS$ERRHARD
00000000'9F FEC7 CF FA 0797 1127 40$: $DS_CKLOOP 10$ ; Scope loop? A2
56 00000000'EF 04 CD 07A0 1128 XORL3 #4,BUS_ERR_REG,R6 ; Get xor of expected and received A2
                                DD 07A8 1129 BEQL 50$ ; Branch if Expected = Received A2
                                DD 07AA 1130 $DS_ERRHARD_S ,#0,CPU,-
                                DD 07AA 1131 PRINT_GENERAL,#4,- ; No. Report the error
                                DD 07AA 1132 #FMT_CPU_2,#4,BUS_ERR_REG,R6 ; A2
                                DD 07AA PUSHL R6
                                DD 07AC PUSHL BUS_ERR_REG
                                DD 07B2 PUSHL #4
                                DD 07B4 PUSHL #FMT_CPU_2
                                DD 07BA PUSHL #4
                                DF 07BC PUSHAL PRINT_GENERAL
                                DF 07C2 PUSHAL CPU
                                DD 07C8 PUSHL #0
                                DD 07CA PUSHL #SER
                                07CC ::: TEST 5, SUBTEST 6, ERROR 4
00000000'9F 09 FB 07CC CALLS $$$M, a#DS$ERRHARD
00000000'9F FE8B CF FA 07D3 1133 50$: $DS_CKLOOP 10$ ; Scope loop? A2
                                DD 07DC 1134 $DS_ENDSUB CALLG 10$, a#DS$CKLOOP
                                DD 07DC T5_S6_X: CALLG $$$, a#DS$ENDSUB
00000000'9F 0000003C'EF FA 07DC 1135 $DS_BGNSUB
```

```
00000000'9F 00000048'EF FA 07E7 T5_S7:: CALLG $$$, a#DS$BGNSUB
07E7
07F2 1136 ;
07F2 1137 ; Part 7
07F2 1138 ;
07F2 1139 ; Check that the controller reports uncorrectable error, info lost properly.
07F2 1140 ;
00000000'EF 000007FD'EF DE 07F2 1141 MOVAL 5$,UNX_UNC_PC ; Save PC for UNEX UNCORRECTABLE ERROR A7
07FD 1142 5$:
04 A2 62 E0000000 8F D0 07FD 1143 10$: MOVL #^XE0000000,CSR0(R2) ; Cleanup the controller
04 A2 50 00000000'EF DE 0804 1144 MOVAL TEMP,R0 ; Get address of location to write
04 A2 50 000001FF 8F CB 080B 1145 BICL3 #^X1FF,R0,CSR1(R2) ; Put page address into CSR1
04 A2 04 A2 0A000000 8F C8 0814 1146 BISL #PM!EDM,CSR1(R2) ; Turn on ECC disable mode
00000000'EF 03 D0 081C 1147 MOVL #3,TEMP ; Write a location in memory
04 A2 50 00000000'EF D0 0823 1148 MOVL TEMP,R0 ; Read it
04 A2 04 A2 02000000 8F CA 082A 1149 BICL #EDM,CSR1(R2) ; Turn off ECC disable mode
00000000'EF D4 0832 1150 CLRL TEMP ; Corrupt two bits of the word
00000000'EF 00000859'EF DE 0838 1151 MOVAL 20$,RETURN_PC ; Set up the return PC
00000000'EF 01 D0 0843 1152 MOVL #1,EXP_UNC_ERR ; Indicate uncorrectable err expected
04 A2 04 A2 0C000000 8F C8 084A 1153 BISL #PM!DCM,CSR1(R2) ; Turn on page mode/diag check mode
54 00000000'EF D0 0852 1154 MOVL TEMP,R4 ; Read the location
04 A2 04 A2 0C000000 8F CA 0859 1155 20$: BICL #PM!DCM,CSR1(R2) ; Turn off page mode/diag check mode
04 A2 04 A2 0A000000 8F C8 0861 1156 BISL #PM!EDM,CSR1(R2) ; Turn on ECC disable mode
04 A2 50 00000000'EF DE 0869 1157 MOVAL TEMP1,R0 ; Get an address in another page
04 A2 50 000001FF 8F CB 0870 1158 BICL3 #^X1FF,R0,CSR1(R2) ; And put page address into pm address
00000000'EF 03 D0 0879 1159 MOVL #3,TEMP1 ; Write a location in another page
04 A2 50 00000000'EF D0 0880 1160 MOVL TEMP1,R0 ; Read it
04 A2 04 A2 02000000 8F CA 0887 1161 BICL #EDM,CSR1(R2) ; Turn off ECC disable mode
00000000'EF D4 088F 1162 CLRL TEMP1 ; Simulate another double-bit error
00000000'EF 000008B6'EF DE 0895 1163 MOVAL 30$,RETURN_PC ; Set up return PC
00000000'EF 01 D0 08A0 1164 MOVL #1,EXP_UNC_ERR ; Indicate uncorrectable err expected
04 A2 04 A2 0C000000 8F C8 08A7 1165 BISL #PM!DCM,CSR1(R2) ; Turn on page mode/diag check mode
54 00000000'EF D0 08AF 1166 MOVL TEMP1,R4 ; Read the corrupted word
04 A2 04 A2 0C000000 8F CA 08B6 1167 30$: BICL #PM!DCM,CSR1(R2) ; Turn off page mode/diag check mode
50 00000000'EF DE 08BE 1168 MOVAL TEMP,R0 ; Get previously corrupted address
53 50 000001FF 8F CB 08C5 1169 CLRL R3 ; Clear expected results register
53 C0000000 8F C8 08C7 1170 BICL3 #^X1FF,R0,R3 ; Mask all put page address
54 62 1F0001FF 8F CB 08CF 1171 BISL #UCE!UCEIL,R3 ; Expect UCE and info lost flags
55 54 53 CD 08DE 1173 XORL3 R3,R4,R5 ; Get CSR0, masking syndrome bits M2
25 13 08E2 1174 BEQL 40$ ; Expected = received?
08E4 1175 $DS_ERRHARD_S ,#0,CONTROL,- ; Report the error
08E4 1176 PRINT GENERAL,#4,-
08E4 1177 #FMT_ECC_7,R3,R4,R5
55 DD 08E4 PUSHL R5
54 DD 08E6 PUSHL R4
53 DD 08E8 PUSHL R3
00000000'8F DD 08EA PUSHL #FMT_ECC_7
04 DD 08F0 PUSHL #4
00000000'EF DF 08F2 PUSHAL PRINT_GENERAL
00000000'EF DF 08F8 PUSHAL CONTROL
00 DD 08FE PUSHL #0
01 DD 0900 PUSHL #SER
0902 ::: TEST 5, SUBTEST 7, ERROR 1
00000000'9F 09 FB 0902 CALLS $$$M, a#DS$ERRHARD
0909 1178 40$: $DS_CKLOOP 10$ ; Scope loop?
00000000'9F FEFO CF FA 0909 CALLG 10$, a#DS$CKLOOP
```

```

0912 1179 ; CLRL CSR1(R2) ; Clear CSR no. 1 D2
0912 1180 $DS_ENDSUB
0912 T5_S7_X:
00000000'9F 00000048'EF FA 0912 CALLG $$$, a#DS$ENDSUB
091D 1181 $DS_BGNSUB
00000000'9F 00000054'EF FA 091D CALLG $$$, a#DS$BGNSUB
091D T5_S8::
0928 1182 ; A2
0928 1183 ; Part 8 A2
0928 1184 ; A2
0928 1185 ; Check that the controller will interrupt on single bit error. A2
0928 1186 ; A2
00000000'EF 0000093E'EF DE 0928 1187 MOVAL 5$,UNX_UNC_PC ; Save PC for UNEX UNCORRECTABLE ERROR A7
00000000'EF 000009A0'EF DE 0933 1188 MOVAL 15$,RETURN_PC
093E 1189 5$:
04 A2 62 E0000000 8F D0 093E 1190 10$: MOVL #^XE0000000,CSR0(R2) ; Cleanup the controller
50 00000000'EF DE 0945 1191 MOVAL TEMP,R0 ; Get address of location to write A2
04 A2 50 000001FF 8F CB 094C 1192 BICL3 #^X1FF,R0,CSR1(R2) ; And put address into pm address A2
04 A2 0A000000 8F C8 0955 1193 BISL #PM!EDM,CSR1(R2) ; Turn on ECC disable mode A2
00000000'EF 01 D0 095D 1194 MOVL #1,TEMP ; Write the test location A2
00000000'EF D5 0964 1195 TSTL TEMP ; Read it A2
04 A2 02000000 8F CA 096A 1196 BICL #EDM,CSR1(R2) ; Turn off ECC disable mode A2
00000000'EF D4 0972 1197 CLRL TEMP ; Simulate a single bit failure A2
00000000'EF 01 D0 0978 1198 MOVL #1,EXP_COR_ERR ; Indicate correctable error expected A2 M
00000000'EF 00000101 8F D0 097F 1199 MOVL #^X101,EXP_UNC_ERR
04 A2 10000000 8F C8 098A 1200 BISL #ERCE,CSR1(R2) ; Enable single bit error interrupt A2 M
04 A2 0C000000 8F C8 0992 1201 BISL2 #PM!DCM,CSR1(R2) ; Turn on page mode/diag check mode A2
00000000'EF D5 099A 1202 TSTL TEMP ; Read the corrupted word A2
29 00000000'EF 00 E4 09A0 1203 15$: BBSC #0,EXP_UNC_ERR,17$ ; Branch if no double bit
09A8 1204 $DS_ERRHARD_S ,#0,ALLMOD,-
09A8 1205 CORRUPT_TEST_LOC,-
09A8 1206 #TEMP
00000000'8F DD 09A8 PUSHL #TEMP
00000000'EF DF 09AE PUSHAL CORRUPT_TEST_LOC
00000000'EF DF 09B4 PUSHAL ALLMOD
00 DD 09BA PUSHL #0
01 DD 09BC PUSHL #SER
09BE ;;; TEST 5, SUBTEST 8, ERROR 1
00000000'9F 05 FB 09BE CALLS $$$M, a#DS$ERRHARD
09C5 1207 $DS_CKLOOP 10$
00000000'9F FF75 CF FA 09C5 CALLG 10$, a#DS$CKLOOP
09CE 1208 $DS_ABORT TEST ; Abort test corrupted test location
50 D4 09CE CLRL R0 ; SEND A WARNING TO THE SUPERVISOR
04 09D0 RET ; TERMINATE TEST
00000000'EF 95 09D1 1209 17$: TSTB EXP_COR_ERR ; Check if interrupt occurred A2
1F 13 09D7 1210 BEQL 20$ ; Branch if it did A2
09D9 1211 $DS_ERRHARD_S ,#0,CONTROL,-
09D9 1212 PRINT_GENERAL,#1,- ; Report the error
09D9 1213 #FMT_ECC_14 ;
00000000'8F DD 09D9 PUSHL #FMT_ECC_14
01 DD 09DF PUSHL #1
00000000'EF DF 09E1 PUSHAL PRINT_GENERAL
00000000'EF DF 09E7 PUSHAL CONTRÖL
00 DD 09ED PUSHL #0
02 DD 09EF PUSHL #SER
00000000'9F 06 FB 09F1 ;;; TEST 5, SUBTEST 8, ERROR 2
09F1 CALLS $$$M, a#DS$ERRHARD

```

```
00000000'9F  FF42 CF  FA 09F8 1214 20$:  $DS_CKLOOP 10$ ; Scope loop? A2
                                09F8 CALLG 10$, a#DS$CKLOOP
                                0A01 1215 $DS_ENDSUB ; A2
00000000'9F  00000054'EF  FA 0A01 T5_S8_X:
                                0A01 CALLG $$$, a#DS$ENDSUB ; A2
                                0A0C 1216 $DS_BGNSUB ; A2
00000000'9F  00000060'EF  FA 0A0C T5_S9::
                                0A0C CALLG $$$, a#DS$BGNSUB
                                0A17 1217 ; A2
                                0A17 1218 ; Part 9 A2
                                0A17 1219 ; A2
                                0A17 1220 ; Check that inhibiting single bit error reporting also inhibits the setting A2
                                0A17 1221 ; of bits <29> and <23:9> in CSRO. A2
                                0A17 1222 ; A2
00000000'EF  00000A2D'EF  DE 0A17 1223 MOVAL 5$,UNX_UNC_PC ; Save PC for UNEX UNCORRECTABLE ERROR A7
00000000'EF  00000A88'EF  DE 0A22 1224 MOVAL 15$,RETURN_PC
                                0A2D 1225 5$:
54 62 E0000000 8F  D0 0A2D 1226 10$: MOVL #^XE0000000,CSRO(R2) ; Cleanup the controller
04 62 FF0001FF 8F  CB 0A34 1227 BICL3 #^C<^X00FFFE00>,CSRO(R2),R4 ; Read and save current error A2
                                ; address bits in CSRO A2
                                0A3C 1228 ; Get address of test location A2
04 A2 50 00000000'EF  DE 0A3C 1229 MOVAL TEMP,R0 ; Put page address into pm address A2
04 A2 50 000001FF 8F  CB 0A43 1230 BICL3 #^X1FF,R0,CSR1(R2) ; Turn on ECC disable mode A2
04 A2 0A000000 8F  C8 0A4C 1231 BISL #PM!EDM,CSR1(R2) ; Write test pattern into memory A2
00000000'EF  01 D0 0A54 1232 MOVL #1,TEMP ; Read it A2
04 A2 00000000'EF  D5 0A5B 1233 TSTL TEMP ; Turn off ECC disable mode A2
04 A2 02000000 8F  CA 0A61 1234 BICL #EDM,CSR1(R2) ; Change pattern in memory A2
00000000'EF  00000000'EF  D4 0A69 1235 CLRL TEMP ; Turn on diagnostic check mode A2
04 A2 00000101 8F  D0 0A6F 1236 MOVL #^X101,EXP_UNC_ERR ; Read the test location A2
04 A2 0C000000 8F  C8 0A7A 1237 BISL #PM!DCM,CSR1(R2) ; Turn off check mode A2
04 A2 00000000'EF  D5 0A82 1238 TSTL TEMP ; Branch if no double bit A2
29 00000000'EF  00 E4 0A88 1239 15$: BICL #PM!DCM,CSR1(R2)
                                0A90 1240 BBSC #0,EXP_UNC_ERR,17$
                                0A98 1241 $DS_ERRHARD_S ,#0,ALLMOD,-
                                0A98 1242 CORRUPT_TEST_LOC,-
                                0A98 1243 #TEMP
                                DD 0A98 PUSHL #TEMP
                                DF 0A9E PUSHAL CORRUPT_TEST_LOC
                                DF 0AA4 PUSHAL ALLMOD
                                DD 0AAA PUSHL #0
                                DD 0AAC PUSHL #SER
                                0AAE ;;; TEST 5, SUBTEST 9, ERROR 1
00000000'9F  05 FB 0AAE CALLS $$$M, a#DS$ERRHARD
00000000'9F  FF74 CF  FA 0AB5 1244 $DS_CKLOOP 10$
                                0AB5 1245 $DS_ABORT CALLG 10$, a#DS$CKLOOP
                                0ABE 1245 $DS_ABORT TEST ; Abort test corrupted test location
                                50 D4 0ABE CLRL R0 ; SEND A WARNING TO THE SUPERVISOR
                                04 0AC0 RET ; TERMINATE TEST
53 62 DF0001FF 8F  CB 0AC1 1246 17$: BICL3 #^C<CE!^X00FFFE00>,CSRO(R2),R3 ; Read CSRO bits <29>,<23:9> A2
55 54 53 CD 0AC9 1247 XORL3 R3,R4,R5 ; Are the bits correct? A2
25 13 0ACD 1248 BEQL 20$ ; Branch if yes A2
                                0ACF 1249 $DS_ERRHARD_S ,#0,CONTROL,-
                                0ACF 1250 PRINT_GENERAL,#4,- ; Report the error
                                0ACF 1251 #FMT_ECC_13,R4,R3,R5 ; A2
                                DD 0ACF PUSHL R5
                                DD 0AD1 PUSHL R3
                                DD 0AD3 PUSHL R4
00000000'8F  DD 0AD5 PUSHL #FMT_ECC_13
```

```
04 DD 0ADB PUSHL #4
00000000'EF DF 0ADD PUSHAL PRINT_GENERAL
00000000'EF DF 0AE3 PUSHAL CONTRÖL
00 DD 0AE9 PUSHL #0
02 DD 0AEB PUSHL #SER
0AED :;; TEST 5, SUBTEST 9, ERROR 2
00000000'9F 09 FB 0AED CALLS $$$M, a#DS$ERRHARD
00000000'9F FF35 CF FA 0AF4 1252 20$: $DS_CKLOOP 10$ ; Scope loop? A2
0AF4 1253 $DS_ENDSUB ; A2
0AFD T5_S9_X:
0AFD CALLG $$$, a#DS$ENDSUB
0B08 1254 $DS_BGNSUB ; A2
0B08 T5_S10::
0B08 CALLG $$$, a#DS$BGNSUB
0B13 1255 ;
0B13 1256 ; Part 10
0B13 1257 ;
0B13 1258 ; Check all combinations of two bits stuck at one or zero (in a field of zero's
0B13 1259 ; and one's) cause a double bit error.
0B13 1260 ;
00000000'EF 00000B1E'EF DE 0B13 1261 MOVAL 2$,UNX_UNC_PC ; Save PC for UNEX UNCORRECTABLE ERROR A7
0B1E 1262 2$:
04 A2 50 00000000'EF DE 0B1E 1263 MOVAL TEMP,R0 ; Get address fo test location A2
04 A2 50 000001FF 8F CB 0B25 1264 BICL3 #^X1FF,R0,CSR1(R2) ; Put page address in pm address A2
56 01 D0 0B2E 1265 MOVL #1,R6 ; Initialize one pattern A2
59 D4 0B31 1266 CLRL R9 ; Setup a counter A2
57 01 D0 0B33 1267 5$: MOVL #1,R7 ; Initialize the second pattern A2
58 D4 0B36 1268 CLRL R8 ; Setup a counter A2
57 56 D1 0B38 1269 10$: CMLP R6,R7 ; Does pattern1 = pattern2? A2
03 12 0B3B 1270 BNEQ 15$ ; Branch if no otherwise A2
0096 31 0B3D 1271 BRW 40$ ; Skip this itteration A2
04 A2 0A000000 8F C8 0B40 1272 15$: BISL #PM!EDM,CSR1(R2) ; Set ECC disable mode A2
00000000'EF 57 56 C9 0B48 1273 BISL3 R6,R7,TEMP ; Put pattern in test location A2
00000000'EF D5 0B50 1274 TSTL TEMP ; Read it A2
04 A2 02000000 8F CA 0B56 1275 BICL #EDM,CSR1(R2) ; Enable ECC A2
00000000'EF D4 0B5E 1276 CLRL TEMP ; Change pattern to force error A2
00000000'EF 01 D0 0B64 1277 MOVL #1,EXP_UNC_ERR ; Set expected error flag A2
00000000'EF 00000B84'EF DE 0B6B 1278 MOVAL 20$,RETURN_PC ; Set return pc for machine check A2
04 A2 0C000000 8F C8 0B76 1279 BISL #PM!DCM,CSR1(R2) ; Set diagnostic check mode A2
00000000'EF D5 0B7E 1280 TSTL TEMP ; Force the double bit error A2
04 A2 0C000000 8F CA 0B84 1281 20$: BICL #PM!DCM,CSR1(R2) ; Clear diagnostic check mode A2
54 62 7FFFFFFF 8F CB 0B8C 1282 BICL3 #^CUCE,CSR0(R2),R4 ; Read UCE bit in CSR0 A2
53 80000000 8F D0 0B94 1283 MOVL #UCE,R3 ; Get expected data A2
55 54 53 CD 0B9B 1284 XORL3 R3,R4,R5 ; Expected = Received? A2
25 13 0B9F 1285 BEQL 30$ ; Branch if yes A2
0BA1 1286 $DS_ERRHARD_S ,#0,CONTROL,-
0BA1 1287 PRINT_GENERAL,#4,- ; Report the error
0BA1 1288 #FMT_ECC_5,R3,R4,R5 ; A2
55 DD 0BA1 PUSHL R5
54 DD 0BA3 PUSHL R4
53 DD 0BA5 PUSHL R3
00000000'8F DD 0BA7 PUSHL #FMT_ECC_5
04 DD 0BAD PUSHL #4
00000000'EF DF 0BAF PUSHAL PRINT_GENERAL
00000000'EF DF 0BB5 PUSHAL CONTRÖL
00 DD 0BBB PUSHL #0
```

```

01 DD 0BBD          PUSHL  #SER
0BBF          CALLS  $$$M, a#DS$ERRHARD
          ::: TEST 5, SUBTEST 10, ERROR 1
00000000'9F 09 FB 0BBF          MOVL  #^XE0000000,CSR0(R2) ; Cleanup the controller
62 E0000000 8F D0 0BC6 1289 30$: $DS_CKLOOP 15$ ; Scope loop?
          OBCD 1290
00000000'9F FF6F CF FA 0BCD          CALLG 15$, a#DS$CKLOOP
57 57 01 9C 0BD6 1291 40$: ROTL #1,R7,R7 ; Shift pattern2
FF58 58 01 1F F1 0BDA 1292 ACBL #31,#1,R8,10$ ; Do for 31 patterns
56 56 01 9C 0BE0 1293 ROTL #1,R6,R6 ; Shift pattern1
FF49 59 01 1F F1 0BE4 1294 ACBL #31,#1,R9,5$ ; Do for 31 patterns
          0BEA 1295 ;
          0BEA 1296 ; Check two zero's in a field of one's
          0BEA 1297 ;
04 A2 50 00000000'EF DE 0BEA 1298 MOVAL TEMP,R0 ; Get address fo test location
50 000001FF 8F CB 0BF1 1299 BICL3 #^X1FF,R0,CSR1(R2) ; Put page address in pm address
56 FFFFFFFF 8F D0 0BFA 1300 MOVL #-2,R6 ; Initialize one pattern
          59 D4 0C01 1301 CLRL R9 ; Setup a counter
57 FFFFFFFF 8F D0 0C03 1302 50$: MOVL #-2,R7 ; Initialize the second pattern
          58 D4 0C0A 1303 CLRL R8 ; Setup a counter
          57 56 D1 0C0C 1304 100$: CMLP R6,R7 ; Does pattern1 = pattern2?
          03 12 0C0F 1305 BNEQ 150$ ; Branch if no otherwise
          009E 31 0C11 1306 BRW 400$ ; Skip this itteration
04 A2 0A000000 8F C8 0C14 1307 150$: BISL #PM!EDM,CSR1(R2) ; Set ECC disable mode
          5A 56 D2 0C1C 1308 MCOML R6,R10 ; Get compliment of first pattern
00000000'EF 57 5A CB 0C1F 1309 BICL3 R10,R7,TEMP ; Put pattern in test location
          00000000'EF D5 0C27 1310 TSTL TEMP ; Read it
04 A2 02000000 8F CA 0C2D 1311 BICL #EDM,CSR1(R2) ; Enable ECC
00000000'EF FFFFFFFF 8F D0 0C35 1312 MOVL #-1,TEMP ; Change pattern to force error
00000000'EF 01 D0 0C40 1313 MOVL #1,EXP_UNC_ERR ; Set expected error flag
00000000'EF 00000C60'EF DE 0C47 1314 MOVAL 200$,RETURN_PC ; Set return pc for machine check
04 A2 0C000000 8F C8 0C52 1315 BISL #PM!DCM,CSR1(R2) ; Set diagnostic check mode
          00000000'EF D5 0C5A 1316 TSTL TEMP ; Force the double bit error
04 A2 0C000000 8F CA 0C60 1317 200$: BICL #PM!DCM,CSR1(R2) ; Clear diagnostic check mode
54 62 7FFFFFFF 8F CB 0C68 1318 BICL3 #^CUCE,CSR0(R2),R4 ; Read UCE bit in CSR0
53 80000000 8F D0 0C70 1319 MOVL #UCE,R3 ; Get expected data
          55 54 53 CD 0C77 1320 XORL3 R3,R4,R5 ; Expected = Received?
          25 13 0C7B 1321 BEQL 300$ ; Branch if yes
          0C7D 1322 $DS_ERRHARD_S ,#0,CONTROL,-
          0C7D 1323 PRINT_GENERAL,#4,- ; Report the error
          0C7D 1324 #FMT_ECC_5,R3,R4,R5 ; A2
          55 DD 0C7D          PUSHL R5
          54 DD 0C7F          PUSHL R4
          53 DD 0C81          PUSHL R3
          00000000'8F DD 0C83          PUSHL #FMT_ECC_5
          04 DD 0C89          PUSHL #4
          00000000'EF DF 0C8B          PUSHAL PRINT_GENERAL
          00000000'EF DF 0C91          PUSHAL CONTROL
          00 DD 0C97          PUSHL #0
          02 DD 0C99          PUSHL #SER
          0C9B          ::: TEST 5, SUBTEST 10, ERROR 2
00000000'9F 09 FB 0C9B          CALLS  $$$M, a#DS$ERRHARD
62 E0000000 8F D0 0CA2 1325 300$: MOVL #^XE0000000,CSR0(R2) ; Cleanup the controller
          0CA9 1326 $DS_CKLOOP 150$ ; Scope loop?
00000000'9F FF67 CF FA 0CA9          CALLG 150$, a#DS$CKLOOP
57 57 01 9C 0CB2 1327 400$: ROTL #1,R7,R7 ; Shift pattern2
FF50 58 01 1F F1 0CB6 1328 ACBL #31,#1,R8,100$ ; Do for 31 patterns
56 56 01 9C 0CBC 1329 ROTL #1,R6,R6 ; Shift pattern1

```

```
FF3D 59 01 1F F1 OCC0 1330 ACBL #31,#1,R9,50$ ; Do for 31 patterns A2
OCC6 1331 $DS_ENDSUB ; A2
OCC6 T5_S10_X:
00000000'9F 0000006C'EF FA OCC6 CALLG $$$, a#DS$ENDSUB ; A2
OCD1 1332 $DS_BGNSUB ; A2
OCD1 T5_S11::
00000000'9F 00000078'EF FA OCD1 CALLG $$$, a#DS$BGNSUB
OCDC 1333 ;
OCDC 1334 ; Part 11
OCDC 1335 ;
OCDC 1336 ; Check that failures in the check bits cause single bit errors and that the
OCDC 1337 ; syndrome is correct.
OCDC 1338 ;
00000000'EF 00000CF2'EF DE OCD1 1339 MOVAL 5$,UNX_UNC_PC ; Save PC for UNEX UNCORRECTABLE ERROR A7
00000000'EF 00000D43'EF DE OCE7 1340 MOVAL 15$,RETURN_PC
OCF2 1341 5$:
50 00000000'EF DE OCF2 1342 MOVAL TEMP,R0 ; Get address of test location A2
56 01 DO OCF9 1343 MOVL #1,R6 ; Initialize a counter and check bit A2
OCFC 1344 ; pattern. A2
04 A2 50 000001FF 8F CB OCFC 1345 BICL3 #^X1FF,R0,CSR1(R2) ; Put page address into pm address A2
62 E0000000 8F DO OD05 1346 10$: MOVL #^XE0000000,CSR0(R2) ; Cleanup the controller
00000000'EF 01101010 8F DO OD0C 1347 MOVL #^X01101010,TEMP ; Load data pattern into test location A2
OD17 1348 ; whose check bits are all zero A2
04 A2 0000007F 8F CA OD17 1349 BICL #^X7F,CSR1(R2) ; Clear diagnostic check bits A2
04 A2 56 C8 OD1F 1350 BISL R6,CSR1(R2) ; Load test pattern into check bits A2
00000000'EF 01 DO OD23 1351 MOVL #1,EXP_COR_ERR ; Setup to catch interrupt A2 M
00000000'EF 00000101 8F DO OD2A 1352 MOVL #^X101,EXP_UNC_ERR
04 A2 1C000000 8F C8 OD35 1353 BISL #ERCE!PM!DCM,CSR1(R2) ; Set diagnostic check mode A2 M
00000000'EF DS OD3D 1354 TSTL TEMP ; Force single bit error A2
04 A2 0C000000 8F CA OD43 1355 15$: BICL #PM!DCM,CSR1(R2) ; Clear diagnostic check mode A2
28 00000000'EF 00 E4 OD4B 1356 BBSC #0,EXP_UNC_ERR,17$ ; Branch if no double bit
OD53 1357 $DS_ERRHARD_S ,#0,ALLMOD,-
OD53 1358 CORRUPT_TEST_LOC,-
OD53 1359 #TEMP
PUSHL #TEMP
PUSHAL CORRUPT_TEST_LOC
PUSHAL ALLMOD
PUSHL #0
PUSHL #SER
PUSHL #ERROR 1
00000000'8F DD OD53
00000000'EF DF OD59
00000000'EF DF OD5F
00 DD OD65
01 DD OD67
OD69
00000000'9F 05 FB OD69
OD70 1360 $DS_CKLOOP
00000000'9F 92 AF FA OD70 CALLS $$$M, a#DS$ERRHARD
OD78 1361 $DS_ABORT TEST ; Abort test corrupted test location
50 D4 OD78 CLRL R0 ; SEND A WARNING TO THE SUPERVISOR
04 OD7A RET ; TERMINATE TEST
54 62 DFFFFFF80 8F CB OD7B 1362 17$: BICL3 #^C<CE!^X7F>,CSR0(R2),R4 ; Read syndrome and CE bit A2
53 20000000 8F DO OD83 1363 MOVL #CE,R3 ; Create expected data A2
53 56 C8 OD8A 1364 BISL R6,R3 ; ... A2
55 54 53 CD OD8D 1365 XORL3 R3,R4,R5 ; Expected = Received? A2
25 13 BEQL 20$ ; Branch if yes A2
OD93 1367 $DS_ERRHARD_S ,#0,CONTROL,-
OD93 1368 PRINT_GENERAL,#4,- ; Report the error
OD93 1369 #FMT_ECC_2,R3,R4,R5 ; A2
55 DD OD93 PUSHL R5
54 DD OD95 PUSHL R4
53 DD OD97 PUSHL R3
```

```
00000000'8F DD 0D99          PUSHL #FMT_ECC_2
00000000'04 DD 0D9F          PUSHL #4
00000000'EF DF 0DA1          PUSHAL PRINT_GENERAL
00000000'EF DF 0DA7          PUSHAL CONTROL
00 DD 0DAD          PUSHL #0
02 DD 0DAF          PUSHL #SER
00000000'9F 09 FB 0DB1      ::: TEST 5, SUBTEST 11, ERROR 2
00000000'9F FF49 CF FA 0DB1 1370 20$: $DS_CKLOOP CALLS $$$M, @#DS$ERRHARD
56 56 01 9C 0DB8 1370 20$: $DS_CKLOOP 10$ ; Scope loop? A2
56 95 0DC1 1371 ROTL #1,R6,R6 ; Select the next pattern A2
03 19 0DC5 1372 TSTB R6 ; A2
FF39 31 0DC7 1373 BLSS 25$ ; Done when bit 7 sets A2
0DC9 1374 BRW 10$ ; ... A2
0DC 1375
0DC 1376
0DC 1377 ; Float a zero thru the check bits
0DC 1378
00000000'EF 00000DD7'EF DE 0DC 1379 25$: MOVAL 27$,UNX_UNC_PC
50 00000000'EF DE 0DD7 1380 27$: MOVAL TEMP,R0 ; Get address of test location A2
56 FFFFFFFE 8F D0 0DDE 1381 MOVL #-2,R6 ; Initialize a counter and check bit A2
04 A2 50 000001FF 8F CB 0DE5 1382 ; pattern. A2
00000000'EF 00000E37'EF 9E 0DEE 1384 MOVAB #AX1FF,R0,CSR1(R2) ; Put page address into pm address A2
62 E0000000 8F D0 0DF9 1385 30$: MOVL #AXE0000000,CSR0(R2) ; Cleanup the controller
00000000'EF 81101010 8F D0 0E00 1386 MOVL #AX81101010,TEMP ; Load data pattern into test location A2
04 A2 0000007F 8F CA 0E0B 1387 ; whose check bits are all one A2
04 A2 04 A2 56 88 0E13 1388 BICL #AX7F,CSR1(R2) ; Clear diagnostic check bits A2
00000000'EF 00000000'EF 01 D0 0E17 1390 BISB R6,CSR1(R2) ; Load test pattern into check bits A2
00000000'EF 00000101 8F D0 0E1E 1391 MOVL #1,EXP_COR_ERR ; Setup to catch interrupt A2 M
04 A2 1C000000 8F C8 0E29 1392 MOVL #AX101,EXP_UNC_ERR
00000000'EF 00000000'EF D5 0E31 1393 BISL #ERCE!PM!DCM,CSR1(R2) ; Set diagnostic check mode A2 M
04 A2 0C000000 8F CA 0E37 1394 35$: TSTL TEMP ; Force single bit error A2
29 00000000'EF 00  E4 0E3F 1395 BBSC #0,EXP_UNC_ERR,37$ ; Clear diagnostic check mode A2
0E47 1396 $DS_ERRHARD_S ,#0,ALLMOD,- ; Branch if no double bit
0E47 1397 CORRUPT_TEST_LOC,-
0E47 1398 #TEMP
00000000'8F DD 0E47          PUSHL #TEMP
00000000'EF DF 0E4D          PUSHAL CORRUPT_TEST_LOC
00000000'EF DF 0E53          PUSHAL ALLMOD
00 DD 0E59          PUSHL #0
03 DD 0E5B          PUSHL #SER
00000000'9F 05 FB 0E5D      ::: TEST 5, SUBTEST 11, ERROR 3
00000000'9F FE9D CF FA 0E5D 1399 $DS_CKLOOP CALLS $$$M, @#DS$ERRHARD
50 D4 0E6D 1400 $DS_ABORT TEST ; Abort test corrupted test location
04 0E6F CLRL R0 ; SEND A WARNING TO THE SUPERVISOR
54 62 DFFFFFF80 8F CB 0E70 1401 37$: BICL3 #AC<CE!^X7F>,CSR0(R2),R4 ; Read syndrome and CE bit A2
53 53 56 D2 0E78 1402 MCOML R6,R3 ; Create expected data A2
55 54 53 C8 0E7B 1403 BISL #CE,R3 ; ... A2
25 13 0E82 1404 XORL3 R3,R4,R5 ; Expected = Received? A2
0E86 1405 BEQL 40$ ; Branch if yes A2
0E88 1406 $DS_ERRHARD_S ,#0,CONTROL,-
0E88 1407 PRINT_GENERAL,#4,- ; Report the error
```

```

00000000'8F DD 0E88 1408 #FMT_ECC_2,R3,R4,R5 ; A2
55 DD 0E88 PUSHL R5
54 DD 0E8A PUSHL R4
53 DD 0E8C PUSHL R3
00000000'8F DD 0E8E PUSHL #FMT_ECC_2
04 DD 0E94 PUSHL #4
00000000'EF DF 0E96 PUSHAL PRINT_GENERAL
00000000'EF DF 0E9C PUSHAL CONTROL
00 DD 0EA2 PUSHL #0
04 DD 0EA4 PUSHL #SER
00000000'9F 09 FB 0EA6 ::: TEST 5, SUBTEST 11, ERROR 4
0EA6 CALLS $$$M, a#DS$ERRHARD
00000000'9F FF48 CF FA 0EAD 1409 40$: $DS_CKLOOP 30$ ; Scope loop? A2
56 56 01 9C 0EB6 1410 ROTL #1,R6,R6 ; Select the next pattern A2
56 95 0EBA 1411 TSTB R6 ; A2
03 14 0EBC 1412 BGTR 45$ ; Done when bit 7 clears A2
FF38 31 0EBE 1413 BRW 30$ ; ... A2
00000000'9F 00000078'EF FA 0EC1 1414 45$: $DS_ENDSUB ; A2
0EC1 T5_S11_X:
00000000'9F 50 01 D0 0ECC 1415 $DS_ENDTEST CALLG $$$, a#DS$ENDSUB
0ECF TEST_005_X:: MOVL #1, R0 ; NORMAL EXIT
00000000'9F 6E FA 0ECF CALLG (SP), a#DS$BREAK ; RETURN TO TEST SEQUENCER
04 0ED6 RET
0ED7 1416 $DS_PAGE

```

```
0ED7          .SBTTL TEST 6: CSRO TEST
00000000      .PSECT TEST_006, PAGE, NOWRT
000C          1419      $DS_BGNTST      <DEFAULT,ALL,EXHAUSTIVE>
000C          DATA_006:      .LONG      0      ; TEST ARGUMENT TABLE TERMINATOR
00000000      TEST_006::      .WORD      ^M<>      ; ENTRY MASK
0000          0010
0012          1420      ;++
0012          1421      ; Functional description:
0012          1422      ;
0012          1423      ; This test, tests additional functions of CSRO that have not
0012          1424      ; previously been tested. This entire test was added in Version 01-02.
0012          1425      ;
0012          1426      ; Test algorithm:
0012          1427      ;
0012          1428      ; 1. Check that there are no stuck bits in the error address field of
0012          1429      ; CSRO. This is done by forcing a single bit error in every page
0012          1430      ; (starting with the first free row above the Diagnostic Supervisor)
0012          1431      ; in all available memory and verifying the error address is correct.
0012          1432      ;
0012          1433      ; 2. Check that a double bit error overwrites the error address and syndrome
0012          1434      ; of a single bit error. This is done by forcing a single bit error in
0012          1435      ; one page followed by a double bit error in a different page and
0012          1436      ; verifying the error address is that of the second page.
0012          1437      ;
0012          1438      ; 3. Check that the uncorrectable error bit in CSRO cannot be cleared
0012          1439      ; if the Information Lost bit is set.
0012          1440      ;
0012          1441      ; 4. Check that the correctable and uncorrectable and information lost
0012          1442      ; bits can be cleared individually.
0012          1443      ;
0012          1444      ; 5. Check that CSRO dose not update when a single bit error is forced
0012          1445      ; with the double bit error flag already set.
0012          1446      ;--
52          00F20000 8F      D0      0012      1447      MOVL      #CONTROLLER,R2      ; Get memory controller address
0019          1448      $DS_BGNSUB
0019          T6_S1::
00000000'9F      00000084'EF      FA      0019          CALLG      $$$, a#DS$BGNSUB
0024          1449      ;
0024          1450      ; Part 1
0024          1451      ;
0024          1452      ; Check that there are no stuck bits in the error address field of CSRO.
0024          1453      ;
0024          1454      ;
00000000'EF      0000003A'EF      DE      0024      1455      MOVAL      5$,UNX_UNC_PC      ; Save PC for UNEX UNCORRECTABLE ERROR A7
00000000'EF      000000A2'EF      DE      002F      1456      MOVAL      15$,RETURN_PC
003A          1457      5$:
00000000'EF      D5      003A      1458      TSTL      FREE_PAGE_P      ; Check if test should be performed
03          12      0040      1459      BNEQ      7$      ; Branch if memory to test
00C3          31      0042      1460      BRW      60$      ; Exit subtest
00000000'EF      00000000'EF      D0      0045      1461      7$:      MOVL      FREE_PAGE_P,TEMP      ; Initialize the page address
00A5          31      0050      1462      BRW      50$      ; Check if memory available
62          E0000000 8F      D0      0053      1463      10$:      MOVL      #^XE0000000,CSRO(R2)
00000000'EF      0A000000 8F      C9      005A      1464      BISL3     #PM!EDM,TEMP,CSR1(R2)      ; Put page address into pm address
04 A2          0065
0067          1465      ; and turn on ECC disable mode
```

```
00000000'FF 01 D0 0067 1466 MOVL #1,@TEMP ; Initial data pattern into memory
00000000'FF D5 006E 1467 TSTL @TEMP ; Read it
04 A2 02000000 8F CA 0074 1468 BICL #EDM,CSR1(R2) ; Clear ECC disable mode and enable
; interrupt
00000000'FF D4 007C 1470 CLRL @TEMP ; Change pattern in memory
00000000'EF 01 D0 0082 1471 MOVL #1,EXP_COR_ERR ; Indicate expected interrupt M7
00000000'EF 00000101 8F D0 0089 1472 MOVL #^X101,EXP_UNC_ERR ; Set expect Uncorrectable Error
04 A2 1C000000 8F C8 0094 1473 BISL #ERCE!PM!DCM,CSR1(R2) ; Set diagnostic check mode M3
00000000'FF D5 009C 1474 TSTL @TEMP ; Cause single bit error
04 A2 0C000000 8F CA 00A2 1475 15$: BICL #PM!DCM,CSR1(R2) ; Turn off diag. check mode
54 62 FF0001FF 8F CB 00AA 1476 BICL3 #^C<^X00FFFE00>,CSR0(R2),R4 ; Read CSRO masking error address
53 00000000'EF D0 00B2 1477 MOVL TEMP,R3 ; Get expected data
55 54 53 CD 00B9 1478 XORL3 R3,R4,R5 ; Expected = Received?
2E 13 00BD 1479 BEQL 20$ ; Branch if yes
00BF 1480 $DS_ERRHARD_S ,#0,CONTROL,-
00BF 1481 PRINT_GENERAL,#4,- ; Report the error
00BF 1482 #FMT_ECC_13,R3,R4,R5
55 DD 00BF PUSHL R5
54 DD 00C1 PUSHL R4
53 DD 00C3 PUSHL R3
00000000'8F DD 00C5 PUSHL #FMT_ECC_13
04 DD 00CB PUSHL #4
00000000'EF DF 00CD PUSHAL PRINT_GENERAL
00000000'EF DF 00D3 PUSHAL CONTROL
00 DD 00D9 PUSHL #0
01 DD 00DB PUSHL #SER
00DD ;:: TEST 6, SUBTEST 1, ERROR 1
00000000'9F 09 FB 00DD CALLS $$$M, a#DS$ERRHARD
00E4 1483 $DS_CKLOOP 10$ ; Scope loop?
00000000'9F FF6B CF FA 00E4 CALLG 10$, a#DS$CKLOOP
00000000'EF 00000200 8F C0 00ED 1484 20$: ADDL #^X200,TEMP ; Increment the test address
00000000'EF 00000000'EF D1 00F8 1485 50$: CMLP TEMP,MAXPFN ; At maximum page number yet?
03 14 0103 1486 BGTR 60$ ; Branch if YES
FF4B 31 0105 1487 BRW 10$
0108 1488 60$: $DS_ENDSUB
0108 T6_S1_X:
00000000'9F 00000084'EF FA 0108 CALLG $$$, a#DS$ENDSUB
0113 1489 $DS_BGNSUB
0113 T6_S2::
00000000'9F 00000090'EF FA 0113 CALLG $$$, a#DS$BGNSUB
011E 1490 ;
011E 1491 ; Part 2
011E 1492 ;
011E 1493 ; Check that a double bit error will overwrite the error address and syndrome
011E 1494 ; of a single bit error.
011E 1495 ;
00000000'EF 00000134'EF DE 011E 1496 MOVAL 5$,UNX_UNC_PC ; Save PC for UNEX UNCORRECTABLE ERROR A7
00000000'EF 0000018E'EF DE 0129 1497 MOVAL 15$,RETURN_PC
0134 1498 5$:
04 A2 62 E0000000 8F D0 0134 1499 10$: MOVL #^XE0000000,CSR0(R2)
50 00000000'EF DE 013B 1500 MOVAL TEMP,R0 ; Get address of single bit error
04 A2 50 000001FF 8F CB 0142 1501 BICL3 #^X1FF,R0,CSR1(R2) ; Put page address into CSR1
04 A2 0A000000 8F C8 014B 1502 BISL #PM!EDM,CSR1(R2) ; Set PM and disable ECC mode
00000000'EF 01 D0 0153 1503 MOVL #1,TEMP ; Put data pattern in memory
00000000'EF D5 015A 1504 TSTL TEMP ; Read it
04 A2 02000000 8F CA 0160 1505 BICL #EDM,CSR1(R2) ; Enable ECC
00000000'EF D4 0168 1506 CLRL TEMP ; Change pattern in memory
```

04 A2	1C000000	8F	C8	016E	1507	BISL	#ERCE!PM!DCM,CSR1(R2)	; Set diagnostic mode	M3	
	00000000	'EF	01	D0	0176	1508	MOVL	#1,EXP_COR_ERR	; Set expected error flag	M7
00000000	'EF	00000101	8F	D0	017D	1509	MOVL	#^X101,EXP_UNC_ERR	; EXPECT UNCORRECTABLE ERROR	
		00000000	'EF	D5	0188	1510	TSTL	TEMP	; Force a single bit error	
04 A2	04000000	8F	CA	018E	1511	15\$:	BICL	#DCM,CSR1(R2)	; Clear diagnostic mode	
29	00000000	'EF	00	E4	0196	1512	BBSC	#0,EXP_UNC_ERR,17\$; Branch if no uncorrectable error	
					019E	1513	\$DS_ERRHARD_S	#0,ALLMOD,-		
					019E	1514		CORRUPT_TEST_LOC,#TEMP		
	00000000	'8F	DD	019E			PUSHL	#TEMP		
	00000000	'EF	DF	01A4			PUSHAL	CORRUPT_TEST_LOC		
	00000000	'EF	DF	01AA			PUSHAL	ALLMOD		
		00	DD	01B0			PUSHL	#0		
		01	DD	01B2			PUSHL	#\$ER		
					01B4					
	00000000	'9F	05	FB	01B4		::: TEST 6, SUBTEST 2, ERROR 1			
					01BB	1515	\$DS_CKLOOP	\$\$\$M, a#DS\$ERRHARD		
00000000	'9F	FF75	CF	FA	01BB		CALLG	10\$, a#DS\$CKLOOP		
					01C4	1516	\$DS_ABORT	TEST	; Abort test corrupted test location	
			50	D4	01C4		CLRL	R0	; SEND A WARNING TO THE SUPERVISOR	
				04	01C6		RET		; TERMINATE TEST	
04 A2	51	00000000	'EF	DE	01C7	1517	17\$:	MOVAL	TEMP1,R1	; Get address of double bit error
	51	000001FF	8F	CB	01CE	1518	BICL3	#^X1FF,R1,CSR1(R2)	; Put page address into CSR1	
04 A2	04	0A000000	8F	C8	01D7	1519	BISL	#PM!EDM,CSR1(R2)	; Set ECC disable mode	
		00000000	'EF	03	D0	01DF	1520	MOVL	#3,TEMP1	; Init the memory location
					D5	01E6	1521	TSTL	TEMP1	; Read it
04 A2	02	00000000	8F	CA	01EC	1522	BICL	#EDM,CSR1(R2)	; Enable ECC	
		00000000	'EF	D4	01F4	1523	CLRL	TEMP1	; Change data in memory	
00000000	'EF	0000021A	'EF	DE	01FA	1524	MOVAL	20\$,RETURN_PC	; Setup return from machine check	
		00000000	'EF	01	D0	0205	1525	MOVL	#1,EXP_UNC_ERR	; Set expected flag
04 A2	0C	00000000	8F	C8	020C	1526	BISL	#PM!DCM,CSR1(R2)	; Enable diagnostic mode	
		00000000	'EF	D5	0214	1527	TSTL	TEMP1	; Force a double bit error	
04 A2	0C	00000000	8F	CA	021A	1528	20\$:	BICL	#PM!DCM,CSR1(R2)	; Disable diagnostic mode
54	62	1F000180	8F	CB	0222	1529	BICL3	#^X1F000180,CSR0(R2),R4	; Read and mask CSRO	
					DE	022A	1530	MOVAL	TEMP1,R1	; Generate expected data
53	51	000001FF	8F	CB	0231	1531	BICL3	#^X1FF,R1,R3	; ...	
	53	A0000041	8F	C8	0239	1532	BISL	#UCE!CE!^X41,R3	; ...	
	55	54	53	CD	0240	1533	XORL3	R3,R4,R5	; Expected = Received?	
			25	13	0244	1534	BEQL	30\$; Branch if yes	
					0246	1535	\$DS_ERRHARD_S	#0,CONTROL,-		
					0246	1536		PRINT_GENERAL,#4,-	; Report the error	
					0246	1537		#FMT_ECC_13,R3,R4,R5		
			55	DD	0246		PUSHL	R5		
			54	DD	0248		PUSHL	R4		
			53	DD	024A		PUSHL	R3		
	00000000	'8F	DD	024C			PUSHL	#FMT_ECC_13		
		04	DD	0252			PUSHL	#4		
	00000000	'EF	DF	0254			PUSHAL	PRINT_GENERAL		
	00000000	'EF	DF	025A			PUSHAL	CONTROL		
		00	DD	0260			PUSHL	#0		
		02	DD	0262			PUSHL	#\$ER		
					0264		::: TEST 6, SUBTEST 2, ERROR 2			
	00000000	'9F	09	FB	0264		CALLS	\$\$\$M, a#DS\$ERRHARD		
					026B	1538	30\$:	\$DS_CKLOOP	10\$; Scope loop?
00000000	'9F	FEC5	CF	FA	026B		CALLG	10\$, a#DS\$CKLOOP		
					0274	1539	\$DS_ENDSUB			
					0274		T6_S2_X:			
00000000	'9F	00000090	'EF	FA	0274		CALLG	\$\$\$M, a#DS\$ENDSUB		

```
00000000'9F 0000009C'EF FA 027F 1540 $DS_BGNSUB
                                027F T6_S3::
                                027F CALLG $$$, a#DS$BGNSUB
                                028A 1541 ;
                                028A 1542 ; Part 3
                                028A 1543 ;
                                028A 1544 ; Check that the Uncorrectable Error bit in CSRO cannot be cleared if
                                028A 1545 ; the Information Lost bit is set.
                                028A 1546 ;
00000000'EF 00000295'EF DE 028A 1547 MOVAL 5$,UNX_UNC_PC ; Save PC for UNEX UNCORRECTABLE ERROR A7
                                0295 1548 5$:
                                0295 1549 10$: MOVL #^XE0000000,CSRO(R2)
04 A2 50 00000000'EF DE 029C 1550 MOVAL TEMP,R0 ; Get address of test location
04 A2 50 000001FF 8F CB 02A3 1551 BICL3 #^X1FF,R0,CSR1(R2) ; Put page address in pm address
04 A2 0A000000 8F C8 02AC 1552 BISL #PM!EDM,CSR1(R2) ; Set ECC disable mode
00000000'EF 03 D0 02B4 1553 MOVL #3,TEMP ; Put initial pattern in memory
00000000'EF D5 02BB 1554 TSTL TEMP ; Read it
04 A2 02000000 8F CA 02C1 1555 BICL #EDM,CSR1(R2) ; Enable ECC
00000000'EF D4 02C9 1556 CLRL TEMP ; Change pattern in memory
00000000'EF 01 D0 02CF 1557 MOVL #1,EXP_UNC_ERR ; Set expected error flag
00000000'EF 000002EF'EF DE 02D6 1558 MOVAL 20$,RETURN_PC ; Set return pc from machine check
04 A2 0C000000 8F C8 02E1 1559 BISL #PM!DCM,CSR1(R2) ; Set diagnostic check mode
00000000'EF D5 02E9 1560 TSTL TEMP ; Force uncorrectable error
00000000'EF 01 D0 02EF 1561 20$: MOVL #1,EXP_UNC_ERR ; ...
00000000'EF 00000342'EF DE 02F6 1562 MOVAL 30$,RETURN_PC ; ...
04 A2 50 00000000'EF DE 0301 1563 MOVAL TEMP1,R0 ; Get second test address
04 A2 50 000001FF 8F CB 0308 1564 BICL3 #^X1FF,R0,CSR1(R2) ; Put page address in pm address
04 A2 0A000000 8F C8 0311 1565 BISL #PM!EDM,CSR1(R2) ; Disable ECC
00000000'EF 03 D0 0319 1566 MOVL #3,TEMP1 ; Write pattern to memory
00000000'EF D5 0320 1567 TSTL TEMP1 ; Read it
04 A2 02000000 8F CA 0326 1568 BICL #EDM,CSR1(R2) ; Enable ECC
00000000'EF D4 032E 1569 CLRL TEMP1 ; Change pattern
04 A2 0C000000 8F C8 0334 1570 BISL #PM!DCM,CSR1(R2) ; Enable diagnostic mode
00000000'EF D5 033C 1571 TSTL TEMP1 ; Force second error
04 A2 0C000000 8F CA 0342 1572 30$: BICL #PM!DCM,CSR1(R2) ; Clear diagnostic check mode
62 80000000 8F D0 034A 1573 MOVL #UCE,CSRO(R2) ; Try and clear UCE bit in CSRO
54 62 3FFFFFFF 8F CB 0351 1574 BICL3 #^C<UCEIL!UCE>,CSRO(R2),R4 ; Read error bits in CSRO
53 C0000000 8F D0 0359 1575 MOVL #UCEIL!UCE,R3 ; Get expected data
55 54 53 CD 0360 1576 XORL3 R3,R4,R5 ; Expected = Received?
25 13 0364 1577 BEQL 40$ ; Branch if yes
                                0366 1578 $DS_ERRHARD_S ,#0,CONTROL,-
                                0366 1579 PRINT_GENERAL,#4,- ; Report the error
                                0366 1580 #FMT_ECC_13,R3,R4,R5
55 DD 0366 PUSHL R5
54 DD 0368 PUSHL R4
53 DD 036A PUSHL R3
00000000'8F DD 036C PUSHL #FMT_ECC_13
04 DD 0372 PUSHL #4
00000000'EF DF 0374 PUSHAL PRINT_GENERAL
00000000'EF DF 037A PUSHAL CONTROL
00 DD 0380 PUSHL #0
01 DD 0382 PUSHL #SER
0384 ;;; TEST 6, SUBTEST 3, ERROR 1
00000000'9F 09 FB 0384 CALLS #$$M, a#DS$ERRHARD
038B 1581 40$: $DS_CKLOOP 10$ ; Scope loop?
00000000'9F FF06 CF FA 038B CALLG 10$, a#DS$CKLOOP
0394 1582 $DS_ENDSUB
```

```

00000000'9F 0000009C'EF FA 0394 T6_S3_X: CALLG $$$, a#DS$ENDSUB
                                0394 1583
                                039F 1584 $DS_BGNSUB
00000000'9F 000000A8'EF FA 039F T6_S4:: CALLG $$$, a#DS$BGNSUB
                                039F
                                03AA 1585 ;
                                03AA 1586 ; Part 4
                                03AA 1587 ;
                                03AA 1588 ; Check that the correctbale error bit clears.
                                03AA 1589 ;
00000000'EF 000003C0'EF DE 03AA 1590 MOVAL 5$,UNX_UNC_PC ; Save PC for UNEX UNCORRECTABLE ERROR A7
00000000'EF 0000041A'EF DE 03B5 1591 MOVAL 15$,RETURN_PC
                                03C0 1592 5$:
04 A2 62 E0000000 8F D0 03C0 1593 10$: MOVL #^XE0000000,CSRO(R2)
                                50 00000000'EF DE 03C7 1594 MOVAL TEMP,R0 ; Get address of test location
                                04 A2 50 000001FF 8F CB 03CE 1595 BICL3 #^X1FF,R0,CSR1(R2) ; Put page address into pm address
                                00000000'EF 01 D0 03DF 1596 BISL #PM!EDM,CSR1(R2) ; Turn on ECC disable mode
                                04 A2 0A000000 8F C8 03D7 1596 BISL #1,TEMP ; Initial data pattern into memory
                                00000000'EF D5 03E6 1598 TSTL TEMP ; Read it
                                04 A2 02000000 8F CA 03EC 1599 BICL #EDM,CSR1(R2) ; Clear ECC disable mode M3
                                00000000'EF D4 03F4 1600 CLRL TEMP ; interrupt
                                00000000'EF 01 D0 03FA 1601 MOVL #1,EXP_COR_ERR ; Change pattern in memory
00000000'EF 00000101 8F D0 0401 1602 MOVL #^X101,EXP_UNC_ERR ; Indicate expected interrupt M7
                                04 A2 1C000000 8F C8 040C 1604 BISL #ERCE!PM!DCM,CSR1(R2) ; Set diagnostic check mode M3
                                00000000'EF D5 0414 1605 TSTL TEMP ; Cause single bit error
                                04 A2 0C000000 8F CA 041A 1606 15$: BICL #PM!DCM,CSR1(R2) ; Turn off diag. check mode
                                29 00000000'EF 00 E4 0422 1607 BBSC #0,EXP_UNC_ERR,17$ ; Branch if no uncorrectable error
                                042A 1608 $DS_ERRHARD_S ,#0,ALLMOD,-
                                042A 1609 CORRUPT_TEST_LOC,#TEMP
                                DD 042A PUSHL #TEMP
                                DF 0430 PUSHAL CORRUPT_TEST_LOC
                                DF 0436 PUSHAL ALLMOD
                                DD 043C PUSHL #0
                                DD 043E PUSHL #SER
                                0440 ;:: TEST 6, SUBTEST 4, ERROR 1
                                00000000'9F 05 FB 0440 CALLS $$$M, a#DS$ERRHARD
                                00000000'9F FF75 CF FA 0447 1610 $DS_CKLOOP 10$
                                0450 1611 $DS_ABORT TEST ; Abort test corrupted test location
                                50 D4 0450 CLRL R0 ; SEND A WARNING TO THE SUPERVISOR
                                04 0452 RET ; TERMINATE TEST
                                54 62 20000000 8F C8 0453 1612 17$: BISL #CE,CSRO(R2) ; Try and clear CE bit
                                62 DFFF FFFF 8F CB 045A 1613 BICL3 #^CCE,CSRO(R2),R4 ; Read the CE bit
                                53 D4 0462 1614 CLRL R3 ; Get expected data
                                55 54 53 CD 0464 1615 XORL3 R3,R4,R5 ; Expected = Received?
                                25 13 0468 1616 BEQL 20$ ; Branch if yes
                                046A 1617 $DS_ERRHARD_S ,#0,CONTROL,-
                                046A 1618 PRINT_GENERAL,#4,- ; Report the error
                                046A 1619 #FMT_ECC_2,R3,R4,R5
                                55 DD 046A PUSHL R5
                                54 DD 046C PUSHL R4
                                53 DD 046E PUSHL R3
                                00000000'8F DD 0470 PUSHL #FMT_ECC_2
                                04 DD 0476 PUSHL #4
                                00000000'EF DF 0478 PUSHAL PRINT_GENERAL

```

```
00000000'EF DF 047E PUSHAL CONTROL
00 DD 0484 PUSHL #0
02 DD 0486 PUSHL #SER
0488
00000000'9F 09 FB 0488 ::: TEST 6, SUBTEST 4, ERROR 2
048F 1620 20$: $DS_CKLOOP 10$ ; Scope loop?
00000000'9F FF2D CF FA 048F CALLG 10$, a#DS$CKLOOP
0498 1621 ;
0498 1622 ; Check that the UCE bit clears
0498 1623 ;
0498 1624 30$: MOVL #^XE0000000,CSR0(R2)
049F 1625 MOVAL TEMP,R1 ; Get address of double bit error
04A6 1626 BICL3 #^X1FF,R1,CSR1(R2) ; Put page address into CSR1
04AF 1627 BISL #PM!EDM,CSR1(R2) ; Set ECC disable mode
04B7 1628 MOVL #3,TEMP ; Init the memory location
04BE 1629 TSTL TEMP ; Read it
04C4 1630 BICL #EDM,CSR1(R2) ; Enable ECC
04CC 1631 CLRL TEMP ; Change data in memory
04D2 1632 MOVAL 40$,RETURN_PC ; Setup return from machine check
04DD 1633 MOVL #1,EXP_UNC_ERR ; Set expected flag
04E4 1634 BISL #PM!DCM,CSR1(R2) ; Enable diagnostic mode
04EC 1635 TSTL TEMP ; Force a double bit error
04F2 1636 40$: BICL #PM!DCM,CSR1(R2) ; Disable diagnostic mode
04FA 1637 BISL #UCE,CSR0(R2) ; Try and clear UCE bit
0501 1638 BICL3 #^CUCE,CSR0(R2),R4 ; Read the UCE bit
0509 1639 CLRL R3 ; Get expected data
050B 1640 XORL3 R3,R4,R5 ; Expected = Received?
050F 1641 BEQL 50$ ; Branch if yes
0511 1642 $DS_ERRHARD_S ,#0,CONTROL,-
0511 1643 PRINT_GENERAL,#4,- ; Report the error
0511 1644 #FMT_ECC_2,R3,R4,R5
0511 DD 0511 PUSHL R5
0513 DD 0513 PUSHL R4
0515 DD 0515 PUSHL R3
0517 DD 0517 PUSHL #FMT_ECC_2
051D DD 051D PUSHL #4
051F DF 051F PUSHAL PRINT_GENERAL
0525 DF 0525 PUSHAL CONTROL
052B DD 052B PUSHL #0
052D DD 052D PUSHL #SER
052F ::: TEST 6, SUBTEST 4, ERROR 3
052F 1645 50$: $DS_CKLOOP 30$ ; Scope loop?
0536 1646 CALLG 30$, a#DS$CKLOOP
053F 1647 ;
053F 1648 ; Check that UCEIL Bit clears
053F 1649 ;
053F 1650 60$: MOVL #^XE0000000,CSR0(R2)
0546 1651 MOVAL TEMP,R1 ; Get address of double bit error
054D 1652 BICL3 #^X1FF,R1,CSR1(R2) ; Put page address into CSR1
0556 1653 BISL #PM!EDM,CSR1(R2) ; Set ECC disable mode
055E 1654 MOVL #3,TEMP ; Init the memory location
0565 1655 TSTL TEMP ; Read it
056B 1656 BICL #EDM,CSR1(R2) ; Enable ECC
0573 1657 CLRL TEMP ; Change data in memory
0579 1658 MOVAL 70$,RETURN_PC ; Setup return from machine check
```

```

00000000'EF 01 D0 0584 1659 MOVL #1,EXP_UNC_ERR ; Set expected flag
04 A2 0C000000 8F C8 058B 1660 BISL #PM!DCM,CSR1(R2) ; Enable diagnostic mode
00000000'EF 03 D5 0593 1661 TSTL TEMP ; Force a double bit error
00000000'EF 000005B1'EF DE 0599 1662 70$: MOVAL 80$,RETURN_PC ; Setup for another double bit error
00000000'EF 01 D0 05A4 1663 MOVL #1,EXP_UNC_ERR ; ...
00000000'EF 03 D5 05AB 1664 TSTL TEMP ; Force second double bit error
04 A2 0C000000 8F CA 05B1 1665 80$: BISL #PM!DCM,CSR1(R2) ; Disable diagnostic mode
62 40000000 8F C8 05B9 1666 BISL #UCEIL,CSR0(R2) ; Try and clear UCEIL bit
54 62 BFFFFFFF 8F CB 05C0 1667 BICL3 #^CUCEIL,CSR0(R2),R4 ; Read the UCEIL bit
53 D4 05C8 1668 CLRL R3 ; Get expected data
55 54 53 CD 05CA 1669 XORL3 R3,R4,R5 ; Expected = Received?
25 13 05CE 1670 BEQL 90$ ; Branch if yes
05D0 1671 $DS_ERRHARD_S ,#0,CONTROL,-
05D0 1672 PRINT_GENERAL,#4,- ; Report the error
05D0 1673 #FMT_ECC_2,R3,R4,R5
55 DD 05D0 PUSHL R5
54 DD 05D2 PUSHL R4
53 DD 05D4 PUSHL R3
00000000'8F DD 05D6 PUSHL #FMT_ECC_2
04 DD 05DC PUSHL #4
00000000'EF DF 05DE PUSHAL PRINT_GENERAL
00000000'EF DF 05E4 PUSHAL CONTROL
00 DD 05EA PUSHL #0
04 DD 05EC PUSHL #SER
05EE ;:: TEST 6, SUBTEST 4, ERROR 4
00000000'9F 09 FB 05EE CALLS $$$M, a#DS$ERRHARD
00000000'9F FF46 CF FA 05F5 1674 90$: $DS_CKLOOP 60$ ; Scope loop?
05FE 1675 $DS_ENDSUB CALLG 60$, a#DS$CKLOOP
00000000'9F 000000A8'EF FA 05FE T6_S4_X: CALLG $$$, a#DS$ENDSUB
0609 1676 $DS_BGNSUB
00000000'9F 000000B4'EF FA 0609 T6_S5:: CALLG $$$, a#DS$BGNSUB
0614 1678 ;
0614 1679 ; Part 5
0614 1680 ;
0614 1681 ; This subtest was added in version 00-07.
0614 1682 ; Check that CSRO dose not change when a single bit error is forced with a
0614 1683 ; double bit error already logged.
0614 1684 ;
62 E0000000 8F D0 0614 1685 10$: MOVL #^XE0000000,CSR0(R2)
04 A2 50 00000000'EF DE 061B 1686 MOVAL TEMP1,R0 ; Get address of double bit error
04 A2 50 000001FF 8F CB 0622 1687 BICL3 #^X1FF,R0,CSR1(R2) ; Put page address in CSR1
00000000'EF 03 D0 0633 1688 BISL #PM!EDM,CSR1(R2) ; Enable page and ECC dis. mode
04 A2 0A000000 8F C8 062B 1689 MOVL #3,TEMP1 ; Init the memory location
00000000'EF 01 D0 063A 1690 BICL #EDM,CSR1(R2) ; Enable ECC
00000000'EF 00000668'EF DE 0642 1691 MOVL #1,EXP_UNC_ERR ; Set the expected error flag
00000000'EF 01 D0 0649 1692 MOVAL 20$,RETURN_PC ; Set the return PC
04 A2 04000000 8F D4 0654 1693 CLRL TEMP1 ; Change pattern in memory
00000000'EF 03 D5 065A 1694 BISL #DCM,CSR1(R2) ; Set diagnostic mode
53 62 1F000180 8F CB 0662 1695 TSTL TEMP1 ; Force a double bit error
08 BB 0670 1696 20$: BICL3 #^X1F000180,CSR0(R2),R3 ; Get CSRO of double bit error A8
04 A2 04000000 8F CA 0672 1697 PUSHR #^M<R3> ; Save
50 00000000'EF DE 067A 1698 BICL #DCM,CSR1(R2) ; Clear diagnostic mode
MOVAL TEMP, R0 ; Get address to force single bit error

```

```
04 A2 50 000001FF 8F CB 0681 1700 BICL3 #^X1FF,R0,CSR1(R2) ; Put page address in CSR1
04 A2 0A000000 8F C8 068A 1701 BISL #PM!EDM,CSR1(R2) ; Disable ECC
00000000'EF 01 D0 0692 1702 MOVL #1,TEMP ; Init pattern in memory
04 A2 02000000 8F CA 0699 1703 BICL #EDM,CSR1(R2) ; Enable ECC
00000000'EF D4 06A1 1704 CLRL TEMP ; Change pattern in memory
00000000'EF 01 D0 06A7 1705 MOVL #1,EXP_COR_ERR ; Set expected error flag
00000000'EF 00000101 8F D0 06AE 1706 MOVL #^X101,EXP_UNC_ERR
00000000'EF 000006D2'EF DE 06B9 1707 MOVAL 25$,RETURN_PC
04 A2 14000000 8F C8 06C4 1708 BISL #ERCE!DCM,CSR1(R2) ; Enable diagnostic mode
00000000'EF D5 06CC 1709 TSTL TEMP ; Force single bit error
04 A2 D4 06D2 1710 25$: CLRL CSR1(R2) ; Disable diagnostic mode
2B 00000000'EF 00 E4 06D5 1711 BBSC #0,EXP_UNC_ERR,17$ ; Branch if no uncorrectable error
06DD 1712 $DS_ERRHARD_S ,#0,ALLMOD,-
06DD 1713 CORRUPT_TEST_LOC,#TEMP
00000000'8F DD 06DD PUSHL #TEMP
00000000'EF DF 06E3 PUSHAL CORRUPT_TEST_LOC
00000000'EF DF 06E9 PUSHAL ALLMOD
00 DD 06EF PUSHL #0
01 DD 06F1 PUSHL #SER
06F3 :::: TEST 6, SUBTEST 5, ERROR 1
00000000'9F 05 FB 06F3 CALLS $$$M, a#DS$ERRHARD
08 BA 06FA 1714 POPR #^M<R3> ; Save CSRO from double bit
06FC 1715 $DS_CKLOOP 10$
00000000'9F FF14 CF FA 06FC CALLG 10$, a#DS$CKLOOP
0705 1716 $DS_ABORT TEST ; Abort test corrupted test location
50 D4 0705 CLRL R0 ; SEND A WARNING TO THE SUPERVISOR
04 0707 RET ; TERMINATE TEST
08 BA 0708 1717 17$: POPR #^M<R3> ; Get saved CSRO from double bit
54 62 1F000180 8F CB 070A 1718 BICL3 #^X1F000180,CSRO(R2),R4 ; Read and mask CSRO
53 20000000 8F C8 0712 1719 BISL #CE,R3 ;
55 54 53 CD 0719 1720 XORL3 R3,R4,R5 ; Received = Expected?
25 13 071D 1721 BEQL 30$ ; Branch if yes
071F 1722 $DS_ERRHARD_S ,#0,CONTROL,-
071F 1723 PRINT_GENERAL,#4,- ; Report the error
071F 1724 #FMT_ECC_13,R3,R4,R5
55 DD 071F PUSHL R5
54 DD 0721 PUSHL R4
53 DD 0723 PUSHL R3
00000000'8F DD 0725 PUSHL #FMT_ECC_13
04 DD 072B PUSHL #4
00000000'EF DF 072D PUSHAL PRINT_GENERAL
00000000'EF DF 0733 PUSHAL CONTROL
00 DD 0739 PUSHL #0
02 DD 073B PUSHL #SER
073D :::: TEST 6, SUBTEST 5, ERROR 2
00000000'9F 09 FB 073D CALLS $$$M, a#DS$ERRHARD
0744 1725 30$: $DS_CKLOOP 10$ ; Scope loop?
00000000'9F FECC CF FA 0744 CALLG 10$, a#DS$CKLOOP
074D 1726 $DS_ENDSUB
074D T6_S5_X: CALLG $$$, a#DS$ENDSUB
00000000'9F 000000B4'EF FA 074D $DS_ENDTEST
50 01 D0 0758 1727 MOVL #1, R0 ; NORMAL EXIT
075B TEST_006_X:: CALLG (SP), a#DS$BREAK
00000000'9F 6E FA 075B RET ; RETURN TO TEST SEQUENCER
04 0762 $DS_PAGE
0763 1728
```

```
00000000 0763 .SBTTL TEST 7: BOOTSTRAP ROM TEST
00000000 0018 .PSECT TEST_007, PAGE, NOWRT
00000000 0018 1731 $DS_BGNTST <DEFAULT,ALL,EXHAUSTIVE>
00000000 0018 DATA_007:
00000000 0018 .LONG 0 ; TEST ARGUMENT TABLE TERMINATOR
0000 001C TEST_007::
0000 001C .WORD ^M<> ; ENTRY MASK
001E 1732 ;++
001E 1733 ; Functional Description:
001E 1734 ;
001E 1735 ; This test looks for the presence of upto 4 bootstrap roms. If it
001E 1736 ; finds them, a message is typed indicating the ROM number and the
001E 1737 ; device type of the ROM. The checksum of the rom is then calculated
001E 1738 ; and compared to the checksum contained in the ROM.
001E 1739 ;
001E 1740 ; If no ROMS are found, a message is typed indicating so.
001E 1741 ;
001E 1742 ; This entire test was added in version 01-02.
001E 1743 ;--
001E 1744
001E 1745 $DS_BGNSUB
001E T7_S1::
00000000'9F 000000C0'EF FA 001E CALLG $$$, a#DS$BGNSUB
00000000'EF 00000034'EF DE 0029 1746 MOVAL 5$,UNX_UNC_PC ; Save PC for UNEX UNCORRECTABLE ERROR A7
00000400 8F 00F20000 8F C9 0034 1747 5$:
00000000'EF 00000000'EF 003F 1748 BISL3 #CONTROLLER,#^X400,TEMP1; Insert ROM offset into base address
00000000'EF 00000000'EF 0044 1749 ; of the controller
00000000'EF 02 08 56 D4 0044 1750 CLRL TEMP ; Initialize a ROM present flag
00000000'EF 00000000'EF 01 90 004A 1751 CLRL R6 ; Initialize a ROM number count
00000000'EF 00000112'EF DE 0055 1752 10$: INSV R6,#8,#2,TEMP1 ; Insert the ROM number into the address
00000000'EF 53 00000000'FF D0 0067 1753 MOVVB #1,EXP_NX_MEM ; Setup to catch non existant memory
00000000'EF 53 00000000'FF D0 0067 1754 MOVAL 35$,RETURN_PC ;
00000000'EF 53 00000000'FF D0 0067 1755 MOVL aTEMP1,R3 ; See if ROM is present by testing byte
00000000'EF 03 12 0070 1756 TSTB R3 ; 0 for all zero's or all one's
00000000'EF 00CA 31 0072 1757 BNEQ 15$ ; Not all zero's
00000000'EF 53 FF 8F 91 0075 1758 BRW 40$ ; Not present
00000000'EF 03 12 0079 1759 15$: CMPB #-1,R3 ;
00000000'EF 00C1 31 007B 1760 BNEQ 16$ ; Present
00000000'EF 00000000'EF D6 007E 1761 BRW 40$ ; Not present
0000FE54'EF 01 D1 0084 1762 16$: INCL TEMP ; Set the ROM present flag
00000000'EF 35 1F 008B 1763 CMPL #1,DSA$GL_PASSNO ; Pass number 1?
00000000'EF 54 00000003'EF DE 008D 1764 BLSSU 17$ ; Branch if no
00000000'EF 74 53 90 0094 1765 MOVAL DEVICE_TYPE+3,R4 ; Get address of device type word
00000000'EF 53 53 F8 8F 9C 0097 1766 MOVVB R3,-(R4) ; Save the device type of the ROM
00000000'EF 74 53 90 009C 1767 ROTL #-8,R3,R3 ;
00000000'EF 56 41 8F 81 00A1 1768 MOVVB R3,-(R4) ;
00000000'EF 57 00000000'EF DE 00AA 1769 TSTB -(R4) ;
00000000'EF 54 DD 00B1 1770 ADDB3 #^A/A/,R6,ROM_TYPE+1 ; Convert rom number into A,B,C, or D
00000000'EF 57 DD 00B3 1771 MOVAL ROM_TYPE,R7 ;
00000000'EF 9F 00B5 1772 $DS_PRINTB $ FMT_ROM_1,R7,R4 ; Print the ROM message and device type
00000000'9F 03 FB 00BB 1773 ;
00C2 1773 ;
```

```
00C2 1774 ; Check the checksum of the ROM
00C2 1775 ;
53 00000000'EF D0 00C2 1776 17$: MOVL TEMP1,R3 ; Get ROM base address
54 D4 00C9 1777 CLRL R4 ; Initialize a byte counter
55 D4 00CB 1778 CLRL R5 ; Initialize the checksum
55 83 80 00CD 1779 20$: ADDB (R3)+,R5 ; Generate the checksum
FFF3 54 01 000000FE 8F F1 00D0 1780 ACBL #254,#1,R4,20$ ; Add 255 bytes
54 63 90 00DA 1781 MOVB (R3),R4 ; Get the checksum in the ROM
55 54 91 00DD 1782 CMPB R4,R5 ; Is checksum correct?
25 13 00E0 1783 BEQL 30$ ; Branch if yes
00E2 1784 $DS_ERRHARD_S ,#0,CONTROL,-
00E2 1785 PRINT_GENERAL,#4,- ; Report the error
00E2 1786 #FMT_ROM_2,R7,R4,R5
55 DD 00E2 PUSHL R5
54 DD 00E4 PUSHL R4
57 DD 00E6 PUSHL R7
00000000'8F DD 00E8 PUSHL #FMT_ROM_2
04 DD 00EE PUSHL #4
00000000'EF DF 00F0 PUSHAL PRINT_GENERAL
00000000'EF DF 00F6 PUSHAL CONTROL
00 DD 00FC PUSHL #0
01 DD 00FE PUSHL #SER
00000000'9F 09 FB 0100 ::: TEST 7, SUBTEST 1, ERROR 1
0100 CALLS $$$M, a#DS$ERRHARD
0107 1787 30$: $DS_CKLOOP 17$ ; Scope loop?
00000000'9F B8 AF FA 0107 CALLG 17$, a#DS$CKLOOP
002D 31 010F 1788 BRW 40$
0112 1789
0112 1790 ;
0112 1791 ; Come here if referance caused a non existant memory machine check
0112 1792 ;
0112 1793 35$: $DS_ERRHARD_S ,#0,CONTROL,-
0112 1794 PRINT_GENERAL,#2,- ; Report the error
0112 1795 #FMT_ROM_4,TEMP1
00000000'EF DD 0112 PUSHL TEMP1
00000000'8F DD 0118 PUSHL #FMT_ROM_4
02 DD 011E PUSHL #2
00000000'EF DF 0120 PUSHAL PRINT_GENERAL
00000000'EF DF 0126 PUSHAL CONTROL
00 DD 012C PUSHL #0
02 DD 012E PUSHL #SER
00000000'9F 07 FB 0130 ::: TEST 7, SUBTEST 1, ERROR 2
0130 CALLS $$$M, a#DS$ERRHARD
0137 1796 $DS_CKLOOP 17$ ; Scope loop?
00000000'9F 88 AF FA 0137 CALLG 17$, a#DS$CKLOOP
013F 1797
FF07 56 01 03 F1 013F 1798 40$: ACBL #3,#1,R6,10$ ; Check all 3 ROM slots
0145 1799
00000000'EF D5 0145 1800 TSTL TEMP ; Any ROMS at all?
0D 12 014B 1801 BNEQ 50$ ; Branch if yes
014D 1802 $DS_PRINTB_S FMT_ROM_3 ; Tell operator no ROMS present
00000000'EF 9F 014D PUSHAB FMT_ROM_3
00000000'9F 01 FB 0153 CALLS $$$N, a#DS$PRINTB
015A 1803 50$: $DS_ENDSUB
015A T7_S1_X:
00000000'9F 000000C0'EF FA 015A CALLG $$$, a#DS$ENDSUB
0165 1804 $DS_ENDTEST
```

ZZ-ECKAM-3.2
ECKAM
03-02

TEST 7: BOOTSTRAP ROM TEST
MEMORY SUBSYSTEM TESTS
TEST 7: BOOTSTRAP ROM TEST

G 11

13-MAY-1987 Fiche 1 Frame G11 Sequence 136
23-JAN-1987 14:30:50 VAX/VMS Macro V04-00 Page 49
23-JAN-1987 14:29:55 [ZECCHINO.ECKAM]ECKAM1.MAR;6 (10)

50	01	D0	0165		MOVL	#1, R0		; NORMAL EXIT
			0168	TEST_007_X::				
00000000'9F	6E	FA	0168		CALLG	(SP), a#DS\$BREAK		
		04	016F		RET			; RETURN TO TEST SEQUENCER
			0170	1805	\$DS_PAGE			


```
00000000'EF DF 0095 PUSHAL PRINT_GENERAL
00000000'EF DF 009B PUSHAL CPU
00 DD 00A1 PUSHL #0
01 DD 00A3 PUSHL #SER
00000000'9F 09 FB 00A5 ;:: TEST 8, SUBTEST 1, ERROR 1
00AC 1851 40$: $DS_CKLOOP CALLS $$$M, a#DS$ERRHARD
00000000'9F 85 AF FA 00AC 1852 40$: $DS_CKLOOP CALLS $$$M, a#DS$CKLOOP ; Scope loop?
00B4 1852 $DS_ENDSUB
00000000'9F 000000CC'EF FA 00B4 T8_S1_X: CALLG $$$, a#DS$ENDSUB
00BF 1853
00BF 1854 $DS_BGNSUB
00000000'9F 000000D8'EF FA 00BF T8_S2:: CALLG $$$, a#DS$BGNSUB
00CA 1855 ;
00CA 1856 ; Part 2
00CA 1857 ;
00CA 1858 ; Force two uncorrectable memory errors and check BUS ERROR register
00CA 1859 ;
00000000'EF 000000D5'EF DE 00CA 1860 MOVAL 5$,UNX_UNC_PC ; Save PC for UNEX UNCORRECTABLE ERROR A7
00D5 1861 5$:
04 A2 62 E0000000 8F D0 00D5 1862 10$: MOVL #^XE0000000,CSR0(R2)
04 A2 52 00F20000 8F D0 00DC 1863 MOVL #CONTROLLER,R2 ; Get address of memory controller
04 A2 50 00000000'EF DE 00E3 1864 MOVAL TEMP,R0 ; Get address of test location
04 A2 50 000001FF 8F CB 00EA 1865 BICL3 #^X1FF,R0,CSR1(R2) ; Put page address in pm address
04 A2 0A000000 8F C8 00F3 1866 BISL #PM!EDM,CSR1(R2) ; Set ECC disabl mode
00000000'EF 03 D0 00FB 1867 MOVL #3,TEMP ; Init data pattern in test location
00000000'EF D5 0102 1868 TSTL TEMP ; Read it
04 A2 02000000 8F CA 0108 1869 BICL #EDM,CSR1(R2) ; Enable ECC
00000000'EF D4 0110 1870 CLRL TEMP ; Change pattern in test location
00000000'EF 00010001 8F D0 0116 1871 MOVL #^X10001,EXP_UNC_ERR ; Set the expected error flag
00000000'EF 0000013A'EF DE 0121 1872 MOVAL 20$,RETURN_PC ; Set the return PC
04 A2 0C000000 8F C8 012C 1873 BISL #PM!DCM,CSR1(R2) ; Set diagnostic mode
00000000'EF D5 0134 1874 TSTL TEMP ; Force uncorrectable error
00000000'EF 00000152'EF DE 013A 1875 20$: MOVAL 30$,RETURN_PC ; Setup for another error
00000000'EF 01 90 0145 1876 MOVB #1,EXP_UNC_ERR ; Set the expected flag
00000000'EF D5 014C 1877 TSTL TEMP ; Force the second error
04 A2 0C000000 8F CA 0152 1878 30$: BICL #PM!DCM,CSR1(R2) ; Clear diagnostic mode
04 A2 53 06 D0 015A 1879 MOVL #6,R3 ; Get the expected data
54 00000000'EF 9A 015D 1880 MOVZBL BUS_ERR_REG,R4 ; Get the received data
55 54 53 CD 0164 1881 XORL3 R3,R4,R5 ; Expected = Received?
00000000'EF 25 13 0168 1882 BEQL 40$ ; Branch if yes
016A 1883 $DS_ERRHARD_S ,#0,CPU,-
016A 1884 PRINT_GENERAL,#4,- ; Report the error
016A 1885 #FMT_CPU_3,R3,R4,R5
55 DD 016A PUSHL R5
54 DD 016C PUSHL R4
53 DD 016E PUSHL R3
00000000'8F DD 0170 PUSHL #FMT_CPU_3
04 DD 0176 PUSHL #4
00000000'EF DF 0178 PUSHAL PRINT_GENERAL
00000000'EF DF 017E PUSHAL CPU
00 DD 0184 PUSHL #0
01 DD 0186 PUSHL #SER
0188 ;:: TEST 8, SUBTEST 2, ERROR 1
00000000'9F 09 FB 0188 CALLS $$$M, a#DS$ERRHARD
```

```
00000000'9F  FF42 CF  FA 018F 1886 40$:  $DS_CKLOOP 10$ ; Scope loop?
                                018F          CALLG 10$, a#DS$CKLOOP
                                0198 1887  $DS_ENDSUB
00000000'9F  000000D8'EF  FA 0198          T8_S2_X:          CALLG $$$, a#DS$ENDSUB
                                01A3 1888
                                01A3 1889  $DS_BGNSUB          ; D4 A
00000000'9F  000000E4'EF  FA 01A3          T8_S3::          CALLG $$$, a#DS$BGNSUB
                                01AE 1890 ;
                                01AF 1891 ; Part 3
                                01AE 1892 ;
                                01AE 1893 ; Force two correctable errors and check the BUS ERROR register
                                01AE 1894 ;
00000000'EF  000001C4'EF  DE 01AE 1895          MOVAL 5$,UNX_UNC_PC          ; Save PC for UNEX UNCORRECTABLE ERROR A7
00000000'EF  00000223'EF  DE 01B9 1896          MOVAL 15$,RETURN_PC
                                01C4 1897 5$:
                                01C4 1898 10$:  MOVL #^XE0000000,CSR0(R2)
                                01CB 1899          MOVL #CONTROLLER,R2          ; Get address of memory controller D4 A
04 A2 50 00000000'EF  DE 01D2 1900          MOVAL TEMP,R0          ; Get address of test location D4 A
04 A2 50 000001FF 8F  CB 01D9 1901          BICL3 #^X1FF,R0,CSR1(R2)          ; Put page address in pm address D4 A
04 A2 0A000000 8F  C8 01E2 1902          BISL #PM!EDM,CSR1(R2)          ; Disable ECC D4 A
00000000'EF  01  D0 01EA 1903          MOVL #1,TEMP          ; Initialize test location D4 A
04 A2 02000000 8F  CA 01F1 1904          BICL #EDM,CSR1(R2)          ; Enable ECC D4 A
00000000'EF  00000000'EF  D4 01F9 1905          CLRL TEMP          ; Change the test pattern D4 A
00000000'EF  00010001 8F  D0 01FF 1906          MOVL #^X10001,EXP_COR_ERR          ; Set the expected error flag D4 A
00000000'EF  00000101 8F  D0 020A 1907          MOVL #^X101,EXP_UNC_ERR
04 A2 1C000000 8F  C8 0215 1908          BISL #ERCE!PM!DCM,CSR1(R2)          ; Set diagnostic mode M3 D
00000000'EF  D5 021D 1909          TSTL TEMP          ; Force a single bit error D4 A
2C 00000000'EF  00  E4 0223 1910 15$:  BBSC #0,EXP_UNC_ERR,17$          ; Branch if no uncorrectable error
                                022B 1911          $DS_ERRHARD_S #0,ALLMOD,-
                                022B 1912          CORRUPT_TEST_LOC,#TEMP
                                DD 022B          PUSHL #TEMP
                                DF 0231          PUSHAL CORRUPT_TEST_LOC
                                DF 0237          PUSHAL ALLMOD
                                DD 023D          PUSHL #0
                                DD 023F          PUSHL #SER
                                0241          ::: TEST 8, SUBTEST 3, ERROR 1
00000000'9F  05  FB 0241          CALLS $$$M, a#DS$ERRHARD
                                04 A2  D4 0248 1913          CLRL CSR1(R2)
                                024B 1914          $DS_CKLOOP 10$
00000000'9F  FF75 CF  FA 024B          CALLG 10$, a#DS$CKLOOP
                                0254 1915          $DS_ABORT TEST          ; Abort test corrupted test location
                                50  D4 0254          CLRL R0          ; SEND A WARNING TO THE SUPERVISOR
                                04  D4 0256          RET          ; TERMINATE TEST
00000000'EF  01  90 0257 1916 17$:  MOVB #1,EXP_COR_ERR          ; Set the expected error flag again D4 A
00000000'EF  00000101 8F  D0 025E 1917          MOVL #^X101,EXP_UNC_ERR
00000000'EF  000002A7'EF  DE 0269 1918          MOVAL 18$,RETURN_PC
04 A2 04000000 8F  CA 0274 1919          BICL #DCM,CSR1(R2)          ; Clear diagnostic mode A7
04 A2 02000000 8F  C8 027C 1920          BISL #EDM,CSR1(R2)          ; Disable ECC A7
00000000'EF  01  D0 0284 1921          MOVL #1,TEMP          ; Init the pattern in memory A7
04 A2 02000000 8F  CA 028B 1922          BICL #EDM,CSR1(R2)          ; Enable ECC A7 M
00000000'EF  D4 0293 1923          CLRL TEMP          ; Change pattern in memory A7
04 A2 04000000 8F  C8 0299 1924          BISL #DCM,CSR1(R2)          ; Enable diagnostic mode A7
00000000'EF  D5 02A1 1925          TSTL TEMP          ; Force a second error D4 A
04 A2 0C000000 8F  CA 02A7 1926 18$:  BICL #PM!DCM,CSR1(R2)          ; Clear diagnostic mode D4 A
29 00000000'EF  00  E4 02AF 1927          BBSC #0,EXP_UNC_ERR,19$          ; Branch if no uncorrectable error
```

```
00000000'8F DD 02B7 1928 $DS_ERRHARD_S ,#0,ALLMOD,-
00000000'EF DF 02B7 1929 CORRUPT_TEST_LOC,#TEMP
00000000'EF DF 02BD PUSHL #TEMP
00000000'EF DF 02C3 PUSHAL CORRUPT_TEST_LOC
00 DD 02C9 PUSHAL ALLMOD
02 DD 02CB PUSHL #0
02C3 02CD :;; TEST 8, SUBTEST 3, ERROR 2
00000000'9F 05 FB 02CD CALLS $$$M, a#DS$ERRHARD
00000000'9F FECC CF FA 02D4 1930 $DS_CKLOOP 10$
02DD 1931 $DS_ABORT TEST ; Abort test corrupted test location
04 02DD CLRL R0 ; SEND A WARNING TO THE SUPERVISOR
04 02DF RET ; TERMINATE TEST
53 03 D0 02E0 1932 19$: MOVL #3,R3 ; Get expected data D4 A
54 00000000'EF 9A 02E3 1933 MOVZBL BUS_ERR,REG,R4 ; Get received data D4 A
55 54 53 CD 02EA 1934 XORL3 R3,R4,R5 ; Expected = Received? D4 A
25 13 02EE 1935 BEQL 20$ ; Branch if yes D4 A
02F0 1936 $DS_ERRHARD_S ,#0,CPU,-
02F0 1937 PRINT_GENERAL,#4,- ; Report the error
02F0 1938 #FMT_CPU_3,R3,R4,R5 ; D4 A
55 DD 02F0 PUSHL R5
54 DD 02F2 PUSHL R4
53 DD 02F4 PUSHL R3
00000000'8F DD 02F6 PUSHL #FMT_CPU_3
04 DD 02FC PUSHL #4
00000000'EF DF 02FE PUSHAL PRINT_GENERAL
00000000'EF DF 0304 PUSHAL CPU
00 DD 030A PUSHL #0
03 DD 030C PUSHL #SER
030E :;; TEST 8, SUBTEST 3, ERROR 3
00000000'9F 09 FB 030E CALLS $$$M, a#DS$ERRHARD
00000000'9F FEAB CF FA 0315 1939 20$: $DS_CKLOOP 10$ ; Scope loop? D4 A
031E 1940 $DS_ENDSUB 10$, a#DS$CKLOOP ; D4 A
031E T8_S3_X:
00000000'9F 000000E4'EF FA 031E CALLG $$$, a#DS$ENDSUB
0329 1941 $DS_ENDTEST MOVL #1, R0 ; NORMAL EXIT
032C TEST_008_X::
00000000'9F 6E FA 032C CALLG (SP), a#DS$BREAK
04 0333 RET ; RETURN TO TEST SEQUENCER
0334 1942 $DS_PAGE
```

```
00000000 0334 .SBTTL TEST 9: CPU XB ERROR BIT TEST
00000000 0000 .PSECT TEST_009, PAGE, NOWRT
0018 0018 1945 $DS_BGNTST <DEFAULT,ALL,EXHAUSTIVE>
0018 0018 DATA_009:
00000000 0018 .LONG 0 ; TEST ARGUMENT TABLE TERMINATOR
0000 001C TEST_009::
0000 001C .WORD ^M<> ; ENTRY MASK
001E 1946 ;++
001E 1947 ;
001E 1948 ; Functional Description:
001E 1949 ;
001E 1950 ; This test checks that XB bit in the MACHINE CHECK ERROR SUMMARY
001E 1951 ; register functions correctly. This is done by forcing a NXM, and
001E 1952 ; UNCORRECTABLE error with a DATA reference and checking that the
001E 1953 ; bit does not set and forcing the errors with an I-STREAM reference
001E 1954 ; and checking that the bit sets.
001E 1955 ;
001E 1956 ; 1. Nonexistant memory error.
001E 1957 ; 2. Uncorrectable memory error.
001E 1958 ;
001E 1959 ; This entire test was added in version 01-02.
001E 1960 ;--
001E 1961
001E 1962 $DS_BGNSUB
00000000'9F 000000F0'EF FA 001E T9_S1::
001E CALLG $$$, a#DS$BGNSUB
0029 1963 ;
0029 1964 ; Part 1
0029 1965 ;
0029 1966 ; Check XB bit with a nonexistant memory error. To force an XB reference
0029 1967 ; to do this requires that the program JUMP to a nonexistant address.
0029 1968 ;
00000000 52 00F20000 8F D0 0029 1969 MOVL #CONTROLLER,R2 ; Get Controller Address
00000000'EF 0000003B'EF DE 0030 1970 MOVAL 5$,UNX_UNC_PC ; Save PC for UNEX UNCORRECTABLE ERROR A7
003B 1971 5$:
00000000 62 E0000000 8F D0 003B 1972 10$: MOVL #^XE0000000,CSRO(R2) ; Cleanup the controller
00000000'EF 00F20000 8F C9 0042 1973 BISL3 #CONTROLLER,#^X10,R0 ; Get nonexistant address
00000000'EF 00010001 8F D0 004A 1974 MOVL #^X10001,EXP_NX_MEM ; Set the expected flag
00000000'EF 00000062'EF DE 0055 1975 MOVAL 20$,RETURN_PC ; Setup the return PC
0060 1976 TSTL (R0) ; Force NXM ERROR
0054 26 DB 0062 1977 20$: MFPR #^X26,R4 ; Get ERROR SUMMARY register
0053 08 D0 0065 1978 MOVL #8,R3 ; Get expected data
55 54 53 CD 0068 1979 XORL3 R3,R4,R5 ; Expected = Received?
006C 1980 BEQL 30$ ; Branch if yes
006E 1981 $DS_ERRHARD_S ,#0,CPU,-
006E 1982 PRINT_GENERAL,#4,- ; Report the error
006E 1983 #FMT_CPU_4,R3,R4,R5
0055 DD 006E PUSHL R5
0054 DD 0070 PUSHL R4
0053 DD 0072 PUSHL R3
00000000'8F DD 0074 PUSHL #FMT_CPU_4
0004 DD 007A PUSHL #4
00000000'EF DF 007C PUSHAL PRINT_GENERAL
00000000'EF DF 0082 PUSHAL CPU
00 DD 0088 PUSHL #0
01 DD 008A PUSHL #ER
```

```
00000000'9F 09 FB 008C      ::: TEST 9, SUBTEST 1, ERROR 1
                26 0F DA 0093 1984 30$: MTPR      $$$M, a#DS$ERRHARD
                                #^XF, #^X26      ; Clear SUMMARY REGISTER
00000000'9F A2 AF FA 0096 1985      $DS_CKLOOP 10$      ; Scope Loop?
                                CALLG 10$, a#DS$CKLOOP
                                009E 1986
                                009E 1987 ;
                                009E 1988 ; Now force an XB NXM error.
                                009E 1989 ;
50 10 00F20000 8F C9 009E 1990 40$: BISL3      #CONTROLLER, #^X10, R0      ; Get nonexistent address
00000000'EF 00010001 8F D0 00A6 1991      MOVL      #^X10001, EXP_NX_MEM      ; Set the expected flag
00000000'EF 000000BE'EF DE 00B1 1992      MOVAL     50$, RETURN_PC          ; Set the return PC
                60 17 00BC 1993      JMP      (R0)                    ; Force the error
                54 26 DB 00BE 1994 50$: MFPR      #^X26, R4          ; Get ERROR SUMMARY register
                53 09 D0 00C1 1995      MOVL      #9, R3                 ; Get expected data
55 54 53 CD 00C4 1996      XORL3     R3, R4, R5              ; Expected = Received?
                25 13 00C8 1997      BEQL     60$                    ; Branch if yes
                00CA 1998      $DS_ERRHARD_S , #0, CPU, -
                00CA 1999      PRINT_GENERAL, #4, -      ; Report the error
                00CA 2000      #FMT_CPU_4, R3, R4, R5
                55 DD 00CA      PUSHL     R5
                54 DD 00CC      PUSHL     R4
                53 DD 00CE      PUSHL     R3
00000000'8F DD 00D0      PUSHL     #FMT_CPU_4
                04 DD 00D6      PUSHL     #4
00000000'EF DF 00D8      PUSHAL    PRINT_GENERAL
00000000'EF DF 00DE      PUSHAL    CPU
                00 DD 00E4      PUSHL     #0
                02 DD 00E6      PUSHL     #SER
00000000'9F 09 FB 00E8      ::: TEST 9, SUBTEST 1, ERROR 2
                26 0F DA 00EF 2001 60$: MTPR      $$$M, a#DS$ERRHARD
                                #^XF, #^X26      ; Clear SUMMARY REGISTER
00000000'9F A9 AF FA 00F2 2002      $DS_CKLOOP 40$      ; Scope Loop?
                                CALLG 40$, a#DS$CKLOOP
00000000'9F 000000F0'EF FA 00FA 2003      $DS_ENDSUB
                                T9_S1_X:
                                CALLG $$$, a#DS$ENDSUB
00000000'9F 000000FC'EF FA 0105 2004      $DS_BGNSUB
                                T9_S2::
                                CALLG $$$, a#DS$BGNSUB
0110 2005 ;
0110 2006 ; Part 2
0110 2007 ;
0110 2008 ; Force Uncorrectable error and check ERROR SUMMARY register
0110 2009 ;
00000000'EF 0000011B'EF DE 0110 2010      MOVAL     5$, UNX_UNC_PC          ; Save PC for UNEX UNCORRECTABLE ERROR A7
                52 00F20000 8F D0 011B 2011 5$:
                50 00000000'EF DE 0122 2012 10$: MOVL      #CONTROLLER, R2      ; Get address of memory controller
04 A2 50 000001FF 8F CB 0129 2013      MOVAL     TEMP, R0              ; Get address of test location
04 A2 04 A2 0A000000 8F C8 0132 2014      BICL3     #^X1FF, R0, CSR1(R2)   ; Put page address in pm address
                00000000'EF 03 D0 013A 2015      BISL      #PM!EDM, CSR1(R2)     ; Disable ECC
04 A2 04 A2 02000000 8F CA 0141 2016      MOVL      #3, TEMP              ; Init test location
                00000000'EF D4 0149 2017      BICL     #EDM, CSR1(R2)         ; Enable ECC
00000000'EF 00010001 8F D0 014F 2018      CLRL     TEMP                  ; Change test location
00000000'EF 00000173'EF DE 015A 2019      MOVL     #^X10001, EXP_UNC_ERR  ; Set the expected error flag
04 A2 04 A2 0C000000 8F C8 0165 2020      MOVAL     20$, RETURN_PC        ; Setup the return PC
                0C000000 8F C8 0165 2021      BISL     #PM!DCM, CSR1(R2)     ; Set diagnostic mode
```

```

04 A2 00000000'EF D5 016D 2022 TSTL TEMP ; Force the error
      0C000000 8F CA 0173 2023 20$: BICL #PM!DCM,CSR1(R2) ; Clear diagnostic mode
      54 26 DB 017B 2024 MFPR #^X26,R4 ; Get ERROR SUMMARY register
      53 08 D0 017E 2025 MOVL #8,R3 ; Get expected data
      55 54 53 CD 0181 2026 XORL3 R3,R4,R5 ; Expected = Received?
      25 13 0185 2027 BEQL 30$ ; Branch if yes
      0187 2028 $DS_ERRHARD_S ,#0,CPU,-
      0187 2029 PRINT_GENERAL,#4,- ; Report the error
      0187 2030 #FMT_CPU_4,R3,R4,R5
      55 DD 0187 PUSHL R5
      54 DD 0189 PUSHL R4
      53 DD 018B PUSHL R3
      00000000'8F DD 018D PUSHL #FMT_CPU_4
      04 DD 0193 PUSHL #4
      00000000'EF DF 0195 PUSHAL PRINT_GENERAL
      00000000'EF DF 019B PUSHAL CPU
      00 DD 01A1 PUSHL #0
      01 DD 01A3 PUSHL #SER
      01A5 ;::: TEST 9, SUBTEST 2, ERROR 1
      00000000'9F 09 FB 01A5 CALLS $$$M, a#DS$ERRHARD
      62 E0000000 8F D0 01AC 2031 30$: MOVL #^XE0000000,CSR0(R2) ; Cleanup the controller
      26 0F DA 01B3 2032 MTPR #^XF,#^X26 ; Clear the ERROR SUMMARY register
      00000000'9F FF61 CF FA 01B6 2033 $DS_CKLOOP 10$ ; Scope loop?
      01B6 2034 CALLG 10$, a#DS$CKLOOP
      01BF 2034 ;
      01BF 2035 ; Force an XB uncorrectable error
      01BF 2036 ;
      01BF 2037 ;
      01BF 2038
      52 00F20000 8F D0 01BF 2039 40$: MOVL #CONTROLLER,R2 ; Get address of memory controller
      50 00000000'EF DE 01C6 2040 MOVAL TEMP,R0 ; Get address of test location
      04 A2 50 000001FF 8F CB 01CD 2041 BICL3 #^X1FF,R0,CSR1(R2) ; Put page address in pm address
      04 A2 0A000000 8F C8 01D6 2042 BISL #PM!EDM,CSR1(R2) ; Disable ECC
      00000000'EF D4 01DE 2043 CLRL TEMP ; Init test location
      04 A2 02000000 8F CA 01E4 2044 BICL #EDM,CSR1(R2) ; Enable ECC
      00000000'EF 01010101 8F D0 01EC 2045 MOVL #^X01010101,TEMP ; Change test location
      00000000'EF 00010001 8F D0 01F7 2046 MOVL #^X10001,EXP_UNC_ERR ; Set the expected error flag
      00000000'EF 0000021B'EF DE 0202 2047 MOVAL 50$,RETURN_PC ; Setup the return PC
      04 A2 0C000000 8F C8 020D 2048 BISL #PM!DCM,CSR1(R2) ; Set diagnostic mode
      00000000'EF 17 0215 2049 JMP TEMP ; Force the error
      04 A2 0C000000 8F CA 021B 2050 50$: BICL #PM!DCM,CSR1(R2) ; Clear diagnostic mode
      54 26 DB 0223 2051 MFPR #^X26,R4 ; Get ERROR SUMMARY register
      53 09 D0 0226 2052 MOVL #9,R3 ; Get expected data
      55 54 53 CD 0229 2053 XORL3 R3,R4,R5 ; Expected = Received?
      25 13 022D 2054 BEQL 60$ ; Branch if yes
      022F 2055 $DS_ERRHARD_S ,#0,CPU,-
      022F 2056 PRINT_GENERAL,#4,- ; Report the error
      022F 2057 #FMT_CPU_4,R3,R4,R5
      55 DD 022F PUSHL R5
      54 DD 0231 PUSHL R4
      53 DD 0233 PUSHL R3
      00000000'8F DD 0235 PUSHL #FMT_CPU_4
      04 DD 023B PUSHL #4
      00000000'EF DF 023D PUSHAL PRINT_GENERAL
      00000000'EF DF 0243 PUSHAL CPU
      00 DD 0249 PUSHL #0
      02 DD 024B PUSHL #SER

```



```
027D          .SBTTL TEST 10: MOVING INVERSIONS TEST
00000000      .PSECT TEST_010, PAGE, NOWRT
001C
001C 2067      $DS_BGNTST <DEFAULT,ALL,EXHAUSTIVE>
001C          DATA_010:
00000000 001C          .LONG 0 ; TEST ARGUMENT TABLE TERMINATOR
0020          TEST_010::
0000 0020      .WORD ^M<> ; ENTRY MASK
0022 2068      $DS_BNQUICK 2$ ; Check quick flag if not set continue
0022          BBC #DSA$V_QUICK,- ; BR IF NOT QUICK
03 0000FE00 08 E1 0022          a#DSA$GL_FLAGS, 2$
0024          $DS_ABORT TEST ; Otherwise abort test.
002A 2069      CLRL R0 ; SEND A WARNING TO THE SUPERVISOR
002A          RET ; TERMINATE TEST
002C
002D 2070      ;++
002D 2071      ; Functional description:
002D 2072      ;
002D 2073      ; In this test the arrays are tested rigorously. The moving inversions
002D 2074      ; algorithm is used in this test, providing an exhaustive test of the
002D 2075      ; RAMs. This test will take approximately 10 minutes per 256kb array
002D 2076      ; and approximately 45 minutes per 1024kb array. Set the quick flag
002D 2077      ; for a quick verify of the memory arrays.
002D 2078      ;
002D 2079      ; Note: Not all rows of all arrays are tested only those rows not protected
002D 2080      ; by the Diagnostic Supervisor are tested. If in doubt of the reliability
002D 2081      ; of this portion of memory, a manual swap of the array cards, to a higher
002D 2082      ; slot should be made. The first printout of the array card under test,
002D 2083      ; (tests 10,11,12) will tell you where testing had begun.
002D 2084      ;
002D 2085      ; Test algorithm:
002D 2086      ;
002D 2087      ; 1. Write all rows of all arrays under test with all zeros.
002D 2088      ;
002D 2089      ; 2. Go to the beginning of the first row under test. Read all zeros,
002D 2090      ; write all ones, and read all ones. Continue sequentially until all
002D 2091      ; longwords in this row are filled with all ones.
002D 2092      ;
002D 2093      ; 3. Go back to the beginning of the first row under test. Read all
002D 2094      ; ones, write all zeros, and read all zeros. Continue sequentially
002D 2095      ; until all longwords in this row are filled with all zeros.
002D 2096      ;
002D 2097      ; 4. Go to the end of the first row under test. Read all zeros, write
002D 2098      ; all ones, and read all ones. Continue backwards until all
002D 2099      ; longwords in this row are filled with all ones.
002D 2100      ;
002D 2101      ; 5. Go back to the end of the first row under test. Read all ones,
002D 2102      ; write all zeros, and read all zeros. Continue backwards until all
002D 2103      ; longwords in this row are filled with all zeros.
002D 2104      ;
002D 2105      ; 6. Repeat steps 2 and 3 for 14 or 16 passes (the number of address bits
002D 2106      ; per chip) dependent on type of array. For each successive pass, a
002D 2107      ; different address bit is used as the least significant bit in counting
002D 2108      ; such that every address line is toggled at the fastest possible rate.
002D 2109      ;
002D 2110      ; 7. Repeat steps 2 - 6 for data patterns which provide the check bit
002D 2111      ; memory with complementing patterns.
002D 2112      ;
```

8. Repeat step 1 through 7 for all rows of all arrays under test.

```

002D 2113 ;
002D 2114 ;--
002D 2115
00000000'EF D5 002D 2116 2$: TSTL FREE_PAGE_P ; Any memory to test?
10 12 0033 2117 BNEQ 5$ ; Branch if yes
00000000'EF 9F 0035 2118 $DS_PRINTB S FMT_NO_ARRAYS ; Type no arrays to test message A5
00000000'9F 01 FB 003B PUSHAB FMT_NO_ARRAYS
049A 31 0042 2119 CALLS $$$N, a#DS$PRINTB
50 00F20000 8F D0 0045 2120 5$: BRW 144$ ; Exit test A5
62 E0000000 8F D0 004C 2121 MOVL #CONTROLLER,R0 ; Get address of controller A2
04 A0 10000000 8F C8 0053 2122 BISL #^XE0000000,CSR0(R2) ; Cleanup the controller
00000000'EF 01 D0 005B 2123 MOVL #ERCE,CSR1(R0) ; Enable error interrupts A2 M
00000000'EF 000004B7'EF DE 0062 2124 MOVL #1,EXP UNC ERR ; Set the expected error flag A2
00000000'EF 16 006D 2125 MOVAL 200$,RETURN_PC ; Set the return pc A2
00000000'EF D4 0073 2126 JSB READ_MAP ; This sets up the slot config table
00000000'EF 01 90 0079 2127 CLRL TEMP1 ; Clear array message flags
0080 2128 MOVB #1,SUMMARY_FLAG ; Enable the summary section
0080 2129 ;
0080 2130 ; Initialize the row counter and address pointer as a function of the
0080 2131 ; first free page above the Supervisor and the program.
58 00000000'EF D0 0080 2132 MOVL FREE_PAGE_P,R8 ; Set address to base of row
00000000'EF 58 D0 0087 2133 MOVL R8,TEMP ; Set base address
5B D4 008E 2134 CLRL R11 ; Slot counter
00000000'EF 94 0090 2135 CLRB DIGIT1 ; Current array counter
00000000'EF D4 0096 2136 CLRL L_CUR_ARRAY ; Initilize array counter
00000000'EF D4 009C 2137 CLRL FLAG1 ; Used to flag 256kb arrays
00000000'EF D4 00A2 2138 CLRL FLAG3 ; Used to flag 1024kb arrays
5B 00000000'EF D0 00A8 2139 MOVL L_CUR_ARRAY,R11 ; Keep track of current array
00000000'EF4B FF 8F 91 00AF 2140 CMPB #-1,SLOT[R11] ; Find array type
20 13 00B8 2141 BEQL 8$ ; Branch if 256kb array
00BA 2142 ;
08 00000000'EF4B 00 E1 00BA 2143 BBC #0,SLOT[R11],7$
57 58 05 14 EF 00C3 2144 EXTZV #20,#5,R8,R7
001C 31 00C8 2145 BRW 9$
00CB 2146 ;
00000000'EF 02 90 00CB 2147 7$: MOVB #2,FLAG1 ; Set flag for 1024kb array
57 58 05 12 EF 00D2 2148 EXTZV #18,#5,R8,R7 ; Get row number of address 1024kb array
000D 31 00D7 2149 BRW 9$ ; Continue normal flow
00000000'EF FF 8F 90 00DA 2150 8$: MOVB #-1,FLAG1 ; Set flag for 256kb array
57 58 05 10 EF 00E2 2151 EXTZV #16,#5,R8,R7 ; Get row number of address 256kb array
00000000'EF D4 00E7 2152 9$: CLRL PATCNT ; Initialize pattern counter
00000000'EF D4 00ED 2153 CLRL NUM_COR_ERR ; Clear the single bit error counter
55 00000000'EF D0 00F3 2154 10$: MOVL PATCNT,R5 ; Get pattern count
00000000'EF45 D0 00FA 2155 MOVL MOVI_PAT_0[R5],PATTERN ; Get first pattern for this pass
00000004'EF 00000000'EF45 D0 0106 2156 MOVL MOVI_PAT_1[R5],PATTERN + 4 ; Get second pattern for this pass
00000000'EF 16 0112 2157 JSB WHICH_ARRAY ; Determine which array is under test
5B 00000000'EF D1 0118 2158 CMPL L_CUR_ARRAY,R11 ; Find out if we need another array
13 13 011F 2159 BEQL 22$ ; Branch if we do
5B 00000000'EF D0 0121 2160 MOVL L_CUR_ARRAY,R11 ; New array number
00000000'EF 00000000'EF4B 90 0128 2161 MOVB SLOT[R11],FLAG1 ; New array type
0D 00000000'EF 50 E2 0134 2162 22$: BBSS R0,TEMP1,23$ ; If array N msg hasn't been printed
013C 2163 $DS_PRINTB S FMT_ARRAY ; Then print it
00000000'EF 9F 013C PUSHAB FMT_ARRAY
00000000'9F 01 FB 0142 CALLS $$$N, a#DS$PRINTB
53 00000000'EF D0 0149 2164 23$: MOVL PATTERN,R3 ; Prepare to write the whole row
56 00000000'EF D0 0150 2165 MOVL TEMP,R6 ; Initialize longword address

```

```

00000000'EF  FF 8F  91 0157 2166  CMPB #-1,FLAG1 ; Check array type
                1F  13 015F 2167  BEQL 26$ ; Branch if 256kb array
                0161 2168 ;
00000000'EF  02  91 0161 2169  CMPB #2,FLAG1 ;[3.0] Check array type
                0B  13 0168 2170  BEQL 25$ ; Branch if 4Mb array
59 56 000FFFFC 8F  C1 016A 2171  ADDL3 #^XFFFFC,R6,R9 ; Calculate highest address of
                0013 31 0172 2172  ; ; row 1024 kb
                0172 2173  BRW 27$ ; Continue with flow
                0175 2174 ;
59 56 0003FFFC 8F  C1 0175 2175 25$: ADDL3 #^X3FFFC,R6,R9 ; Calculate highest address of row 256kb
                0008 31 017D 2176  BRW 27$ ; Continue with flow
59 56 0000FFFC 8F  C1 0180 2177 26$: ADDL3 #^XFFFC,R6,R9 ; Calculate highest address of row 64kb
                66 53  D0 0188 2178 27$: MOVL R3,(R6) ; Write the pattern
FFF7 56 04 59  F1 018B 2179  ACBL R9,#4,R6,27$ ; Do for all longwords in the row
                59  D4 0191 2180  CLRL R9 ; Initialize address bit counter
                00000000'EF 94 0193 2181  CLRB SEQUENCE ; Indicate forward sequence
                00000000'EF 94 0199 2182  CLRB DATA ; Indicate first forward sequence
5A 58 FFFF0000 8F  D4 019F 2183 30$: CLRL R8 ; Initialize address counter
                07  12 01A1 2184 34$: BICL3 #^XFFFF0000,R8,R10 ; Time for supervisor call?
                01AB 2185  BNEQ 38$ ; Branch if no
                01AB 2186  $DS_BREAK
00000000'9F  6E  FA 01AB ; CALLG (SP), @#DS$BREAK
                5A 58  D0 01B2 2187 38$: MOVL R8,R10 ; Copy current raw address
00000000'EF  FF 8F  91 01B5 2188  CMPB #-1,FLAG1 ; Check array type
                19  13 01BD 2189  BEQL 42$ ; Branch if 256kb array
                01BF 2190 ;
00000000'EF  02  91 01BF 2191  CMPB #2,FLAG1
                08  13 01C6 2192  BEQL 40$
5A 12 0E 5A  F0 01C8 2193  INSV R10,#14,#18,R10 ; Copy raw address into upper 18 bits
                000D 31 01CD 2194  BRW 43$
                01D0 2195 ;
5A 10 10 5A  F0 01D0 2196 40$: INSV R10,#16,#16,R10 ; Copy raw address into upper 16 bits
                0005 31 01D5 2197  BRW 43$ ; Continue flow
5A 0E 12 5A  F0 01D8 2198 42$: INSV R10,#18,#14,R10 ; Copy raw address into upper 14 bits
                5A 5A 59 9C 01DD 2199 43$: ROTL R9,R10,R10 ; Rotate R9 times
00000000'EF  FF 8F  91 01E1 2200  CMPB #-1,FLAG1 ; Check array type
                19  13 01E9 2201  BEQL 46$ ; Branch if 256kb array
                01EB 2202 ;
00000000'EF  02  91 01EB 2203  CMPB #2,FLAG1 ;[3.0] Check array type
                08  13 01F2 2204  BEQL 45$ ; Branch if 1024kb array
50 5A 12 00  EF 01F4 2205  EXTZV #0,#18,R10,R0 ; Calculate actual
                000D 31 01F9 2206  BRW 48$ ; Continue flow
                01FC 2207 ;
50 5A 10 00  EF 01FC 2208 45$: EXTZV #0,#16,R10,R0 ; Calculate actual
                0005 31 0201 2209  BRW 48$ ; Continue flow
50 5A 0E 00  EF 0204 2210 46$: EXTZV #0,#14,R10,R0 ; Calculate actual 256kb arrays
                50 50 02 78 0209 2211 48$: ASHL #2,R0,R0 ; Obtain longword address
56 00000000'EF  D0 020D 2212  MOVL TEMP,R6 ; Initialize address pointer
                56 50  C0 0214 2213  ADDL R0,R6 ; Address to use
                66  D5 0217 2214  TSTL (R6) ; Read the location under test
                00000000'EF 95 0219 2215  TSTB ERR_LOG_FULL ; Is error log full? [3.1]
                03  13 021F 2216  BEQL 50$
                01CC 31 0221 2217  BRW 133$
09 00000000'EF  E8 0224 2218 50$: BLBS DATA,55$ ; If this is 0 -> 1 then
53 00000004'EF  D0 022B 2219  MOVL PATTERN+4,R3 ; Use the second pattern
                07  11 0232 2220  BRB 57$ ; And branch
53 00000000'EF  D0 0234 2221 55$: MOVL PATTERN,R3 ; Otherwise, use the first pattern

```

		66	53	D0	023B	2222	57\$:	MOVL	R3,(R6)	; Write the pattern	
			66	D5	023E	2223		TSTL	(R6)	; Read the data	M2
		00000000	'EF	95	0240	2224		TSTB	ERR_LOG_FULL	; Is error log full? [3.1]	
			03	13	0246	2225		BEQL	58\$		
			01A5	31	0248	2226		BRW	133\$		
		00000000	'EF	FF	8F	91	58\$:	CMPB	#-1,FLAG1	; Check array type	
			23	13	0253	2228		BEQL	61\$; Branch if 256kb arrays	
					0255	2229					
		00000000	'EF	02	91	0255		CMPB	#2,FLAG1	;[3.0] Check array type	
			0D	13	025C	2231		BEQL	60\$; Branch if 1024kb arrays	
FF39	58	01	0003FFFF	8F	F1	025E	2232	ACBL	#262143,#1,R8,34\$; Do for all addresses in the row 1024kb	
			0017	31	0268	2233		BRW	62\$; Continue flow	
					026B	2234					
FF2C	58	01	0000FFFF	8F	F1	026B	2235	60\$:	ACBL	#65535,#1,R8,34\$; Do for all addresses in the row 256kb
			000A	31	0275	2236		BRW	62\$; Continue flow	
FF1F	58	01	00003FFF	8F	F1	0278	2237	61\$:	ACBL	#16383,#1,R8,34\$; Do for all addresses in the row 64kb
					0282	2238	62\$:	\$DS_BREAK		; Check for CTRL C	A7
		00000000	'9F	6E	FA	0282		CALLG	(SP), a#DS\$BREAK		
		00000000	'EF	95	0289	2239		TSTB	DATA	; Done for 1's and 0's both?	
			0A	12	028F	2240		BNEQ	70\$; Branch if so	
		00000000	'EF	01	90	0291	2241	MOVB	#1,DATA	; Set data flag	
			FF04	31	0298	2242		BRW	30\$; And branch to do forward seq again	
		00000000	'EF	94	029B	2243	70\$:	CLRB	DATA	; Clear data flag	
		00000000	'EF	FF	8F	91	80\$:	CMPB	#-1,FLAG1	; Check array type	
			1D	13	02A9	2245		BEQL	85\$; Branch if 256kb arrays	
					02AB	2246					
		00000000	'EF	02	91	02AB		CMPB	#2,FLAG1	;[3.0] Check array type	
			0A	13	02B2	2248		BEQL	83\$; Branch if 1024kb arrays	
58		0003FFFF	8F	D0	02B4	2249		MOVL	#262143,R8	; Initialize address counter 4Mb array	
			0011	31	02BB	2250		BRW	88\$; Continue flow	
					02BE	2251					
58		0000FFFF	8F	D0	02BE	2252	83\$:	MOVL	#65535,R8	; Initialize address counter 1024kb array	
			0007	31	02C5	2253		BRW	88\$; Continue flow	
58		00003FFF	8F	D0	02C8	2254	85\$:	MOVL	#16383,R8	; Initialize address counter 256kb array	
5A	58	FFFF8000	8F	CB	02CF	2255	88\$:	BICL3	#^XFFFF8000,R8,R10	; Time for supervisor call?	
			07	12	02D7	2256		BNEQ	89\$; Branch if no	
					02D9	2257		\$DS_BREAK			
		00000000	'9F	6E	FA	02D9		CALLG	(SP), a#DS\$BREAK		
		00000000	'EF	5A	58	D0	89\$:	MOVL	R8,R10	; Copy current row address	
			FF	8F	91	02E3		CMPB	#-1,FLAG1	; Check array type	
			19	13	02EB	2260		BEQL	92\$; Branch if 256kb arrays	
					02ED	2261					
		00000000	'EF	02	91	02ED		CMPB	#2,FLAG1	;[3.0] Check array type	
			08	13	02F4	2263		BEQL	91\$; Branch if 1024kb arrays	
5A	12	0E	5A	F0	02F6	2264		INSV	R10,#14,#18,R10	; Copy row address into upper 18 bits	
			000D	31	02FB	2265		BRW	93\$; Continue flow	
					02FE	2266					
5A	10	10	5A	F0	02FE	2267	91\$:	INSV	R10,#16,#16,R10	; Copy row address into upper 16 bits	
			0005	31	0303	2268		BRW	93\$; Continue flow	
5A	0E	12	5A	F0	0306	2269	92\$:	INSV	R10,#18,#14,R10	; Copy row address into upper 14 bits	
		5A	5A	9C	030B	2270	93\$:	ROTL	R9,R10,R10	; Rotate R9 times	
		00000000	'EF	FF	8F	91		CMPB	#-1,FLAG1	; Check array type	
			19	13	0317	2272		BEQL	96\$; Branch if 256kb arrays	
					0319	2273					
		00000000	'EF	02	91	0319		CMPB	#2,FLAG1	;[3.0] Check array type	
			08	13	0320	2275		BEQL	94\$; Branch if 1024kb arrays	
50	5A	12	00	EF	0322	2276		EXTZV	#0,#18,R10,R0	; Calculate actual for 4Mb arrays	

```

000D 31 0327 2277 BRW 98$ ; Continue flow
          032A 2278 ;
50 5A 10 00 EF 032A 2279 94$: EXTZV #0,#16,R10,R0 ; Calculate actual
          0005 31 032F 2280 BRW 98$ ; Continue flow
50 5A 0E 00 EF 0332 2281 96$: EXTZV #0,#14,R10,R0 ; Calculate actual for 256kb arrays
          50 50 02 78 0337 2282 98$: ASHL #2,R0,R0 ; Obtain longword address
56 00000000'EF D0 033B 2283 MOVL TEMP,R6 ; Initialize address pointer
          56 50 C0 0342 2284 ADDL R0,R6 ; Address to use
          66 D5 0345 2285 TSTL (R6) ; Read the location under test
          00000000'EF 95 0347 2286 TSTB ERR_LOG_FULL ; Is error log full? [3.1] M2
          03 13 034D 2287 BEQL 100$
          009E 31 034F 2288 BRW 133$
          09 00000000'EF E8 0352 2289 100$: BLBS DATA,105$ ; If this is 0 -> 1 then
53 00000004'EF D0 0359 2290 MOVL PATTERN+4,R3 ; Use the second pattern
          07 11 0360 2291 BRB 107$ ; And branch
53 00000000'EF D0 0362 2292 105$: MOVL PATTERN,R3 ; Otherwise, use the first pattern
          66 53 D0 0369 2293 107$: MOVL R3,(R6) ; Write the pattern
          66 D5 036C 2294 TSTL (R6) ; Read the data
          00000000'EF 95 036E 2295 TSTB ERR_LOG_FULL ; Is error log full? [3.1] M2
          7A 12 0374 2296 BNEQ 133$
          58 D7 0376 2297 110$: DECL R8 ; Subtract one from address counter
          03 19 0378 2298 BLSS 120$ ; Branch if done for all addresses
          FF52 31 037A 2299 BRW 88$ ; Branch to do next address
          037D 2300 120$: $DS_BREAK ; Check for CTRL C A7
          00000000'9F 6E FA 037D CALLG (SP), a#DS$BREAK
          00000000'EF 95 0384 2301 TSTB DATA ; Done for 1's and 0's both?
          0A 12 038A 2302 BNEQ 128$ ; Branch if so
          00000000'EF 01 90 038C 2303 MOVB #1,DATA ; Set data flag
          FF0B 31 0393 2304 BRW 80$ ; Branch to do reverse seq again
          00000000'EF 94 0396 2305 128$: CLRB DATA ; Clear data flag
00000000'EF FF 8F 91 039C 2306 CMPB #-1,FLAG1 ; Check array type
          1B 13 03A4 2307 BEQL 131$ ; Branch if 256kb arrays
          03A6 2308 ;
          00000000'EF 02 91 03A6 2309 CMPB #2,FLAG1 ;[3.0] Check array type
          09 13 03AD 2310 BEQL 130$ ; Branch if 256kb arrays
FDEA 59 01 11 F1 03AF 2311 ACBL #17,#1,R9,30$ ; Do for all address bits in address
          000F 31 03B5 2312 BRW 132$ ; Continue flow
          03B8 2313 ;
FDE1 59 01 0F F1 03B8 2314 130$: ACBL #15,#1,R9,30$ ; Do for all address bits in address
          0006 31 03BE 2315 BRW 132$ ; Continue flow
FDD8 59 01 0D F1 03C1 2316 131$: ACBL #13,#1,R9,30$ ; Do for all address bits in address
FD22 00000000'EF 01 01 F1 03C7 2317 132$: ACBL #1,#1,PATCNT,10$ ; Do for data and check bit patterns
          50 57 D0 03D1 2318 MOVL R7,R0 ; Copy row number
          51 D4 03D4 2319 CLRL R1 ; Clear high part of quadword for EDIV
          SA 52 50 04 7B 03D6 2320 EDIV #4,R0,R2,R10 ; Calculate row # of array under test
          03DB 2321 $DS_PRINTB_S FMT_ROW_MSG,- ; Print the row number under test
          03DB 2322 R10,NUM_COR_ERR ; single bit errors for this row
          00000000'EF DD 03DB PUSHL NUM_COR_ERR
          SA DD 03E1 PUSHL R10
          00000000'EF 9F 03E3 PUSHAB FMT_ROW_MSG
          00000000'9F 03 FB 03E9 CALLS $$$N, a#DS$PRINTB
          03F0 2323 ;
          03F0 2324 ; If running under manufacturing APT and there are single bit errors,
          03F0 2325 ; make an ERRHARD call.
          03F0 2326 ;
0000FE00'EF 80000000 8F D3 03F0 2327 133$: BITL #DSA$M_APT,DSA$GL_FLAGS ; APT mode?
          3E 13 03FB 2328 BEQL 134$ ; Branch if no

```

```

00000000'EF 95 03FD 2329 TSTB NUM_COR_ERR ; Any single bit errors?
          4E 13 0403 2330 BEQL 135$ ; Branch if no
SA 57 FFFFFFFC 8F CB 0405 2331 BICL3 #^XFFFFFFFC,R7,R10 ; Compute current array
      52 57 FE 8F 78 040D 2332 ASHL #-2,R7,R2 ; and row from rows done
          0412 2333 $DS_ERRHARD_S ,#0,-
          0412 2334 ARRAY_SGL_BIT,-
          0412 2335 PRINT_GENERAL,-
          0412 2336 P1 = #4,-
          0412 2337 P2 = #FMT_SGL_BIT_ERR,-
          0412 2338 P3 = R2,-
          0412 2339 P4 = R10,-
          0412 2340 P5 = NUM_COR_ERR; Report the error
00000000'EF DD 0412 PUSHL NUM_COR_ERR
          5A DD 0418 PUSHL R10
          52 DD 041A PUSHL R2
00000000'8F DD 041C PUSHL #FMT_SGL_BIT_ERR
          04 DD 0422 PUSHL #4
00000000'EF DF 0424 PUSHAL PRINT_GENERAL
00000000'EF DF 042A PUSHAL ARRAY_SGL_BIT
          00 DD 0430 PUSHL #0
          01 DD 0432 PUSHL #SER
          0434 ::: TEST 10, SUBTEST 0, ERROR 1
00000000'9F 09 FB 0434 CALLS $$$M, a#DS$ERRHARD
          00000000'EF 95 043B 2341 134$: TSTB ERR_LOG_FULL
          10 13 0441 2342 BEQL 135$
          0443 2343 $DS_PRINTB_S FMT_LOG_FULL ; If error log is full then
          00000000'EF 9F 0443 PUSHL FMT_LOG_FULL
00000000'9F 01 FB 0449 CALLS $$$N, a#DS$PRINTB
          50 D4 0450 2344 $DS_ABORT TEST ; exit test [3.1]
          04 0452 CLRL R0 ; SEND A WARNING TO THE SUPERVISOR
00000000'EF FF 8F 91 0453 2345 135$: CMPB #-1,FLAG1 ; TERMINATE TEST
          3D 13 045B 2346 BEQL 137$ ; Check array type
          045D 2347 ; Branch for 256kb arrays
00000000'EF 02 91 045D 2348 CMPB #2,FLAG1 ;[3.0] Check array type
          24 13 0464 2349 BEQL 136$ ; Branch for 1024kb arrays
          57 D6 0466 2350 INCL R7 ; Bump the row counter 4Mb arrays
00000000'EF 00100000 8F C0 0468 2351 ADDL2 #^X100000,TEMP ; Calculate the new base address
00000000'EF 00000000'EF D1 0473 2352 CMPL TEMP,TRANSITION4 ; Are we at end of 4Mb arrays?
          27 13 047E 2353 BEQL 139$
00000000'EF 02 90 0480 2354 MOVB #2,FLAG1
          001D 31 0487 2355 BRW 139$ ; Continue flow
          048A 2356 ;
00000000'EF 00040000 8F D6 048A 2357 136$: INCL R7 ; Bump the row counter 1024kb arrays
          000D 31 0497 2358 ADDL2 #^X40000,TEMP ; Calculate the new base address
          57 D6 049A 2360 137$: INCL R7 ; Continue flow
00000000'EF 00010000 8F C0 049C 2361 ADDL2 #^X10000,TEMP ; Bump the row counter for 256kb arrays
          50 00000000'EF 01 C3 04A7 2362 139$: SUBL3 #1,MAXROW,R0 ; Calculate the new base address
          57 50 D1 04AF 2363 142$: CMPL R0,R7 ; Adjust MAXROW
          2B 19 04B2 2364 BLSS 144$ ; Done for all rows ?
          FC30 31 04B4 2365 BRW 9$ ; branch to test end
          04B7 2366 ; Branch if not to get next array
          04B7 2367 ; Come here if uncorrectable error
          04B7 2368 ;
00000000'EF 16 04B7 2369 200$: JSB WHICH_ARRAY ; Determine failing module
00000000'EF 03 90 04BD 2370 MOVB #3,ERROR_TYPE ; Point to RAM failure

```

ZZ-ECKAM-3.2
ECKAM
03-02

TEST 10: MOVING INVERSIONS TEST
MEMORY SUBSYSTEM TESTS
TEST 10: MOVING INVERSIONS TEST

I 12

13-MAY-1987

Fiche 1 Frame I12

Sequence 151

23-JAN-1987 14:30:50

VAX/VMS Macro V04-00

Page 64

23-JAN-1987 14:29:55

[ZECCHINO.ECKAM]ECKAM1.MAR;6

(13)

```

55 54 53 CD 04C4 2371 XORL3 R3,R4,R5 ; Generate xor of expected and received A2
04C8 2372 $DS_ERRHARD_S ,#0,ARRAY,PRINT ; Report the error A2
00000000'EF DF 04C8 PUSHAL PRINT
00000000'EF DF 04CE PUSHAL ARRAY
00 DD 04D4 PUSHL #0
02 DD 04D6 PUSHL #SER
04D8 ;:: TEST 10, SUBTEST 0, ERROR 2
00000000'9F 04 FB 04D8 CALLS $$$M, a#DS$ERRHARD
04DF 2373 144$: $DS_ENDTEST
50 01 D0 04DF TEST_010_X:: MOVL #1, R0 ; NORMAL EXIT
04E2
00000000'9F 6E FA 04E2 CALLG (SP), a#DS$BREAK
04 04E9 RET ; RETURN TO TEST SEQUENCER
04EA 2374 $DS_PAGE
```

```

00000000 04EA .SBTTL TEST 11: MOVI WITH MANUAL ARRAY SELECT
00000000 0000 .PSECT TEST_011, PAGE, NOWRT
00000000 0020
00000000 0020 2377 $DS_BGNTST <MANUAL>
00000000 0020 DATA_011:
00000000 0020 .LONG 0 ; TEST ARGUMENT TABLE TERMINATOR
00000000 0024 TEST_011::
00000000 0024 .WORD ^M<> ; ENTRY MASK
00000000 0026 2378 ;++
00000000 0026 2379 ; Functional description:
00000000 0026 2380 ;
00000000 0026 2381 ; This test is identical to the previous test, except that the user may
00000000 0026 2382 ; select the arrays she desires to test.
00000000 0026 2383 ;
00000000 0026 2384 ; Test algorithm:
00000000 0026 2385 ;
00000000 0026 2386 ; 1. Write all rows of all arrays under test with all zeros.
00000000 0026 2387 ;
00000000 0026 2388 ; 2. Go to the beginning of the first row under test. Read all zeros,
00000000 0026 2389 ; write all ones, and read all ones. Continue sequentially until all
00000000 0026 2390 ; longwords in this row are filled with all ones.
00000000 0026 2391 ;
00000000 0026 2392 ; 3. Go back to the beginning of the first row under test. Read all
00000000 0026 2393 ; ones, write all zeros, and read all zeros. Continue sequentially
00000000 0026 2394 ; until all longwords in this row are filled with all zeros.
00000000 0026 2395 ;
00000000 0026 2396 ; 4. Go to the end of the first row under test. Read all zeros, write
00000000 0026 2397 ; all ones, and read all ones. Continue backwards until all
00000000 0026 2398 ; longwords in this row are filled with all ones.
00000000 0026 2399 ;
00000000 0026 2400 ; 5. Go back to the end of the first row under test. Read all ones,
00000000 0026 2401 ; write all zeros, and read all zeros. Continue backwards until all
00000000 0026 2402 ; longwords in this row are filled with all zeros.
00000000 0026 2403 ;
00000000 0026 2404 ; 6. Repeat steps 2 and 3 for 14 or 16 passes (the number of address bits
00000000 0026 2405 ; per chip) dependent on array type. For each successive pass, a
00000000 0026 2406 ; different address bit is used as the least significant bit in counting
00000000 0026 2407 ; such that every address line is toggled at the fastest possible rate.
00000000 0026 2408 ;
00000000 0026 2409 ; 7. Repeat steps 2 - 6 for data patterns which provide the check bit
00000000 0026 2410 ; memory with complementing patterns.
00000000 0026 2411 ;
00000000 0026 2412 ; 8. Repeat step 1 through 7 for all rows of all arrays under test.
00000000 0026 2413 ;--
00000000 0026 2414
00000000 0026 2415 TSTL FREE_PAGE_P ; Any memory test ?
00000000 0026 2416 BNEQ 2$ ; Branch if yes
00000000 0026 2417 $DS_PRINTB_S FMT_NO_ARRAYS ; Type no arrays to test message
00000000 002E
00000000 0034 PUSHAB FMT_NO_ARRAYS
00000000 0034 CALLS $$$N, a#DS$PRINTB
00000000 003B 2418 BRW 140$ ; Exit test
00000000 003E 2419 2$: MOVL #CONTROLLER,R0 ; Get address of memory controller A2
00000000 0045 2420 MOVL #^XE0000000,CSR0(R2) ; Cleanup the Controller
00000000 004C 2421 BISL #ERCE,CSR1(R0) ; Enable error interrupts A2 M
00000000 0054 2422 MOVL #1,EXP_UNC_ERR ; Set expected uncorr error flag A2
00000000 005B 2423 MOVAL 200$,RETURN_PC ; Set the return pc A2
00000000 0066 2424 JSB READ_MAP ; This sets up the slot config table

```

	00000000	'EF	16	006C	2425		JSB	GET_ARRAYS	:	This gets the row limits
			52	D5	0072		TSTL	R2	:	Any arrays to test?
			03	12	0074		BNEQ	5\$:	Yes, we do
			03C9	31	0076		BRW	140\$:	No, so the test is all over
	00000000	'EF	D4	0079	2429	5\$:	CLRL	TEMP1	:	Clear array message flags
57	00000000	'EF	D0	007F	2430		MOVL	LOWROW,R7	:	Initialize row counter
00000000	00000000	'EF	D0	0086	2431		MOVL	LOWADDR,TEMP	:	Initialize address pointer
	00000000	'EF	01	90	0091		MOVB	#1,SUMMARY_FLAG	:	Enable the summary section
	00000000	'EF	D4	0098	2433	3\$:	CLRL	PATCNT	:	Initialize pattern counter
	00000000	'EF	D4	009E	2434		CLRL	NUM_COR_ERR	:	Clear the single-bit error counter
55	00000000	'EF	D0	00A4	2435	10\$:	MOVL	PATCNT,R5	:	Get pattern count
00000000	00000000	'EF45	D0	00AB	2436		MOVL	MOVI_PAT_0[R5],PATTERN	:	Get first pattern for this pass
00000004	00000000	'EF45	D0	00B7	2437		MOVL	MOVI_PAT_1[R5],PATTERN	+	4 ; Get second pattern for this pass
	00000000	'EF	16	00C3	2438		JSB	WHICH_ARRAY	:	Determine which array is under test
5B	00000000	'EF	D1	00C9	2439		CMPB	L_CUR_ARRAY,R11	:	Find out if we need another array
			13	13	00D0		BEQL	22\$:	Branch if we do
5B	00000000	'EF	D0	00D2	2441		MOVL	L_CUR_ARRAY,R11	:	New array number
00000000	00000000	'EF4B	90	00D9	2442		MOVB	SLOT[R11],FLAG1	:	New array type
0D	00000000	'EF	50	E2	00E5	22\$:	BBSS	R0,TEMP1,23\$:	If array N msg hasn't been printed
					00ED				:	Then print it
	00000000	'EF	9F	00ED			\$DS_PRINTB	S FMT_ARRAY		
	00000000	'9F	01	FB	00F3		PUSHAB	FMT_ARRAY		
53	00000000	'EF	D0	00FA	2445	23\$:	CALLS	\$\$\$N, a#DS\$PRINTB		
56	00000000	'EF	D0	0101	2446		MOVL	PATTERN,R3	:	Prepare to write the whole row
00000000	'EF	FF 8F	91	0108	2447		MOVL	TEMP,R6	:	Initialize longword address
			1F	13	0110		CMPB	#-1,FLAG1	:	Check array type
					0112		BEQL	26\$:	Branch if 256kb array
	00000000	'EF	02	91	0112				:	[3.0] Check array type
			0B	13	0119		CMPB	#2,FLAG1	:	Branch if 1024kb array
59	56	000FFFFC	8F	C1	011B	2452	ADDL3	#^XFFFFC,R6,R9	:	Calculate highest address of row 1024KBkb
			0013	31	0123	2453	BRW	27\$:	Continue with flow
					0126				:	
59	56	0003FFFC	8F	C1	0126	2455	ADDL3	#^X3FFFC,R6,R9	:	Calculate highest address of row 256kb
			0008	31	012E	2456	BRW	27\$:	Continue with flow
59	56	0000FFFC	8F	C1	0131	2457	ADDL3	#^XFFFC,R6,R9	:	Calculate highest address of row 64kb
			66 53	D0	0139	2458	MOVL	R3,(R6)	:	Write the pattern
FFF7	56	04	59	F1	013C	2459	ACBL	R9,#4,R6,27\$:	Do for all longwords in the row
			59	D4	0142	2460	CLRL	R9	:	Initialize address bit counter
	00000000	'EF	94	0144	2461		CLRB	SEQUENCE	:	Indicate forward sequence
	00000000	'EF	94	014A	2462		CLRB	DATA	:	Indicate first forward sequence
			58	D4	0150	2463	CLRL	R8	:	Initialize address counter
5A	58	FFFF0000	8F	CB	0152	2464	BICL3	#^XFFFF0000,R8,R10	:	Time for supervisor call?
			07	12	015A	2465	BNEQ	38\$:	Branch if no
					015C	2466			:	
	00000000	'9F	6E	FA	015C		\$DS_BREAK			
			5A 58	D0	0163	2467	CALLG	(SP), a#DS\$BREAK		
00000000	'EF	FF 8F	91	0166	2468	38\$:	MOVL	R8,R10	:	Copy current row address
			19	13	016E	2469	CMPB	#-1,FLAG1	:	Check array type
					0170		BEQL	42\$:	Branch if 256kb arrays
	00000000	'EF	02	91	0170				:	[3.0] Check array type
			08	13	0177		CMPB	#2,FLAG1	:	Branch if 1024kb arrays
5A	12	0E	5A	F0	0179	2473	INSV	R10,#14,#18,R10	:	Copy row address into upper 16 bits
			000D	31	017E	2474	BRW	43\$:	Continue flow
					0181	2475			:	
5A	10	10	5A	F0	0181	2476	INSV	R10,#16,#16,R10	:	Copy row address into upper 16 bits
			0005	31	0186	2477	BRW	43\$:	Continue flow
5A	0E	12	5A	F0	0189	2478	INSV	R10,#18,#14,R10	:	Copy row address into upper 14 bits

00000000	5A	5A	59	9C	018E	2479	43\$:	ROTL	R9,R10,R10	:	Rotate R9 times	
	'EF	FF	8F	91	0192	2480		CMPB	#-1,FLAG1	:	Check array type	
			19	13	019A	2481		BEQL	46\$:	Branch if 256kb arrays	
					019C	2482	:					
00000000	'EF		02	91	019C	2483		CMPB	#2,FLAG1	:	[3.0] Check array type	
			08	13	01A3	2484		BEQL	45\$:	Branch if 1024kb arrays	
50	5A	12	00	EF	01A5	2485		EXTZV	#0,#18,R10,R0	:	Calculate actual 4Mb arrays	
			000D	31	01AA	2486		BRW	48\$:	Continue flow	
					01AD	2487	:					
50	5A	10	00	EF	01AD	2488	45\$:	EXTZV	#0,#16,R10,R0	:	Calculate actual	
			0005	31	01B2	2489		BRW	48\$:	Continue flow	
50	5A	0E	00	EF	01B5	2490	46\$:	EXTZV	#0,#14,R10,R0	:	Calculate actual 256kb arrays	
	50	50	02	78	01BA	2491	48\$:	ASHL	#2,R0,R0	:	Obtain longword address	
56	00000000	'EF		D0	01BE	2492		MOVL	TEMP,R6	:	Initialize address pointer	
			56	C0	01C5	2493		ADDL	R0,R6	:	Address to use	
			66	D5	01C8	2494		TSTL	(R6)	:	Read the location under test	M2
	00000000	'EF		95	01CA	2495		TSTB	ERR_LOG_FULL	:	Is error log full? [3.1]	
			03	13	01D0	2496		BEQL	50\$			
			01CF	31	01D2	2497		BRW	133\$			
09	00000000	'EF		E8	01D5	2498	50\$:	BLBS	DATA,55\$:	If this is 0 -> 1 then	
53	00000004	'EF		D0	01DC	2499		MOVL	PATTERN+4,R3	:	Use the second pattern	
			07	11	01E3	2500		BRB	57\$:	And branch	
53	00000000	'EF		D0	01E5	2501	55\$:	MOVL	PATTERN,R3	:	Otherwise, use the first pattern	
			66	D0	01EC	2502	57\$:	MOVL	R3,(R6)	:	Write the pattern	
			66	D5	01EF	2503		TSTL	(R6)	:	Read the data	M2
	00000000	'EF		95	01F1	2504		TSTB	ERR_LOG_FULL	:	Is error log full? [3.1]	
			03	13	01F7	2505		BEQL	58\$			
			01A8	31	01F9	2506		BRW	133\$			
00000000	'EF	FF	8F	91	01FC	2507	58\$:	CMPB	#-1,FLAG1	:	Check array type	
			23	13	0204	2508		BEQL	61\$:	Branch if 256kb arrays	
					0206	2509	:					
00000000	'EF		02	91	0206	2510		CMPB	#2,FLAG1	:	Check array type	
			0D	13	020D	2511		BEQL	60\$:	Branch if 1024kb arrays	
FF39	58	01	00040000	8F	F1	020F	2512	ACBL	#262144,#1,R8,35\$:	Do for all addresses in the row 1024kb	
			0017	31	0219	2513		BRW	62\$:	Continue flow	
					021C	2514	:					
FF2C	58	01	0000FFFF	8F	F1	021C	2515	60\$:	ACBL	#65535,#1,R8,35\$:	Do for all addresses in the row 256kb
			000A	31	0226	2516		BRW	62\$:	Continue flow	
FF1F	58	01	00003FFF	8F	F1	0229	2517	61\$:	ACBL	#16383,#1,R8,35\$:	Do for all addresses in the row 64kb
					0233	2518	62\$:	\$DS_BREAK		:	Check for CTRL C	A7
00000000	'9F		6E	FA	0233			CALLG	(SP),a#DS\$BREAK			
	00000000	'EF		95	023A	2519		TSTB	DATA	:	Done for 1's and 0's both?	
			0A	12	0240	2520		BNEQ	70\$:	Branch if so	
00000000	'EF		01	90	0242	2521		MOVB	#1,DATA	:	Set data flag	
			FF04	31	0249	2522		BRW	30\$:	And branch to do forward seq again	
	00000000	'EF		94	024C	2523	70\$:	CLRB	DATA	:	Clear data flag	
00000000	'EF	FF	8F	91	0252	2524	75\$:	CMPB	#-1,FLAG1	:	Check array type	
			1D	13	025A	2525		BEQL	85\$:	Branch if 256kb arrays	
					025C	2526	:					
00000000	'EF		02	91	025C	2527		CMPB	#2,FLAG1	:	[3.0] Check array type	
			0A	13	0263	2528		BEQL	80\$:	Branch if 1024kb arrays	
58	00040000	8F		D0	0265	2529		MOVL	#262144,R8	:	Initialize address counter 4Mb array	
			0011	31	026C	2530		BRW	88\$:	Continue flow	
					026F	2531	:					
58	0000FFFF	8F		D0	026F	2532	80\$:	MOVL	#65535,R8	:	Initialize address counter 1024kb array	
			0018	31	0276	2533		BRW	90\$:	Continue flow	
58	00003FFF	8F		D0	0279	2534	85\$:	MOVL	#16383,R8	:	Initialize address counter 256kb array	

SA	58	FFFF8000	8F	CB	0280	2535	88\$:	BICL3	#AXFFFF8000,R8,R10	; Time for supervisor call?		
			07	12	0288	2536		BNEQ	90\$; Branch if no		
					028A	2537		\$DS_BREAK				
		00000000	'9F	6E	FA	028A		CALLG	(SP), a#DS\$BREAK			
			5A	58	D0	0291	2538	90\$:	MOVL	R8,R10	; Copy current row address	
		00000000	'EF	FF	8F	91	0294	2539	CMPB	#-1,FLAG1	; Check array type	
				19	13	029C	2540		BEQL	92\$; Branch if 256kb arrays	
						029E	2541					
		00000000	'EF	02	91	029E	2542		CMPB	#2,FLAG1	; [3.0] Check array type	
				08	13	02A5	2543		BEQL	91\$; Branch if 256kb arrays	
		5A	12	0E	5A	F0	02A7	2544	INSV	R10,#14,#18,R10	; Copy row address into upper 18 bits	
				000D	31	02AC	2545		BRW	93\$; Continue flow	
						02AF	2546					
		5A	10	10	5A	F0	02AF	2547	91\$:	INSV	R10,#16,#16,R10	; Copy row address into upper 16 bits
						31	02B4	2548		BRW	93\$; Continue flow
		5A	0E	12	5A	F0	02B7	2549	92\$:	INSV	R10,#18,#14,R10	; Copy row address into upper 14 bits
			5A	5A	59	9C	02BC	2550	93\$:	ROTL	R9,R10,R10	; Rotate R9 times
		00000000	'EF	FF	8F	91	02C0	2551		CMPB	#-1,FLAG1	; Check array type
				19	13	02C8	2552		BEQL	96\$; Branch if 256kb arrays	
						02CA	2553					
		00000000	'EF	02	91	02CA	2554		CMPB	#2,FLAG1	; [3.0] Check array type	
				08	13	02D1	2555		BEQL	94\$; Branch if 1024kb arrays	
		50	5A	12	00	EF	02D3	2556		EXTZV	#0,#18,R10,R0	; Calculate actual 4Mb array
				000D	31	02D8	2557		BRW	98\$; Continue flow	
						02DB	2558					
		50	5A	10	00	EF	02DB	2559	94\$:	EXTZV	#0,#16,R10,R0	; Calculate actual
						31	02E0	2560		BRW	98\$; Continue flow
		50	5A	0E	00	EF	02E3	2561	96\$:	EXTZV	#0,#14,R10,R0	; Calculate actual for 256kb arrays
			50	50	02	78	02E8	2562	98\$:	ASHL	#2,R0,R0	; Obtain longword address
		56	00000000	'EF	D0	02EC	2563		MOVL	TEMP,R6	; Initialize address pointer	
				56	50	C0	02F3	2564		ADDL	R0,R6	; Address to use
						D5	02F6	2565		TSTL	(R6)	; Read the location under test
			00000000	'EF	95	02F8	2566		TSTB	ERR_LOG_FULL	; Is error log full? [3.1]	
				03	13	02FE	2567		BEQL	100\$		
				00A1	31	0300	2568		BRW	133\$		
		09	00000000	'EF	E8	0303	2569	100\$:	BLBS	DATA,105\$; If this is 0 -> 1 then	
		53	00000004	'EF	D0	030A	2570		MOVL	PATTERN+4,R3	; Use the second pattern	
				07	11	0311	2571		BRB	107\$; And branch	
		53	00000000	'EF	D0	0313	2572	105\$:	MOVL	PATTERN,R3	; Otherwise, use the first pattern	
				66	53	D0	031A	2573	107\$:	MOVL	R3,(R6)	; Write the pattern
						D5	031D	2574		TSTL	(R6)	; Read the data
			00000000	'EF	95	031F	2575		TSTB	ERR_LOG_FULL	; Is error log full? [3.1]	
				03	13	0325	2576		BEQL	110\$		
				007A	31	0327	2577		BRW	133\$		
				58	D7	032A	2578	110\$:	DECL	R8	; Subtract one from address counter	
				03	19	032C	2579		BLSS	120\$; Branch if done for all addresses	
				FF4F	31	032E	2580		BRW	88\$; Branch to do next address	
						0331	2581	120\$:	\$DS_BREAK		; Check for CTRL C	
		00000000	'9F	6E	FA	0331			CALLG	(SP), a#DS\$BREAK		
			00000000	'EF	95	0338	2582		TSTB	DATA	; Done for 1's and 0's both?	
				0A	12	033E	2583		BNEQ	125\$; Branch if so	
		00000000	'EF	01	90	0340	2584		MOVB	#1,DATA	; Set data flag	
				FF08	31	0347	2585		BRW	75\$; Branch to do reverse seq again	
			00000000	'EF	94	034A	2586	125\$:	CLRB	DATA	; Clear data flag	
		00000000	'EF	FF	8F	91	0350	2587		CMPB	#-1,FLAG1	; Check array type
				1B	13	0358	2588		BEQL	131\$; Branch if 256kb arrays	
						035A	2589					

```
00000000'EF 02 91 035A 2590 CMPB #2,FLAG1 ;[3.0] Check array type
09 13 0361 2591 BEQL 128$ ; Branch if 1024kb arrays
FDE7 59 01 11 F1 0363 2592 ACBL #17,#1,R9,30$ ; Do for all address bits in address
000F 31 0369 2593 BRW 132$ ; Continue flow
036C 2594 ;
FDDE 59 01 0F F1 036C 2595 128$: ACBL #15,#1,R9,30$ ; Do for all address bits in address
0006 31 0372 2596 BRW 132$ ; Continue flow
FDD5 59 01 0D F1 0375 2597 131$: ACBL #13,#1,R9,30$ ; Do for all address bits in address
FD1F 00000000'EF 01 01 F1 037B 2598 132$: ACBL #1,#1,PATCNT,10$ ; Do for data and check bit patterns
50 57 D0 0385 2599 MOVL R7,R0 ; Copy row number
51 D4 0388 2600 CLRL R1 ; Clear high part of quadword for EDIV
5A 52 50 04 7B 038A 2601 EDIV #4,R0,R2,R10 ; Calculate row # of array under test
038F 2602 $DS_PRINTB_S FMT_ROW_MSG,- ; Print the row number under test
038F 2603 R10,NUM_COR_ERR ; single bit errors for this row
00000000'EF DD 038F PUSHL NUM_COR_ERR
5A DD 0395 PUSHL R10
00000000'EF 9F 0397 PUSHAB FMT_ROW_MSG
00000000'9F 03 FB 039D CALLS $$$N, a#DS$PRINTB
03A4 2604 ;
00000000'EF 95 03A4 2605 133$: TSTB ERR_LOG_FULL ; If error log is full then
10 13 03AA 2606 BEQL 134$ ; end test [3.1]
03AC 2607 $DS_PRINTB_S FMT_LOG_FULL
00000000'EF 9F 03AC PUSHAB FMT_LOG_FULL
00000000'9F 01 FB 03B2 CALLS $$$N, a#DS$PRINTB
03B9 2608 $DS_ABORT TEST
50 D4 03B9 CLRL R0 ; SEND A WARNING TO THE SUPERVISOR
04 03BB RET ; TERMINATE TEST
03BC 2609 ;
00000000'EF FF 8F 91 03BC 2610 134$: CMPB #-1,FLAG1 ; Check array type
3B 13 03C4 2611 BEQL 137$ ; Branch for 256kb arrays
03C6 2612 ;
00000000'EF 02 91 03C6 2613 CMPB #2,FLAG1 ;[3.0] Check array type
19 13 03CD 2614 BEQL 135$ ; Branch for 1024kb arrays
57 D6 03CF 2615 INCL R7 ; Bump the row counter 4Mb arrays
00000000'EF 00100000 8F C0 03D1 2616 ADDL2 #^X100000,TEMP ; Calculate the new base address
57 00000000'EF D1 03DC 2617 CMPL HIGHROW,R7 ; Done for all rows under test? (256kb)
5D 19 03E3 2618 BLSS 140$ ; Branch to end test
FCB0 31 03E5 2619 BRW 3$ ; Branch to get next array
03E8 2620 ;
00000000'EF 57 D6 03E8 2621 135$: INCL R7 ; Bump the row counter 1024k arrays
00040000 8F C0 03EA 2622 ADDL2 #^X40000,TEMP ; Calculate the new base address
57 00000000'EF D1 03F5 2623 CMPL HIGHROW,R7 ; Done for all rows under test? (256kb)
44 19 03FC 2624 BLSS 140$ ; Branch to end test
FC97 31 03FE 2625 BRW 3$ ; Branch to get next array
57 D6 0401 2626 137$: INCL R7 ; Bump the row counter for 256kb arrays
00000000'EF 00010000 8F C0 0403 2627 ADDL2 #^X10000,TEMP ; Calculate the new base address
57 00000000'EF D1 040E 2628 CMPL HIGHROW,R7 ; Done for all rows under test?(64kb)
2B 19 0415 2629 BLSS 140$ ; Branch to end test
FC7E 31 0417 2630 BRW 3$ ; Branch to get next array
041A 2631 ;
041A 2632 ; Come here if uncorrectable error
041A 2633 ;
00000000'EF 16 041A 2634 200$: JSB WHICH_ARRAY ; Determine failing module
00000000'EF 03 90 0420 2635 MOVB #3,ERROR_TYPE ; Point to RAM failure
55 54 53 CD 0427 2636 XORL3 R3,R4,R5 ; Generate xor of expected and received
042B 2637 $DS_ERRHARD_S #0,ARRAY,PRINT ; Report the error
00000000'EF DF 042B PUSHAL PRINT
```

ZZ-ECKAM-3.2
ECKAM
03-02

TEST 11: MOVI WITH MANUAL ARRAY SELECT

MEMORY SUBSYSTEM TESTS

TEST 11: MOVI WITH MANUAL ARRAY SELECT

B 13

13-MAY-1987

Fiche 1 Frame B13

Sequence 157

23-JAN-1987 14:30:50

VAX/VMS Macro V04-00

Page 70

23-JAN-1987 14:29:55

[ZECCHINO.ECKAM]ECKAM1.MAR;6

(14)

```
00000000'EF DF 0431          PUSHAL ARRAY
                00 DD 0437          PUSHL #0
                01 DD 0439          PUSHL #SER
00000000'9F 04 FB 043B          ;:: TEST 11, SUBTEST 0, ERROR 1
                043B          CALLS #$$M, @#DS$ERRHARD
                0442 2638 140$: $DS_ENDTEST
                50 01 D0 0442          MOVL #1, R0          ; NORMAL EXIT
                0445          TEST_011_X::
00000000'9F 6E FA 0445          CALLG (SP), @#DS$BREAK
                04 044C          RET          ; RETURN TO TEST SEQUENCER
                044D 2639          $DS_PAGE
```

```
044D                    .SBTTL TEST 12: MEMORY QUICK VERIFY TEST
00000000                .PSECT TEST_012, PAGE, NOWRT
001C                    2642            $DS_BGNTST            <DEFAULT,ALL,EXHAUSTIVE>
001C                    DATA_012:                            .LONG    0                    ; TEST ARGUMENT TABLE TERMINATOR
00000000                TEST_012::                            .WORD    ^M<>                    ; ENTRY MASK
0000                    0020                                    2$                    ; If Quick Flag is set continue
08    E0                0022            $DS_BQUICK            BBS                    #DSA$V_QUICK,-            ; BR IF QUICK
03 0000FE00 9F            0022                                    a#DSA$GL_FLAGS, 2$            ;
50    D4                002A            $DS_ABORT            TEST                    ; Otherwise abort test
                          002A                                    CLRL                    R0                    ; SEND A WARNING TO THE SUPERVISOR
                          002C                                    RET                    ; TERMINATE TEST
002D                    2645 ;++
002D                    2646 ; Functional description:
002D                    2647 ;
002D                    2648 ;                    In this test the arrays are tested for any gross errors. A memory
002D                    2649 ;                    march algorithm is used in this test, providing a quick check of
002D                    2650 ;                    all the RAMs. Any gross errors will be detected. Set the Quick Flag
002D                    2651 ;                    to invoke this test.
002D                    2652 ;
002D                    2653 ; Note: Not all rows of all arrays will be tested. Only thoes rows not protected
002D                    2654 ;                    by the Diagnostic Supervisor. If in doubt of the reliability of this
002D                    2655 ;                    portion of memory, a manual swap of the array cards, to a higher slot
002D                    2656 ;                    should be made. The first printout of the array under test, (tests 10,
002D                    2657 ;                    11,12) will tell you where testing had begun.
002D                    2658 ;
002D                    2659 ; Test algorithm:
002D                    2660 ;
002D                    2661 ;                    1. Write all rows of all arrays under test with all zeros. One row at
002D                    2662 ;                    a time. (A longword at a time, row size dependent on array type
002D                    2663 ;                    either 64/256kb boundarys).
002D                    2664 ;
002D                    2665 ;                    2. Go to the beginning of the first row under test. Read all zeros,
002D                    2666 ;                    write all ones, and read all ones. Continue sequentially until all
002D                    2667 ;                    longwords in this row are filled with all ones.
002D                    2668 ;
002D                    2669 ;                    3. Go back to the beginning of the first row under test. Read all
002D                    2670 ;                    ones, write all zeros, and read all zeros. Continue sequentially
002D                    2671 ;                    until all longwords in this row are filled with all zeros.
002D                    2672 ;
002D                    2673 ;                    4. Go to the end of the first row under test. Read all zeros, write
002D                    2674 ;                    all ones, and read all ones. Continue backwards until all
002D                    2675 ;                    longwords in this row are filled with all ones.
002D                    2676 ;
002D                    2677 ;                    5. Go back to the end of the first row under test. Read all ones,
002D                    2678 ;                    write all zeros, and read all zeros. Continue backwards until all
002D                    2679 ;                    longwords in this row are filled with all zeros.
002D                    2680 ;
002D                    2681 ;                    6. Repeat steps 2 - 5 for data patterns which provide the check bit
002D                    2682 ;                    memory with complementing patterns.
002D                    2683 ;
002D                    2684 ;                    7. Repeat step 1 through 7 for all rows of all arrays under test.
002D                    2685 ;--
00000000 *EF    D5    002D            2686
                          002D            2687 2$:            TSTL    FREE_PAGE_P                    ; Any memory to test?
```



```

59 56 000FFFFC 0B 13 0168 2741 BEQL 25$ ; Branch if 1024kb array
      8F C1 016A 2742 ADDL3 #^XFFFFC,R6,R9 ; Calculate highest address of row 1024bk
      0013 31 0172 2743 BRW 27$ ; Continue with flow
      0175 2744 ;
59 56 0003FFFC 8F C1 0175 2745 25$: ADDL3 #^X3FFFC,R6,R9 ; Calculate highest address of row 256bk
      0008 31 017D 2746 BRW 27$ ; Continue with flow
59 56 0000FFFC 8F C1 0180 2747 26$: ADDL3 #^XFFFC,R6,R9 ; Calculate highest address of row 64kb
      66 53 D0 0188 2748 27$: MOVL R3,(R6) ; Write the pattern
      FFF7 56 04 59 F1 018B 2749 ACBL R9,#4,R6,27$ ; Do for all longwords in the row
      0191 2750 $DS_BREAK ; Check for control C
      00000000'9F 6E FA 0191 CALLG (SP), a#DS$BREAK
56 00000000'EF D0 0198 2751 28$: MOVL TEMP,R6 ; Initialize address
      66 D5 019F 2752 30$: TSTL (R6) ; Read the data
      00000000'EF 95 01A1 2753 TSTB ERR_LOG_FULL ; Is error log full? [3.1]
      03 13 01A7 2754 BEQL 33$
      00DE 31 01A9 2755 BRW 131$
      0A 00000000'EF E8 01AC 2756 33$: BLBS DATA,35$ ; If this is 0 ->1 then
53 00000004'EF D0 01B3 2757 MOVL PATTERN+4,R3 ; Use the second pattern
      0007 31 01BA 2758 BRW 36$ ; Continue....
53 00000000'EF D0 01BD 2759 35$: MOVL PATTERN,R3 ; Otherwise use the first pattern
      66 53 D0 01C4 2760 36$: MOVL R3,(R6) ; Write the pattern
      66 D5 01C7 2761 TSTL (R6) ; Read the data
      00000000'EF 95 01C9 2762 TSTB ERR_LOG_FULL ; Is error log full? [3.1]
      03 13 01CF 2763 BEQL 40$
      00B6 31 01D1 2764 BRW 131$
      FFC5 56 04 59 F1 01D4 2765 40$: ACBL R9,#4,R6,30$ ; Do for all the longwords in the row
      01DA 2766 $DS_BREAK ; Check for CTRL C
      00000000'9F 6E FA 01DA CALLG (SP), a#DS$BREAK
      00000000'EF 95 01E1 2767 TSTB DATA ; Done for 1's and 0's both ?
      0A 12 01E7 2768 BNEQ 70$ ; Branch if so
      00000000'EF 01 90 01E9 2769 MOVB #1,DATA ; Set data flag
      FFA5 31 01F0 2770 BRW 28$ ; Go do again with patterns in reverse
      01F3 2771 ;
      01F3 2772 ;THIS IS WHERE THE BACKWORDS SEQUENCE DOWN THROUGH MEMORY BEGINS
      01F3 2773 ;
      00000000'EF 94 01F3 2774 70$: CLRB DATA ; Initilize data flag
      56 04 C2 01F9 2775 SUBL2 #4,R6 ; Decrement address within range
      66 D5 01FC 2776 72$: TSTL (R6) ; Read data
      00000000'EF 95 01FE 2777 TSTB ERR_LOG_FULL ; Is error log full? [3.1]
      03 13 0204 2778 BEQL 80$
      0081 31 0206 2779 BRW 131$
      0A 00000000'EF E8 0209 2780 80$: BLBS DATA,105$ ; Check data flag for pattern
53 00000004'EF D0 0210 2781 MOVL PATTERN+4,R3 ; Use the second pattern
      0007 31 0217 2782 BRW 107$ ; Continue....
53 00000000'EF D0 021A 2783 105$: MOVL PATTERN,R3 ; Otherwise use the first pattern
      66 53 D0 0221 2784 107$: MOVL R3,(R6) ; Write pattern
      66 D5 0224 2785 TSTL (R6) ; Read the data
      00000000'EF 95 0226 2786 TSTB ERR_LOG_FULL ; Is error log full? [3.1]
      03 13 022C 2787 BEQL 110$
      0059 31 022E 2788 BRW 131$
56 FFFFFFFC 8F 00000000'EF F1 0231 2789 110$: ACBL TEMP,#-4,R6,72$ ; Decrement the address
      FFBD 023D $DS_BREAK ; Check for CTRL C
      023F 2790 CALLG (SP), a#DS$BREAK
      00000000'9F 6E FA 023F MOVL R9,R6 ; Set up max row address again
      56 59 D0 0246 2791 TSTB DATA ; Done for 1's and 0's ?
      00000000'EF 95 0249 2792 BNEQ 130$ ; Branch if yes
      0A 12 024F 2793

```

```

00000000'EF 01 90 0251 2794      MOVB   #1,DATA      ; Set flag for reverse pattern
                FFA1 31 0258 2795      BRW    72$          ; Go do again with reverse pattern
00000000'EF 94 025B 2796 130$:    CLRB   DATA        ; Clear data flag
FE82 00000000'EF 01 01 F1 0261 2797      ACBL   #1,#1,PATCNT,10$ ; Do for data and check bit patterns
                50 57 D0 026B 2798      MOVL   R7,R0        ; Copy row number
                51 51 D4 026E 2799      CLRL   R1           ; Clear high part of quadword for EDIV
SA 52 50 04 7B 0270 2800      EDIV   #4,R0,R2,R10 ; Calculate row # of array under test
                0275 2801      $DS_PRINTB_S FMT_ROW_MSG,- ; Print the row number under test
                0275 2802      R10,NUM_COR_ERR ; single bit errors for this row
                DD 0275      PUSHL   NUM_COR_ERR
                5A DD 027B      PUSHL   R10
                00000000'EF 9F 027D      PUSHAB FMT_ROW_MSG
00000000'9F 03 FB 0283      CALLS  $$$N, a#DS$PRINTB
                028A 2803 ;
                028A 2804 ; If running under manufacturing APT and there are single bit errors,
                028A 2805 ; make an ERRHARD call.
                028A 2806 ;
0000FE00'EF 80000000 8F D3 028A 2807 131$: BITL   #DSA$M_APT,DSA$GL_FLAGS ; APT mode ?
                3E 13 0295 2808      BEQL   132$         ; Branch if no
                00000000'EF 95 0297 2809      TSTB   NUM_COR_ERR ; Any single bit errors ?
                4E 13 029D 2810      BEQL   135$         ; Branch if no
SA 57 FFFFFFFC 8F CB 029F 2811      BICL3  #^FFFFFFFC,R7,R10 ; Compute current array [3.1]
                52 57 FE 8F 78 02A7 2812      ASHL   #-2,R7,R2 ; and row from rows done
                02AC 2813      $DS_ERRHARD_S #0,-
                02AC 2814      ARRAY_SGL_BIT,-
                02AC 2815      PRINT_GENERAL,-
                02AC 2816      P1 = #4,-
                02AC 2817      P2 = #FMT_SGL_BIT_ERR,-
                02AC 2818      P3 = R2,-
                02AC 2819      P4 = R10,-
                02AC 2820      P5 = NUM_COR_ERR; Report the error
                DD 02AC      PUSHL   NUM_COR_ERR
                5A DD 02B2      PUSHL   R10
                52 DD 02B4      PUSHL   R2
00000000'8F 04 DD 02B6      PUSHL   #FMT_SGL_BIT_ERR
                04 DD 02BC      PUSHL   #4
00000000'EF DF 02BE      PUSHAL PRINT_GENERAL
00000000'EF DF 02C4      PUSHAL ARRAY_SGL_BIT
                00 DD 02CA      PUSHL   #0
                01 DD 02CC      PUSHL   #SER
                02CE      ::: TEST 12, SUBTEST 0, ERROR 1
00000000'9F 09 FB 02CE      CALLS  $$$M, a#DS$ERRHARD
                02D5 2821 ;
00000000'EF 95 02D5 2822 132$: TSTB   ERR_LOG_FULL ; If error log is full then
                10 13 02DB 2823      BEQL   135$         ; end test [3.1]
                02DD 2824      $DS_PRINTB_S FMT_LOG_FULL
                00000000'EF 9F 02DD      PUSHAB FMT_LOG_FULL
00000000'9F 01 FB 02E3      CALLS  $$$N, a#DS$PRINTB
                50 D4 02EA 2825      $DS_ABORT TEST
                04 02EA      CLRL   R0 ; SEND A WARNING TO THE SUPERVISOR
                00000000'EF FF 8F 91 02ED 2826 135$: CMPB   #-1,FLAG1 ; TERMINATE TEST
                29 13 02F5 2827      BEQL   137$         ; Check array type
                02F7 2828 ; ; Branch for 256kb arrays
00000000'EF 02 91 02F7 2829      CMPB   #2,FLAG1 ; [3.0] Check array type
                10 13 02FE 2830      BEQL   136$         ; Branch for 1024kb arrays
                57 D6 0300 2831      INCL   R7 ; Bump the row counter 4Mb arrays

```

```
00000000'EF 00100000 8F C0 0302 2832 ADDL2 #^X100000,TEMP ; Calculate the new base address
                001D 31 030D 2833 BRW 139$ ; Continue flow
                57 D6 0310 2834 ;
00000000'EF 00040000 8F C0 0312 2835 136$: INCL R7 ; Bump the row counter 1024kb arrays
                000D 31 031D 2836 ADDL2 #^X40000,TEMP ; Calculate the new base address
                57 D6 0320 2837 BRW 139$ ; Continue flow
00000000'EF 00010000 8F C0 0322 2838 137$: INCL R7 ; Bump the row counter for 256kb arrays
50 00000000'EF 01 C3 032D 2839 ADDL2 #^X10000,TEMP ; Calculate the new base address
    57 50 D1 0335 2840 139$: SUBL3 #1,MAXROW,R0 ; Adjust MAXROW
    2B 19 0338 2841 142$: CMPL R0,R7 ; Done for all rows ?
    FDA4 31 033A 2842 BLSS 144$ ; branch to test end
                033D 2843 BRW 9$ ; Branch if not to get next array
                033D 2844 ;
                033D 2845 ; Come here if uncorrectable error
                033D 2846 ;
00000000'EF 16 033D 2847 200$: JSB WHICH_ARRAY ; Determine failing module A2
00000000'EF 03 90 0343 2848 MOVB #3,ERROR_TYPE ; Point to RAM failure A2
55 54 53 CD 034A 2849 XORL3 R3,R4,R5 ; Generate xor of expected and received A2
                034E 2850 $DS_ERRHARD_S ,#0,ARRAY,PRINT ; Report the error A2
00000000'EF DF 034E PUSHAL PRINT
00000000'EF DF 0354 PUSHAL ARRAY
00 DD 035A PUSHL #0
02 DD 035C PUSHL #SER
                035E ;;; TEST 12, SUBTEST 0, ERROR 2
00000000'9F 04 FB 035E CALLS $$$M, a#DS$ERRHARD
    50 01 D0 0365 2851 144$: $DS_ENDTEST
                0368 TEST_012_X:: MOVL #1, R0 ; NORMAL EXIT
00000000'9F 6E FA 0368 CALLG (SP), a#DS$BREAK
    04 036F RET ; RETURN TO TEST SEQUENCER
                0370 2852 $DS_ENDMOD
                00000000 .PSECT $TSTCNT, NOEXE, NOWRT, OVR, LONG
00000000C 0000 .LONG $TN
                0004 2853 .END
```

\$\$\$	=	00000000		
\$\$E	=	00000001		
\$\$M	=	00000004		
\$\$N	=	00000000		
\$\$\$	=	FFFFFFFF		
\$ENV	=	00000001	G	
\$ER	=	FFFFFFFF		
\$MO	=	00000001	G	
\$\$\$	=	000000FC	R	08
\$ST	=	00000000		
\$TN	=	0000000C		
ALL	=	00000001	G	
ALLMOD		*****	X	02
ARRAY		*****	X	0D
ARRAY_SGL_BIT		*****	X	0D
BASE_001		00000000	R	02
BASE_002		00000000	R	04
BASE_003		00000000	R	05
BASE_004		00000000	R	06
BASE_005		00000000	R	07
BASE_006		00000000	R	09
BASE_007		00000000	R	0A
BASE_008		00000000	R	0B
BASE_009		00000000	R	0C
BASE_010		00000000	R	0D
BASE_011		00000000	R	0E
BASE_012		00000000	R	0F
BIT...	=	00000002		
BIT0	=	00000001		
BIT1	=	00000002		
BIT10	=	00000400		
BIT11	=	00000800		
BIT12	=	00001000		
BIT13	=	00002000		
BIT14	=	00004000		
BIT15	=	00008000		
BIT16	=	00010000		
BIT17	=	00020000		
BIT18	=	00040000		
BIT19	=	00080000		
BIT2	=	00000004		
BIT20	=	00100000		
BIT21	=	00200000		
BIT22	=	00400000		
BIT23	=	00800000		
BIT24	=	01000000		
BIT25	=	02000000		
BIT26	=	04000000		
BIT27	=	08000000		
BIT28	=	10000000		
BIT29	=	20000000		
BIT3	=	00000008		
BIT30	=	40000000		
BIT31	=	80000000		
BIT4	=	00000010		
BIT5	=	00000020		
BIT6	=	00000040		

BIT7	=	00000080		
BIT8	=	00000100		
BIT9	=	00000200		
BUS_ERR_REG		*****	X	07
CE	=	20000000		
CEP_FUNCTIONAL	=	00000000		
CEP_REPAIR	=	00000001		
CK		*****	X	07
CKBIT		*****	X	07
CODEWORDS		*****	X	05
CONTROL		*****	X	07
CONTROLLER	=	00F20000		
CORRUPT_TEST_LOC		*****	X	07
CPU		*****	X	07
CSR0	=	00000000		
CSR1	=	00000004		
CSR2	=	00000008		
DATA		*****	X	0D
DATA_001		00000020	R	02
DATA_002		00000010	R	04
DATA_003		00000018	R	05
DATA_004		00000014	R	06
DATA_005		00000014	R	07
DATA_006		0000000C	R	09
DATA_007		00000018	R	0A
DATA_008		00000018	R	0B
DATA_009		00000018	R	0C
DATA_010		0000001C	R	0D
DATA_011		00000020	R	0E
DATA_012		0000001C	R	0F
DCM	=	04000000		
DECODED_WORDS		*****	X	05
DECODER		*****	X	05
DEFAULT	=	00000000	G	
DEVICE_TYPE		*****	X	0A
DIGIT1		*****	X	0D
DS\$BGNSUB		*****	X	07
DS\$BREAK		*****	X	02
DS\$CKLOOP		*****	X	02
DS\$ENDSUB		*****	X	07
DS\$ERRHARD		*****	X	02
DS\$K_BBC	=	0000001D	G	
DS\$K_BBCC	=	00000021	G	
DS\$K_BBCCI	=	00000023	G	
DS\$K_BBCS	=	0000001F	G	
DS\$K_BBS	=	0000001C	G	
DS\$K_BBSC	=	00000020	G	
DS\$K_BBSS	=	0000001E	G	
DS\$K_BBSSI	=	00000022	G	
DS\$K_BCC_M	=	0000001B	G	
DS\$K_BCS_M	=	0000001A	G	
DS\$K_BEQLU_B	=	00000005	G	
DS\$K_BEQLU_M	=	00000011	G	
DS\$K_BEQL_B	=	00000004	G	
DS\$K_BEQL_M	=	00000010	G	
DS\$K_BERROR	=	00000026	G	
DS\$K_BGEQU_B	=	00000007	G	

DS\$K_BGEQU_M	= 00000013	G	
DS\$K_BGEQ_B	= 00000006	G	
DS\$K_BGEQ_M	= 00000012	G	
DS\$K_BGTRU_B	= 00000009	G	
DS\$K_BGTRU_M	= 00000015	G	
DS\$K_BGTR_B	= 00000008	G	
DS\$K_BGTR_M	= 00000014	G	
DS\$K_BLBC	= 00000025	G	
DS\$K_BLBS	= 00000024	G	
DS\$K_BLEQU_B	= 00000003	G	
DS\$K_BLEQU_M	= 0000000F	G	
DS\$K_BLEQ_B	= 00000002	G	
DS\$K_BLEQ_M	= 0000000E	G	
DS\$K_BLSSU_B	= 00000001	G	
DS\$K_BLSSU_M	= 0000000D	G	
DS\$K_BLSS_B	= 00000000	G	
DS\$K_BLSS_M	= 0000000C	G	
DS\$K_BNEQU_B	= 0000000B	G	
DS\$K_BNEQU_M	= 00000017	G	
DS\$K_BNEQ_B	= 0000000A	G	
DS\$K_BNEQ_M	= 00000016	G	
DS\$K_BNERROR	= 00000027	G	
DS\$K_BVC_M	= 00000019	G	
DS\$K_BVS_M	= 00000018	G	
DS\$PRINTB	*****	X	02
DS\$PRINTX	*****	X	04
DSA\$AL_APTMAIL	0000FE00		
DSA\$AT_APTTXT	0000FA00		
DSA\$GL_APTCOM	0000FE04		
DSA\$GL_DEVLEN	0000FE58		
DSA\$GL_ERRNO	0000FE44		
DSA\$GL_EVENT	0000FE48		
DSA\$GL_FLAGS	0000FE00		
DSA\$GL_MSGTYP	0000FE40		
DSA\$GL_PASSES	0000FE08		
DSA\$GL_PASSNO	0000FE54		
DSA\$GL_SECTNO	0000FE10		
DSA\$GL_SID	0000FE14		
DSA\$GL_SUBTNO	0000FE4C		
DSA\$GL_TESTNO	0000FE50		
DSA\$GL_UNITS	0000FE0C		
DSA\$GQ_MSGPTR	0000FE68		
DSA\$GT_DEVNAM	0000FE5C		
DSA\$M_APT	= 80000000		
DSA\$V_QUICK	= 00000008		
EDM	= 02000000		
ENV\$M_DOMAIN	= 00000002		
ENV\$M_LEVEL	= 00000001		
ENV\$M_SUPER	= 000003FC		
ENV\$S_DOMAIN	= 00000001		
ENV\$S_LEVEL	= 00000001		
ENV\$S_SUPER	= 00000008		
ENV\$V_DOMAIN	= 00000001		
ENV\$V_LEVEL	= 00000000		
ENV\$V_SUPER	= 00000002		
ENV\$CPU	= 00000000		
ENV\$FUNCTIONAL	= 00000000		

ENV\$REPAIR	= 00000001		
ENV\$SUPER	= 00000001		
ENV\$SYSTEM	= 00000001		
ERCE	= 10000000		
ERROR_TYPE	*****	X	05
ERR_LOG_FULL	*****	X	0D
EXHAUSTIVE	= 00000002	G	
EXP_COR_ERR	*****	X	07
EXP_NX_MEM	*****	X	0A
EXP_UNC_ERR	*****	X	04
FLAG1	*****	X	05
FLAG2	*****	X	05
FLAG3	*****	X	05
FLAG4	*****	X	05
FMT_1024_KB	*****	X	02
FMT_256_KB	*****	X	02
FMT_4_MB	*****	X	02
FMT_ARRAY	*****	X	0D
FMT_CPU_1	*****	X	07
FMT_CPU_2	*****	X	07
FMT_CPU_3	*****	X	0B
FMT_CPU_4	*****	X	0C
FMT_DATA_BUS	*****	X	04
FMT_ECC_1	*****	X	07
FMT_ECC_10	*****	X	07
FMT_ECC_11	*****	X	07
FMT_ECC_12	*****	X	07
FMT_ECC_13	*****	X	07
FMT_ECC_14	*****	X	07
FMT_ECC_2	*****	X	07
FMT_ECC_4	*****	X	07
FMT_ECC_5	*****	X	07
FMT_ECC_7	*****	X	07
FMT_ECC_8	*****	X	07
FMT_ECC_9	*****	X	07
FMT_EMPTY	*****	X	02
FMT_LOG_FULL	*****	X	0D
FMT_MEM_MAP	*****	X	02
FMT_MEM_MAP2	*****	X	02
FMT_MEM_MAP3	*****	X	02
FMT_NO_ARRAYS	*****	X	04
FMT_ROM_1	*****	X	0A
FMT_ROM_2	*****	X	0A
FMT_ROM_3	*****	X	0A
FMT_ROM_4	*****	X	0A
FMT_ROW_MSG	*****	X	0D
FMT_SAO	*****	X	04
FMT_SAI	*****	X	04
FMT_SGL_BIT_ERR	*****	X	0D
FREE_PAGE_P	*****	X	04
FROW	*****	X	05
GET_ARRAYS	*****	X	0E
HIGHROW	*****	X	0E
LOWADDR	*****	X	0E
LOWROW	*****	X	0E
L_CUR_ARRAY	*****	X	0D
MANUAL	= 00000003	G	

MEMORY SUBSYSTEM TESTS

MAXLWA	*****	X	02	T5_S9_X	00000AFD	R	07
MAXPFN	*****	X	04	T6_S1	00000019	RG	09
MAXROW	*****	X	05	T6_S1_X	00000108	R	09
MOVI_PAT_0	*****	X	0D	T6_S2	00000113	RG	09
MOVI_PAT_1	*****	X	0D	T6_S2_X	00000274	R	09
MSG_001	00000002	R	02	T6_S3	0000027F	RG	09
MSG_002	00000002	R	04	T6_S3_X	00000394	R	09
MSG_003	00000002	R	05	T6_S4	0000039F	RG	09
MSG_004	00000002	R	06	T6_S4_X	000005FE	R	09
MSG_005	00000002	R	07	T6_S5	00000609	RG	09
MSG_006	00000002	R	09	T6_S5_X	0000074D	R	09
MSG_007	00000002	R	0A	T7_S1	0000001E	RG	0A
MSG_008	00000002	R	0B	T7_S1_X	0000015A	R	0A
MSG_009	00000002	R	0C	T8_S1	0000001E	RG	0B
MSG_010	00000002	R	0D	T8_S1_X	000000B4	R	0B
MSG_011	00000002	R	0E	T8_S2	000000BF	RG	0B
MSG_012	00000002	R	0F	T8_S2_X	00000198	R	0B
NOISY_WORDS	*****	X	05	T8_S3	000001A3	RG	0B
NUM_COR_ERR	*****	X	0D	T8_S3_X	0000031E	R	0B
PATCNT	*****	X	0D	T9_S1	0000001E	RG	0C
PATTERN	*****	X	0D	T9_S1_X	000000FA	R	0C
PM	= 08000000			T9_S2	00000105	RG	0C
PRINT	*****	X	05	T9_S2_X	00000267	R	0C
PRINT_BAD_CONFIG	*****	X	02	TEMP	*****	X	06
PRINT_GENERAL	*****	X	07	TEMP1	*****	X	07
PRINT_UNX_UNC	*****	X	04	TEST_001	00000024	RG	02
READ_MAP	*****	X	02	TEST_001_X	00000116	RG	02
RETURN_PC	*****	X	04	TEST_002	00000014	RG	04
ROM_TYPE	*****	X	0A	TEST_002_X	00000198	RG	04
SEP_FUNCTIONAL	= 00000002			TEST_003	0000001C	RG	05
SEP_REPAIR	= 00000003			TEST_003_X	000002DE	RG	05
SEQUENCE	*****	X	0D	TEST_004	00000018	RG	06
SIZ...	= 00000008			TEST_004_X	00000230	RG	06
SLOT	*****	X	02	TEST_005	00000018	RG	07
SUMMARY_FLAG	*****	X	0D	TEST_005_X	00000ECF	RG	07
SYNDROME	*****	X	07	TEST_006	00000010	RG	09
T5_S1	00000021	RG	07	TEST_006_X	0000075B	RG	09
T5_S10	00000B08	RG	07	TEST_007	0000001C	RG	0A
T5_S10_X	00000CC6	R	07	TEST_007_X	00000168	RG	0A
T5_S11	00000CD1	RG	07	TEST_008	0000001C	RG	0B
T5_S11_X	00000EC1	R	07	TEST_008_X	0000032C	RG	0B
T5_S1_X	00000113	R	07	TEST_009	0000001C	RG	0C
T5_S2	0000011E	RG	07	TEST_009_X	00000275	RG	0C
T5_S2_X	000001E6	R	07	TEST_010	00000020	RG	0D
T5_S3	000001F1	RG	07	TEST_010_X	000004E2	RG	0D
T5_S3_X	000002DF	R	07	TEST_011	00000024	RG	0E
T5_S4	000002EA	RG	07	TEST_011_X	00000445	RG	0E
T5_S4_X	000003BC	R	07	TEST_012	00000020	RG	0F
T5_S5	000003C7	RG	07	TEST_012_X	00000368	RG	0F
T5_S5_X	00000641	R	07	TEST_ADDRESS	*****	X	02
T5_S6	0000064C	RG	07	TRANSITION	*****	X	05
T5_S6_X	000007DC	R	07	TRANSITION4	*****	X	05
T5_S7	000007E7	RG	07	UCE	= 80000000		
T5_S7_X	00000912	R	07	UCEIL	= 40000000		
T5_S8	0000091D	RG	07	UNX_UNC_PC	*****	X	04
T5_S8_X	00000A01	R	07	WHICH_ARRAY	*****	X	0D
T5_S9	00000A0C	RG	07				

 ! Psect synopsis !

PSECT name	Allocation	PSECT No.	Attributes
. ABS	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
\$ABS\$	0000FE70 (65136.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
TEST_001	0000011E (286.)	02 (2.)	NOPIC USR CON REL LCL NOSHR EXE RD NOWRT NOVEC PAGE
DISPATCH	00000120 (288.)	03 (3.)	NOPIC USR CON REL LCL NOSHR NOEXE RD NOWRT NOVEC LONG
TEST_002	000001A0 (416.)	04 (4.)	NOPIC USR CON REL LCL NOSHR EXE RD NOWRT NOVEC PAGE
TEST_003	000002E6 (742.)	05 (5.)	NOPIC USR CON REL LCL NOSHR EXE RD NOWRT NOVEC PAGE
TEST_004	00000238 (568.)	06 (6.)	NOPIC USR CON REL LCL NOSHR EXE RD NOWRT NOVEC PAGE
TEST_005	00000ED7 (3799.)	07 (7.)	NOPIC USR CON REL LCL NOSHR EXE RD NOWRT NOVEC PAGE
ARGLIST	00000108 (264.)	08 (8.)	NOPIC USR CON REL LCL NOSHR EXE RD NOWRT NOVEC LONG
TEST_006	00000763 (1891.)	09 (9.)	NOPIC USR CON REL LCL NOSHR EXE RD NOWRT NOVEC PAGE
TEST_007	00000170 (368.)	0A (10.)	NOPIC USR CON REL LCL NOSHR EXE RD NOWRT NOVEC PAGE
TEST_008	00000334 (820.)	0B (11.)	NOPIC USR CON REL LCL NOSHR EXE RD NOWRT NOVEC PAGE
TEST_009	0000027D (637.)	0C (12.)	NOPIC USR CON REL LCL NOSHR EXE RD NOWRT NOVEC PAGE
TEST_010	000004EA (1258.)	0D (13.)	NOPIC USR CON REL LCL NOSHR EXE RD NOWRT NOVEC PAGE
TEST_011	0000044D (1101.)	0E (14.)	NOPIC USR CON REL LCL NOSHR EXE RD NOWRT NOVEC PAGE
TEST_012	00000370 (880.)	0F (15.)	NOPIC USR CON REL LCL NOSHR EXE RD NOWRT NOVEC PAGE
\$TSTCNT	00000004 (4.)	10 (16.)	NOPIC USR OVR REL LCL NOSHR NOEXE RD NOWRT NOVEC LONG

 ! Performance indicators !

Phase	Page faults	CPU Time	Elapsed Time
Initialization	29	00:00:00.07	00:00:00.31
Command processing	867	00:00:00.60	00:00:01.01
Pass 1	1263	00:00:21.21	00:00:33.19
Symbol table sort	0	00:00:00.69	00:00:00.75
Pass 2	438	00:00:07.59	00:00:10.47
Symbol table output	2	00:00:00.18	00:00:00.29
Psect synopsis output	2	00:00:00.04	00:00:00.05
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	2602	00:00:30.38	00:00:46.07

The working set limit was 2048 pages.
 462790 bytes (904 pages) of virtual memory were used to buffer the intermediate code.
 There were 40 pages of symbol table space allocated to hold 396 non-local and 378 local symbols.
 2853 source lines were read in Pass 1, producing 92 object records in Pass 2.
 43 pages of virtual memory were used to define 40 macros.

 ! Macro library statistics !

Macro library name	Macros defined
SYS\$COMMON:[SYSLIB]DIAG.MLB;1	34
SYS\$COMMON:[SYSLIB]STARLET.MLB;2	6
TOTALS (all libraries)	40

ZZ-ECKAM-3.2 VAX-11 Macro Run Statistics
ECKAM MEMORY SUBSYSTEM TESTS
VAX-11 Macro Run Statistics

L 13

13-MAY-1987 Fiche 1 Frame L13 Sequence 167
23-JAN-1987 14:30:50 VAX/VMS Macro V04-00 Page 80
23-JAN-1987 14:29:55 [ZECCHINO.ECKAM]ECKAM1.MAR;6 (15)

539 GETS were required to define 40 macros.

There were no errors, warnings or information messages.

MAC/LIS ECKAM1

! Object Module Synopsis !

Module Name	Ident	Bytes	File	Creation Date	Creator
ECKAM	03-02	13507	[ZECCHINO.ECKAM]ECKAM0.OBJ;2	23-JAN-1987 14:30	VAX/VMS Macro V04-00
ECKAM	03-02	13322	[ZECCHINO.ECKAM]ECKAM1.OBJ;4	23-JAN-1987 14:30	VAX/VMS Macro V04-00

! Image Section Synopsis !

Cluster	Type	Pages	Base Addr	Disk	VCN	PFC	Protection and Paging	Global Sec. Name	Match	Majorid	Minorid
DEFAULT_CLUSTER	0	59	00000200		1	0	READ ONLY				

Key for special characters above:

! R - Relocatable !
! P - Protected !

! Program Section Synopsis !

Psect Name	Module Name	Base	End	Length	Align	Attributes
-----	-----	----	----	-----	-----	-----
\$HEADER	ECKAM	00000200 00000200	0000027F 0000027F	00000080 () 00000080 ()	128.) PAGE 9 128.) PAGE 9	NOPIC,USR,CON,REL,LCL,NOSHR,NOEXE, RD,NOWRT,NOVEC
\$TSTCNT	ECKAM ECKAM	00000280 00000280 00000280	00000283 00000280 00000283	00000004 () 00000000 () 00000004 ()	4.) LONG 2 0.) LONG 2 4.) LONG 2	NOPIC,USR,OVR,REL,LCL,NOSHR,NOEXE, RD,NOWRT,NOVEC
ARGLIST	ECKAM	00000284 00000284	0000038B 0000038B	00000108 () 00000108 ()	264.) LONG 2 264.) LONG 2	NOPIC,USR,CON,REL,LCL,NOSHR, EXE, RD,NOWRT,NOVEC
CLEANUP	ECKAM	0000038C 0000038C	000003DD 000003DD	00000052 () 00000052 ()	82.) LONG 2 82.) LONG 2	NOPIC,USR,CON,REL,LCL,NOSHR, EXE, RD, WRT,NOVEC
DATA	ECKAM	000003E0 000003E0	00002992 00002992	000025B3 () 000025B3 ()	9651.) LONG 2 9651.) LONG 2	NOPIC,USR,CON,REL,LCL,NOSHR, EXE, RD, WRT,NOVEC
DISPATCH	ECKAM ECKAM	00002994 00002994 00002994	00002AB3 00002994 00002AB3	00000120 () 00000000 () 00000120 ()	288.) LONG 2 0.) LONG 2 288.) LONG 2	NOPIC,USR,CON,REL,LCL,NOSHR,NOEXE, RD,NOWRT,NOVEC
DISPATCH_X	ECKAM	00002AB4 00002AB4	00002ACB 00002ACB	00000018 () 00000018 ()	24.) LONG 2 24.) LONG 2	NOPIC,USR,CON,REL,LCL,NOSHR,NOEXE, RD,NOWRT,NOVEC
HEADER	ECKAM	00002ACC 00002ACC	00002B0C 00002B0C	00000041 () 00000041 ()	65.) LONG 2 65.) LONG 2	NOPIC,USR,CON,REL,LCL,NOSHR, EXE, RD, WRT,NOVEC
INITIALIZE	ECKAM	00002B10 00002B10	00002D3E 00002D3E	0000022F () 0000022F ()	559.) LONG 2 559.) LONG 2	NOPIC,USR,CON,REL,LCL,NOSHR, EXE, RD, WRT,NOVEC
ROUTINES	ECKAM	00002E00 00002E00	000038C1 000038C1	00000AC2 () 00000AC2 ()	2754.) PAGE 9 2754.) PAGE 9	NOPIC,USR,CON,REL,LCL,NOSHR, EXE, RD,NOWRT,NOVEC
SUMMARY	ECKAM	000038C4 000038C4	000039B7 000039B7	000000F4 () 000000F4 ()	244.) LONG 2 244.) LONG 2	NOPIC,USR,CON,REL,LCL,NOSHR, EXE, RD, WRT,NOVEC
TEST_001	ECKAM	00003A00 00003A00	00003B1D 00003B1D	0000011E () 0000011E ()	286.) PAGE 9 286.) PAGE 9	NOPIC,USR,CON,REL,LCL,NOSHR, EXE, RD,NOWRT,NOVEC
TEST_002	ECKAM	00003C00 00003C00	00003D9F 00003D9F	000001A0 () 000001A0 ()	416.) PAGE 9 416.) PAGE 9	NOPIC,USR,CON,REL,LCL,NOSHR, EXE, RD,NOWRT,NOVEC
TEST_003	ECKAM	00003E00 00003E00	000040E5 000040E5	000002E6 () 000002E6 ()	742.) PAGE 9 742.) PAGE 9	NOPIC,USR,CON,REL,LCL,NOSHR, EXE, RD,NOWRT,NOVEC
TEST_004	ECKAM	00004200 00004200	00004437 00004437	00000238 () 00000238 ()	568.) PAGE 9 568.) PAGE 9	NOPIC,USR,CON,REL,LCL,NOSHR, EXE, RD,NOWRT,NOVEC
TEST_005		00004600	000054D6	00000ED7 ()	3799.) PAGE 9	NOPIC,USR,CON,REL,LCL,NOSHR, EXE, RD,NOWRT,NOVEC

<u>Psect Name</u>	<u>Module Name</u>	<u>Base</u>	<u>End</u>	<u>Length</u>	<u>Align</u>	<u>Attributes</u>
TEST_005	ECKAM	00004600 00004600	000054D6 000054D6	00000ED7 () 00000ED7 ()	3799.) PAGE 9 3799.) PAGE 9	NOPIC,USR,CON,REL,LCL,NOSHR, EXE, RD,NOWRT,NOVEC
TEST_006	ECKAM	00005600 00005600	00005D62 00005D62	00000763 () 00000763 ()	1891.) PAGE 9 1891.) PAGE 9	NOPIC,USR,CON,REL,LCL,NOSHR, EXE, RD,NOWRT,NOVEC
TEST_007	ECKAM	00005E00 00005E00	00005F6F 00005F6F	00000170 () 00000170 ()	368.) PAGE 9 368.) PAGE 9	NOPIC,USR,CON,REL,LCL,NOSHR, EXE, RD,NOWRT,NOVEC
TEST_008	ECKAM	00006000 00006000	00006333 00006333	00000334 () 00000334 ()	820.) PAGE 9 820.) PAGE 9	NOPIC,USR,CON,REL,LCL,NOSHR, EXE, RD,NOWRT,NOVEC
TEST_009	ECKAM	00006400 00006400	0000667C 0000667C	0000027D () 0000027D ()	637.) PAGE 9 637.) PAGE 9	NOPIC,USR,CON,REL,LCL,NOSHR, EXE, RD,NOWRT,NOVEC
TEST_010	ECKAM	00006800 00006800	00006CE9 00006CE9	000004EA () 000004EA ()	1258.) PAGE 9 1258.) PAGE 9	NOPIC,USR,CON,REL,LCL,NOSHR, EXE, RD,NOWRT,NOVEC
TEST_011	ECKAM	00006E00 00006E00	0000724C 0000724C	0000044D () 0000044D ()	1101.) PAGE 9 1101.) PAGE 9	NOPIC,USR,CON,REL,LCL,NOSHR, EXE, RD,NOWRT,NOVEC
TEST_012	ECKAM	00007400 00007400	0000776F 0000776F	00000370 () 00000370 ()	880.) PAGE 9 880.) PAGE 9	NOPIC,USR,CON,REL,LCL,NOSHR, EXE, RD,NOWRT,NOVEC
_LAST	ECKAM	00007800 00007800	00007800 00007800	00000000 () 00000000 ()	0.) PAGE 9 0.) PAGE 9	NOPIC,USR,CON,REL,LCL,NOSHR, EXE, RD, WRT,NOVEC

 ! Symbol Cross Reference !

Symbol	Value	Defined By	Referenced By ...
-----	-----	-----	-----
\$ENV	00000001	ECKAM	
\$MO	00000001	ECKAM	
ALL	00000001	ECKAM	
ALLMOD	00001583-R	ECKAM	ECKAM
ARRAY	0000157A-R	ECKAM	ECKAM
ARRAY_SGL_BIT	000015B4-R	ECKAM	ECKAM
ASK	00001791-R	ECKAM	
BUS_ERR_REG	0000040D-R	ECKAM	ECKAM
CK	000009B1-R	ECKAM	ECKAM
CK0	000009B1-R	ECKAM	
CK1	000009B7-R	ECKAM	
CK2	000009BE-R	ECKAM	
CK3	000009C6-R	ECKAM	
CK4	000009CD-R	ECKAM	
CK5	000009D5-R	ECKAM	
CK6	000009DC-R	ECKAM	
CKALL	000009E3-R	ECKAM	
CKBIT	000008E9-R	ECKAM	ECKAM
CKBIT0	000008E9-R	ECKAM	
CKBIT1	00000901-R	ECKAM	
CKBIT2	0000091D-R	ECKAM	
CKBIT3	0000093D-R	ECKAM	
CKBIT4	00000959-R	ECKAM	
CKBIT5	00000979-R	ECKAM	
CKBIT6	00000995-R	ECKAM	
CODEVECTORS	00000855-R	ECKAM	
CODEWORDS	00000869-R	ECKAM	ECKAM
CONTROL	00001595-R	ECKAM	ECKAM
CORRUPT_TEST_LOC	00003013-R	ECKAM	ECKAM
CORR_ERR_ISR	00003170-R	ECKAM	
COR_ERRORS	0000101C-R	ECKAM	
COUNTER	00001444-R	ECKAM	
CPU	000015A8-R	ECKAM	ECKAM
DATA	000003E0-R	ECKAM	ECKAM
DECODED_WORDS	00000F1C-R	ECKAM	ECKAM
DECODER	000030A5-R	ECKAM	ECKAM
DEFAULT	00000000	ECKAM	
DEVICE_TYPE	000003F8-R	ECKAM	ECKAM
DIGIT	00001581-R	ECKAM	
DIGIT1	00001802-R	ECKAM	ECKAM
DS\$ABORT	00010020	ECKAM	
DS\$ASKADR	00010090	ECKAM	
DS\$ASKDATA	00010080	ECKAM	
DS\$ASKLGCL	00010098	ECKAM	
DS\$ASKSTR	000100A0	ECKAM	
DS\$ASKVLD	00010088	ECKAM	
DS\$ATTACH	000101A8	ECKAM	
DS\$BGNSUB	00010030	ECKAM	ECKAM
DS\$BRANCH	000100A8	ECKAM	

Symbol	Value	Defined By	Referenced By ...
DS\$BREAK	00010058	ECKAM	ECKAM
DS\$CANWAIT	00010070	ECKAM	
DS\$CHANNEL	00010180	ECKAM	
DS\$CKLOOP	00010040	ECKAM	ECKAM
DS\$CLRVEC	00010168	ECKAM	
DS\$CNTRLC	00010078	ECKAM	
DS\$CVTREG	000100B0	ECKAM	
DS\$ENDPASS	00010010	ECKAM	
DS\$ENDSUB	00010038	ECKAM	ECKAM
DS\$ERRDEV	000100C8	ECKAM	
DS\$ERRHARD	000100D0	ECKAM	ECKAM
DS\$ERRPREP	00010108	ECKAM	
DS\$ERRSOFT	000100D8	ECKAM	
DS\$ERRSYS	000100C0	ECKAM	
DS\$ESCAPE	00010050	ECKAM	
DS\$FREEDBGSYM	000101B8	ECKAM	
DS\$GETBUF	00010120	ECKAM	
DS\$GETMEM	00010130	ECKAM	
DS\$GETTERM	000101C8	ECKAM	
DS\$GPHARD	00010018	ECKAM	
DS\$HELP	000101B0	ECKAM	
DS\$INITSCB	00010170	ECKAM	
DS\$INLOOP	00010048	ECKAM	
DS\$K_BBC	0000001D	ECKAM	
DS\$K_BBCC	00000021	ECKAM	
DS\$K_BBCCI	00000023	ECKAM	
DS\$K_BBCS	0000001F	ECKAM	
DS\$K_BBS	0000001C	ECKAM	
DS\$K_BBSC	00000020	ECKAM	
DS\$K_BBSS	0000001E	ECKAM	
DS\$K_BBSSI	00000022	ECKAM	
DS\$K_BCC_M	0000001B	ECKAM	
DS\$K_BCS_M	0000001A	ECKAM	
DS\$K_BEQLU_B	00000005	ECKAM	
DS\$K_BEQLU_M	00000011	ECKAM	
DS\$K_BEQL_B	00000004	ECKAM	
DS\$K_BEQL_M	00000010	ECKAM	
DS\$K_BERRÖR	00000026	ECKAM	
DS\$K_BGEQU_B	00000007	ECKAM	
DS\$K_BGEQU_M	00000013	ECKAM	
DS\$K_BGEQ_B	00000006	ECKAM	
DS\$K_BGEQ_M	00000012	ECKAM	
DS\$K_BGTRÜ_B	00000009	ECKAM	
DS\$K_BGTRU_M	00000015	ECKAM	
DS\$K_BGTR_B	00000008	ECKAM	
DS\$K_BGTR_M	00000014	ECKAM	
DS\$K_BLBC	00000025	ECKAM	
DS\$K_BLBS	00000024	ECKAM	
DS\$K_BLEQU_B	00000003	ECKAM	
DS\$K_BLEQU_M	0000000F	ECKAM	
DS\$K_BLEQ_B	00000002	ECKAM	
DS\$K_BLEQ_M	0000000E	ECKAM	

Symbol	Value	Defined By	Referenced By ...
DS\$K_BLSSU_B	00000001	ECKAM	
DS\$K_BLSSU_M	0000000D	ECKAM	
DS\$K_BLSS_B	00000000	ECKAM	
DS\$K_BLSS_M	0000000C	ECKAM	
DS\$K_BNEQU_B	0000000B	ECKAM	
DS\$K_BNEQU_M	00000017	ECKAM	
DS\$K_BNEQ_B	0000000A	ECKAM	
DS\$K_BNEQ_M	00000016	ECKAM	
DS\$K_BNERROR	00000027	ECKAM	
DS\$K_BVC_M	00000019	ECKAM	
DS\$K_BVS_M	00000018	ECKAM	
DS\$LOAD	00010198	ECKAM	
DS\$LOADPCS	000101C0	ECKAM	
DS\$MAPDBGBLOCK	00010118	ECKAM	
DS\$MEMSIZE	000101D8	ECKAM	
DS\$MMOFF	00010158	ECKAM	
DS\$MMON	00010150	ECKAM	
DS\$MOVPHY	00010148	ECKAM	
DS\$MOVVRT	00010140	ECKAM	
DS\$PARSE	000100B8	ECKAM	
DS\$PRINTB	000100E0	ECKAM	ECKAM
DS\$PRINTF	000100F0	ECKAM	
DS\$PRINTS	000100F8	ECKAM	
DS\$PRINTSIG	00010100	ECKAM	
DS\$PRINTX	000100E8	ECKAM	ECKAM
DS\$PROBE	000101A0	ECKAM	
DS\$RELBUF	00010128	ECKAM	
DS\$RELMEM	00010138	ECKAM	
DS\$SERVICE_DISPATCHER	000101D0	ECKAM	
DS\$SETIPL	00010178	ECKAM	
DS\$SETMAP	00010188	ECKAM	
DS\$SETPRIEXV	00010110	ECKAM	
DS\$SETVEC	00010160	ECKAM	
DS\$SHOCHAN	00010190	ECKAM	
DS\$SUMMARY	00010028	ECKAM	
DS\$WAITMS	00010060	ECKAM	
DS\$WAITUS	00010068	ECKAM	
ERROR_TYPE	000003F4-R	ECKAM	ECKAM
ERR_LOG_FULL	00000821-R	ECKAM	ECKAM
EXHAUSTIVE	00000002	ECKAM	
EXP_COR_ERR	000003FD-R	ECKAM	ECKAM
EXP_INF_LST	00000405-R	ECKAM	
EXP_MSG	000025BC-R	ECKAM	
EXP_NX_MEM	00000409-R	ECKAM	ECKAM
EXP_UNC_ERR	00000401-R	ECKAM	ECKAM
FLAG1	00001434-R	ECKAM	ECKAM
FLAG2	00001438-R	ECKAM	ECKAM
FLAG3	0000143C-R	ECKAM	ECKAM
FLAG4	00001440-R	ECKAM	ECKAM
FMT_1024_KB	00001699-R	ECKAM	ECKAM
FMT_256_KB	0000167C-R	ECKAM	ECKAM
FMT_4_MB	000016B7-R	ECKAM	ECKAM

Symbol	Value	Defined By	Referenced By ...
FMT_ADR_BUS	000020FB-R	ECKAM	
FMT_ARRAY	000017F9-R	ECKAM	ECKAM
FMT_ASK_HIGH	00001786-R	ECKAM	
FMT_ASK_LOW	0000177B-R	ECKAM	
FMT_AVAILABLE	00001719-R	ECKAM	
FMT_BAD_LOC	0000228D-R	ECKAM	
FMT_BODY	000020A3-R	ECKAM	
FMT_COMP_ERR	000020BC-R	ECKAM	
FMT_CPE_MC	00002153-R	ECKAM	
FMT_CPU_0	0000182D-R	ECKAM	
FMT_CPU_1	0000188D-R	ECKAM	ECKAM
FMT_CPU_2	000018F1-R	ECKAM	ECKAM
FMT_CPU_3	00001959-R	ECKAM	ECKAM
FMT_CPU_4	000019AE-R	ECKAM	ECKAM
FMT_CS_BUS	00002168-R	ECKAM	
FMT_CS_MC	000020E1-R	ECKAM	
FMT_DATA_BUS	00002117-R	ECKAM	ECKAM
FMT_ECC_1	00001AC0-R	ECKAM	ECKAM
FMT_ECC_10	00001E2E-R	ECKAM	ECKAM
FMT_ECC_11	00001E9A-R	ECKAM	ECKAM
FMT_ECC_12	00001EBA-R	ECKAM	ECKAM
FMT_ECC_13	00001EF9-R	ECKAM	ECKAM
FMT_ECC_14	00001F4D-R	ECKAM	ECKAM
FMT_ECC_15	00001F90-R	ECKAM	
FMT_ECC_16	00001FC4-R	ECKAM	
FMT_ECC_2	00001B10-R	ECKAM	ECKAM
FMT_ECC_3	00001B6F-R	ECKAM	
FMT_ECC_4	00001BE2-R	ECKAM	ECKAM
FMT_ECC_5	00001C44-R	ECKAM	ECKAM
FMT_ECC_6	00001CA5-R	ECKAM	
FMT_ECC_7	00001D01-R	ECKAM	ECKAM
FMT_ECC_8	00001D60-R	ECKAM	ECKAM
FMT_ECC_9	00001DBD-R	ECKAM	ECKAM
FMT_EMPTY	00001704-R	ECKAM	ECKAM
FMT_ILL_CFG	0000251F-R	ECKAM	
FMT_INF_LST	0000212B-R	ECKAM	
FMT_LOG_FULL	00002795-R	ECKAM	ECKAM
FMT_MCH_STACK1	00002814-R	ECKAM	
FMT_MCH_STACK2	000028D1-R	ECKAM	
FMT_MEM_MAP	000015E9-R	ECKAM	ECKAM
FMT_MEM_MAP2	0000161A-R	ECKAM	ECKAM
FMT_MEM_MAP3	0000164B-R	ECKAM	ECKAM
FMT_MEM_REG	000027AB-R	ECKAM	
FMT_MEM_SIZ	000015C2-R	ECKAM	
FMT_NO_ARRAYS	00001793-R	ECKAM	ECKAM
FMT_NO_SUPP	000016D2-R	ECKAM	
FMT_NSBH	00002187-R	ECKAM	
FMT_NX_MEM	000021A1-R	ECKAM	
FMT_RAM	000021BD-R	ECKAM	
FMT_REG_ERR	00002565-R	ECKAM	
FMT_ROM_1	000017C1-R	ECKAM	ECKAM
FMT_ROM_2	00001A10-R	ECKAM	ECKAM

Symbol	Value	Defined By	Referenced By ...
FMT_ROM_3	000017E5-R	ECKAM	ECKAM
FMT_ROM_4	00001A83-R	ECKAM	ECKAM
FMT_ROW	000021E3-R	ECKAM	
FMT_ROW_MSG	00001817-R	ECKAM	ECKAM
FMT_SAO	00002204-R	ECKAM	ECKAM
FMT_SAI	00002212-R	ECKAM	ECKAM
FMT_SGL_BIT_ERR	00002961-R	ECKAM	ECKAM
FMT_SHORT	0000221F-R	ECKAM	
FMT_SIZ_ERR	00002232-R	ECKAM	
FMT_UNK_ET	00001FFB-R	ECKAM	
FMT_UNK_MC	00002010-R	ECKAM	
FMT_UNX_TO	00002039-R	ECKAM	
FMT_UNX_UNC	0000205D-R	ECKAM	
FMT_WARN_MES2	00002437-R	ECKAM	
FMT_WARN_MESS	00002350-R	ECKAM	
FMT_WBE_MC	0000250D-R	ECKAM	
FREE_PAGE_P	00000845-R	ECKAM	ECKAM
FREE_PAGE_V	0000084D-R	ECKAM	
FROM	00001430-R	ECKAM	ECKAM
GET_ARRAYS	00003558-R	ECKAM	ECKAM
HIGHADDR	00000841-R	ECKAM	
HIGHROW	00000839-R	ECKAM	ECKAM
LOWADDR	0000083D-R	ECKAM	ECKAM
LOWROW	00000835-R	ECKAM	ECKAM
L_CUR_ARRAY	00001450-R	ECKAM	ECKAM
MACHINECK	00003230-R	ECKAM	
MANUAL	00000003	ECKAM	
MAXLWA	00000829-R	ECKAM	ECKAM
MAXPFN	00000825-R	ECKAM	ECKAM
MAXROW	0000082D-R	ECKAM	ECKAM
MOVI_PAT_0	00000F04-R	ECKAM	ECKAM
MOVI_PAT_1	00000F0C-R	ECKAM	ECKAM
NOISY_WORDS	00000F9C-R	ECKAM	ECKAM
NUM_COR_ERR	00000815-R	ECKAM	ECKAM
PATCNT	000003E8-R	ECKAM	ECKAM
PATTERN	00000F14-R	ECKAM	ECKAM
PGE_COR_ERR	00000411-R	ECKAM	
PRINT	00002E00-R	ECKAM	ECKAM
PRINT_BAD_CONFIG	00002FF7-R	ECKAM	ECKAM
PRINT_GENERAL	00002EC8-R	ECKAM	ECKAM
PRINT_UNX_TO	00003026-R	ECKAM	
PRINT_UNX_UNC	00002F36-R	ECKAM	ECKAM
READ_MAP	000033B9-R	ECKAM	ECKAM
REC_MSG	000025B0-R	ECKAM	
RETURN_PC	000003EC-R	ECKAM	ECKAM
ROM_TYPE	000003FB-R	ECKAM	ECKAM
ROWCON	00001428-R	ECKAM	
SEQUENCE	000003E4-R	ECKAM	ECKAM
SLOT	0000141C-R	ECKAM	ECKAM
SUMMARY_FLAG	0000081D-R	ECKAM	ECKAM
SYNDROME	000009E4-R	ECKAM	ECKAM
SYSSALLOC	00010238	ECKAM	

Symbol	Value	Defined By	Referenced By ...
SYSSASCTIM	00010248	ECKAM	
SYSSASSIGN	00010250	ECKAM	
SYSSBINTIM	00010258	ECKAM	
SYSSCANCEL	00010260	ECKAM	
SYSSCANTIM	00010268	ECKAM	
SYSSCLOSE	000105B8	ECKAM	
SYSSCLREF	00010298	ECKAM	
SYSSCONNECT	000105C0	ECKAM	
SYSSDALLOC	000102D8	ECKAM	
SYSSDASSGN	000102E0	ECKAM	
SYSSDISCONNECT	000105D0	ECKAM	
SYSSFAO	00010350	ECKAM	
SYSSFAOL	00010358	ECKAM	
SYSSGET	00010580	ECKAM	
SYSSGETCHN	000104C8	ECKAM	
SYSSGETTIM	00010378	ECKAM	
SYSSLKWSSET	000103A0	ECKAM	
SYSSNUMTIM	000103B8	ECKAM	
SYSSOPEN	00010608	ECKAM	
SYSSQIO	000103C8	ECKAM	
SYSSQIOW	00010200	ECKAM	
SYSSREAD	00010590	ECKAM	
SYSSREADEF	000103D0	ECKAM	
SYSSSETAST	000103F8	ECKAM	
SYSSSETEF	00010400	ECKAM	
SYSSSETIMR	00010420	ECKAM	
SYSSSETPRI	00010428	ECKAM	
SYSSSETPRT	00010430	ECKAM	
SYSSSETRWM	00010438	ECKAM	
SYSSSETSWM	00010448	ECKAM	
SYSSULKPAG	00010460	ECKAM	
SYSSULWSET	00010468	ECKAM	
SYSSUNWIND	00010470	ECKAM	
SYSSWAITFR	00010478	ECKAM	
SYSSWFLAND	00010488	ECKAM	
SYSSWFLOR	00010490	ECKAM	
T5_S1	00004621-R	ECKAM	
T5_S10	00005108-R	ECKAM	
T5_S11	000052D1-R	ECKAM	
T5_S2	0000471E-R	ECKAM	
T5_S3	000047F1-R	ECKAM	
T5_S4	000048EA-R	ECKAM	
T5_S5	000049C7-R	ECKAM	
T5_S6	00004C4C-R	ECKAM	
T5_S7	00004DE7-R	ECKAM	
T5_S8	00004F1D-R	ECKAM	
T5_S9	0000500C-R	ECKAM	
T6_S1	00005619-R	ECKAM	
T6_S2	00005713-R	ECKAM	
T6_S3	0000587F-R	ECKAM	
T6_S4	0000599F-R	ECKAM	
T6_S5	00005C09-R	ECKAM	

Symbol	Value	Defined By	Referenced By ...
T7_S1	00005E1E-R	ECKAM	
T8_S1	0000601E-R	ECKAM	
T8_S2	000060BF-R	ECKAM	
T8_S3	000061A3-R	ECKAM	
T9_S1	0000641E-R	ECKAM	
T9_S2	00006505-R	ECKAM	
TEMP	00000C04-R	ECKAM	ECKAM
TEMP1	00000E84-R	ECKAM	ECKAM
TEST_001	00003A24-R	ECKAM	
TEST_001_X	00003B16-R	ECKAM	
TEST_002	00003C14-R	ECKAM	
TEST_002_X	00003D98-R	ECKAM	
TEST_003	00003E1C-R	ECKAM	
TEST_003_X	000040DE-R	ECKAM	
TEST_004	00004218-R	ECKAM	
TEST_004_X	00004430-R	ECKAM	
TEST_005	00004618-R	ECKAM	
TEST_005_X	000054CF-R	ECKAM	
TEST_006	00005610-R	ECKAM	
TEST_006_X	00005D5B-R	ECKAM	
TEST_007	00005E1C-R	ECKAM	
TEST_007_X	00005F68-R	ECKAM	
TEST_008	0000601C-R	ECKAM	
TEST_008_X	0000632C-R	ECKAM	
TEST_009	0000641C-R	ECKAM	
TEST_009_X	00006675-R	ECKAM	
TEST_010	00006820-R	ECKAM	
TEST_010_X	00006CE2-R	ECKAM	
TEST_011	00006E24-R	ECKAM	
TEST_011_X	00007245-R	ECKAM	
TEST_012	00007420-R	ECKAM	
TEST_012_X	00007768-R	ECKAM	
TEST_ADDRESS	00000831-R	ECKAM	ECKAM
TIME_LEFT	00001454-R	ECKAM	
TOTAL_COR_ERR	00000819-R	ECKAM	
TRANSITION	00001448-R	ECKAM	ECKAM
TRANSITION4	0000144C-R	ECKAM	ECKAM
UNIX_UNC_PC	000003F0-R	ECKAM	ECKAM
VA_COR_ADDRESS	00000415-R	ECKAM	
WHICH_ARRAY	0000307A-R	ECKAM	ECKAM
XOR_MSG	000025C8-R	ECKAM	

! Symbols By Value !

Value		Symbols...		
00000000	DEFAULT	DS\$K_BLSS_B		
00000001	\$ENV	\$MO	ALL	DS\$K_BLSSU_B
00000002	DS\$K_BLEQ_B	EXHAUSTIVE		
00000003	DS\$K_BLEQU_B	MANUAL		
00000004	DS\$K_BEQL_B			
00000005	DS\$K_BEQLU_B			
00000006	DS\$K_BGEQ_B			
00000007	DS\$K_BGEQU_B			
00000008	DS\$K_BGTR_B			
00000009	DS\$K_BGTRU_B			
0000000A	DS\$K_BNEQ_B			
0000000B	DS\$K_BNEQU_B			
0000000C	DS\$K_BLSS_M			
0000000D	DS\$K_BLSSU_M			
0000000E	DS\$K_BLEQ_M			
0000000F	DS\$K_BLEQU_M			
00000010	DS\$K_BEQL_M			
00000011	DS\$K_BEQLU_M			
00000012	DS\$K_BGEQ_M			
00000013	DS\$K_BGEQU_M			
00000014	DS\$K_BGTR_M			
00000015	DS\$K_BGTRU_M			
00000016	DS\$K_BNEQ_M			
00000017	DS\$K_BNEQU_M			
00000018	DS\$K_BVS_M			
00000019	DS\$K_BVC_M			
0000001A	DS\$K_BCS_M			
0000001B	DS\$K_BCC_M			
0000001C	DS\$K_BBS			
0000001D	DS\$K_BBC			
0000001E	DS\$K_BBSS			
0000001F	DS\$K_BBCS			
00000020	DS\$K_BBSC			
00000021	DS\$K_BBCC			
00000022	DS\$K_BBSSI			
00000023	DS\$K_BBCCI			
00000024	DS\$K_BLBS			
00000025	DS\$K_BLBC			
00000026	DS\$K_BERROR			
00000027	DS\$K_BNERROR			
000003E0	R-DATA			
000003E4	R-SEQUENCE			
000003E8	R-PATCNT			
000003EC	R-RETURN_PC			
000003F0	R-UNIX_UNC_PC			
000003F4	R-ERROR_TYPE			
000003F8	R-DEVICE_TYPE			
000003FB	R-ROM_TYPE			
000003FD	R-EXP_COR_ERR			

Value	Symbols...
00000401	R-EXP_UNC_ERR
00000405	R-EXP_INF_LST
00000409	R-EXP_NX_MEM
0000040D	R-BUS_ERR_REG
00000411	R-PGE_COR_ERR
00000415	R-VA_COR_ADDRESS
00000815	R-NUM_COR_ERR
00000819	R-TOTAL_COR_ERR
0000081D	R-SUMMARY_FLAG
00000821	R-ERR_LOG_FULL
00000825	R-MAXPFN
00000829	R-MAXLWA
0000082D	R-MAXROW
00000831	R-TEST_ADDRESS
00000835	R-LOWROW
00000839	R-HIGHROW
0000083D	R-LOWADDR
00000841	R-HIGHADDR
00000845	R-FREE_PAGE_P
0000084D	R-FREE_PAGE_V
00000855	R-CODEVECTORS
00000869	R-CODEWORDS
000008E9	R-CKBIT
00000901	R-CKBIT1
0000091D	R-CKBIT2
0000093D	R-CKBIT3
00000959	R-CKBIT4
00000979	R-CKBIT5
00000995	R-CKBIT6
000009B1	R-CK
000009B7	R-CK1
000009BE	R-CK2
000009C6	R-CK3
000009CD	R-CK4
000009D5	R-CK5
000009DC	R-CK6
000009E3	R-CKALL
000009E4	R-SYNDROME
00000C04	R-TEMP
00000E84	R-TEMP1
00000F04	R-MOVI_PAT_0
00000F0C	R-MOVI_PAT_1
00000F14	R-PATTERN
00000F1C	R-DECODED_WORDS
00000F9C	R-NOISY_WORDS
0000101C	R-COR_ERRORS
0000141C	R-SLOT
00001428	R-ROWCON
00001430	R-FROW
00001434	R-FLAG1
00001438	R-FLAG2
0000143C	R-FLAG3

Value	Symbols...
00001440	R-FLAG4
00001444	R-COUNTER
00001448	R-TRANSITION
0000144C	R-TRANSITION4
00001450	R-L_CUR_ARRAY
00001454	R-TIME_LEFT
0000157A	R-ARRAY
00001581	R-DIGIT
00001583	R-ALLMOD
00001595	R-CONTROL
000015A8	R-CPU
000015B4	R-ARRAY_SGL_BIT
000015C2	R-FMT_MEM_SIZ
000015E9	R-FMT_MEM_MAP
0000161A	R-FMT_MEM_MAP2
0000164B	R-FMT_MEM_MAP3
0000167C	R-FMT_256_KB
00001699	R-FMT_1024_KB
000016B7	R-FMT_4_MB
000016D2	R-FMT_NO_SUPP
00001704	R-FMT_EMPTY
00001719	R-FMT_AVAILABLE
0000177B	R-FMT_ASK_LOW
00001786	R-FMT_ASK_HIGH
00001791	R-ASK
00001793	R-FMT_NO_ARRAYS
000017C1	R-FMT_ROM_1
000017E5	R-FMT_ROM_3
000017F9	R-FMT_ARRAY
00001802	R-DIGIT1
00001817	R-FMT_ROW_MSG
0000182D	R-FMT_CPU_0
0000188D	R-FMT_CPU_1
000018F1	R-FMT_CPU_2
00001959	R-FMT_CPU_3
000019AE	R-FMT_CPU_4
00001A10	R-FMT_ROM_2
00001A83	R-FMT_ROM_4
00001AC0	R-FMT_ECC_1
00001B10	R-FMT_ECC_2
00001B6F	R-FMT_ECC_3
00001BE2	R-FMT_ECC_4
00001C44	R-FMT_ECC_5
00001CA5	R-FMT_ECC_6
00001D01	R-FMT_ECC_7
00001D60	R-FMT_ECC_8
00001DBD	R-FMT_ECC_9
00001E2E	R-FMT_ECC_10
00001E9A	R-FMT_ECC_11
00001EBA	R-FMT_ECC_12
00001EF9	R-FMT_ECC_13
00001F4D	R-FMT_ECC_14

Value	Symbols...
00001F90	R-FMT_ECC_15
00001FC4	R-FMT_ECC_16
00001FFB	R-FMT_UNK_ET
00002010	R-FMT_UNK_MC
00002039	R-FMT_UNX_TO
0000205D	R-FMT_UNX_UNC
000020A3	R-FMT_BODY
000020BC	R-FMT_COMP_ERR
000020E1	R-FMT_CS_MC
000020FB	R-FMT_ADR_BUS
00002117	R-FMT_DATA_BUS
0000212B	R-FMT_INF_LST
00002153	R-FMT_CPE_MC
00002168	R-FMT_CS_BUS
00002187	R-FMT_NSBH
000021A1	R-FMT_NX_MEM
000021BD	R-FMT_RAM
000021E3	R-FMT_ROW
00002204	R-FMT_SAO
00002212	R-FMT_SAI
0000221F	R-FMT_SHORT
00002232	R-FMT_SIZ_ERR
0000228D	R-FMT_BAD_LOC
00002350	R-FMT_WARN_MESS
00002437	R-FMT_WARN_MES2
0000250D	R-FMT_WBE_MC
0000251F	R-FMT_ILL_CFG
00002565	R-FMT_REG_ERR
000025B0	R-REC_MSG
000025BC	R-EXP_MSG
000025C8	R-XOR_MSG
00002795	R-FMT_LOG_FULL
000027AB	R-FMT_MEM_REG
00002814	R-FMT_MCH_STACK1
000028D1	R-FMT_MCH_STACK2
00002961	R-FMT_SGL_BIT_ERR
00002E00	R-PRINT
00002EC8	R-PRINT_GENERAL
00002F36	R-PRINT_UNX_UNC
00002FF7	R-PRINT_BAD_CONFIG
00003013	R-CORRUPT_TEST_LOC
00003026	R-PRINT_UNX_TO
0000307A	R-WHICH_ARRAY
000030A5	R-DECODER
00003170	R-CORR_ERR_ISR
00003230	R-MACHINECK
000033B9	R-READ_MAP
00003558	R-GET_ARRAYS
00003A24	R-TEST_001
00003B16	R-TEST_001_X
00003C14	R-TEST_002
00003D98	R-TEST_002_X

Value	Symbols...
00003E1C	R-TEST_003
000040DE	R-TEST_003_X
00004218	R-TEST_004
00004430	R-TEST_004_X
00004618	R-TEST_005
00004621	R-T5_S1
0000471E	R-T5_S2
000047F1	R-T5_S3
000048EA	R-T5_S4
000049C7	R-T5_S5
00004C4C	R-T5_S6
00004DE7	R-T5_S7
00004F1D	R-T5_S8
0000500C	R-T5_S9
00005108	R-T5_S10
000052D1	R-T5_S11
000054CF	R-TEST_005_X
00005610	R-TEST_006
00005619	R-T6_S1
00005713	R-T6_S2
0000587F	R-T6_S3
0000599F	R-T6_S4
00005C09	R-T6_S5
00005D5B	R-TEST_006_X
00005E1C	R-TEST_007
00005E1E	R-T7_S1
00005F68	R-TEST_007_X
0000601C	R-TEST_008
0000601E	R-T8_S1
000060BF	R-T8_S2
000061A3	R-T8_S3
0000632C	R-TEST_008_X
0000641C	R-TEST_009
0000641E	R-T9_S1
00006505	R-T9_S2
00006675	R-TEST_009_X
00006820	R-TEST_010
00006CE2	R-TEST_010_X
00006E24	R-TEST_011
00007245	R-TEST_011_X
00007420	R-TEST_012
00007768	R-TEST_012_X
00010010	DS\$ENDPASS
00010018	DS\$GPHARD
00010020	DS\$ABORT
00010028	DS\$SUMMARY
00010030	DS\$BGNSUB
00010038	DS\$ENDSUB
00010040	DS\$CKLOOP
00010048	DS\$INLOOP
00010050	DS\$ESCAPE
00010058	DS\$BREAK

Value	Symbols...
00010060	DS\$WAITMS
00010068	DS\$WAITUS
00010070	DS\$CANWAIT
00010078	DS\$CNTRLC
00010080	DS\$ASKDATA
00010088	DS\$ASKVLD
00010090	DS\$ASKADR
00010098	DS\$ASKLGCL
000100A0	DS\$ASKSTR
000100A8	DS\$BRANCH
000100B0	DS\$CVTREG
000100B8	DS\$PARSE
000100C0	DS\$ERRSYS
000100C8	DS\$ERRDEV
000100D0	DS\$ERRHARD
000100D8	DS\$ERRSOFT
000100E0	DS\$PRINTB
000100E8	DS\$PRINTX
000100F0	DS\$PRINTF
000100F8	DS\$PRINTS
00010100	DS\$PRINTSIG
00010108	DS\$ERRPREP
00010110	DS\$SETPRIEXV
00010118	DS\$MAPDBGBLOCK
00010120	DS\$GETBUF
00010128	DS\$RELBUF
00010130	DS\$GETMEM
00010138	DS\$RELMEM
00010140	DS\$MOVVRT
00010148	DS\$MOVPHY
00010150	DS\$MMON
00010158	DS\$MMOFF
00010160	DS\$SETVEC
00010168	DS\$CLRVEC
00010170	DS\$INITSCB
00010178	DS\$SETIPL
00010180	DS\$CHANNEL
00010188	DS\$SETMAP
00010190	DS\$SHOCHAN
00010198	DS\$LOAD
000101A0	DS\$PROBE
000101A8	DS\$ATTACH
000101B0	DS\$HELP
000101B8	DS\$FREEDBGSYM
000101C0	DS\$LOADPCS
000101C8	DS\$GETTERM
000101D0	DS\$SERVICE_DISPATCHER
000101D8	DS\$MEMSIZE
00010200	SY\$QIOW
00010238	SY\$ALLOC
00010248	SY\$ASCTIM
00010250	SY\$ASSIGN

Value	Symbols...
00010258	SYSS\$BINTIM
00010260	SYSS\$CANCEL
00010268	SYSS\$CANTIM
00010298	SYSS\$CLREF
000102D8	SYSS\$DALLOC
000102E0	SYSS\$DASSGN
00010350	SYSS\$FAO
00010358	SYSS\$FAOL
00010378	SYSS\$GETTIM
000103A0	SYSS\$LKWSET
000103B8	SYSS\$NUMTIM
000103C8	SYSS\$QIO
000103D0	SYSS\$READEF
000103F8	SYSS\$SETAST
00010400	SYSS\$SETEF
00010420	SYSS\$SETIMR
00010428	SYSS\$SETPRI
00010430	SYSS\$SETPRT
00010438	SYSS\$SETRWM
00010448	SYSS\$SETSWM
00010460	SYSS\$ULKPAG
00010468	SYSS\$ULWSET
00010470	SYSS\$UNWIND
00010478	SYSS\$WAITFR
00010488	SYSS\$WFLAND
00010490	SYSS\$WFLOR
000104C8	SYSS\$GETCHN
00010580	SYSS\$GET
00010590	SYSS\$READ
000105B8	SYSS\$CLOSE
000105C0	SYSS\$CONNECT
000105D0	SYSS\$DISCONNECT
00010608	SYSS\$OPEN

Key for special characters above:

!	*	-	Undefined	!
!	U	-	Universal	!
!	R	-	Relocatable	!
!	X	-	External	!
!	V	-	Vectored	!

! Image Synopsis !

Virtual memory allocated: 00000200 000077FF 00007600 (30208. bytes, 59. pages)
Stack size: 0. pages
Image binary virtual block limits: 1. 59. (59. blocks)
Image name and identification: ECKAM 03-02
Number of files: 2.
Number of modules: 2.
Number of program sections: 26.
Number of global symbols: 350.
Number of cross references: 450.
Number of image sections: 1.
Image type: SYSTEM.
Map format: FULL WITH CROSS REFERENCE in file DISK\$DIAGPACK04:[ZECCHINO.ECKAM]ECKAM.MAP;106
Estimated map length: 123. blocks

! Link Run Statistics !

Performance Indicators	Page Faults	CPU Time	Elapsed Time
-----	-----	-----	-----
Command processing:	105	00:00:00.23	00:00:00.51
Pass 1:	117	00:00:00.57	00:00:01.25
Allocation/Relocation:	45	00:00:00.22	00:00:00.89
Pass 2:	74	00:00:01.21	00:00:03.07
Map data after object module synopsis:	8	00:00:01.58	00:00:02.57
Symbol table output:	0	00:00:00.02	00:00:00.11
Total run values:	349	00:00:03.83	00:00:08.40

Using a working set limited to 856 pages and 61 pages of data storage (excluding image)

Total number object records read (both passes): 326
of which 0 were in libraries and 4 were DEBUG data records containing 573 bytes

Number of modules extracted explicitly = 0
with 0 extracted to resolve undefined symbols

0 library searches were for symbols not in the library searched

A total of 0 global symbol table records was written

LINK/CONTIG/SYST=xX200/EXE=ECKAM/MAP=ECKAM ECKAM0+ECKAM1/FULL/CROSS