EK-8600S-MM-004

VAX 8600/8650 System

Fault Isolation Manual

Prepared by Educational Services
of Digital Equipment Corporation

1st Edition, October 1985
2nd Edition, December 1985
3rd Edition, April 1987
4th Edition, November 1987

The material in this manual is for informational purposes and is subject to change without notice.

Digital Equipment Corporation assumes no responsibility for any errors that may appear in this manual.

Printed in U.S.A.

The manuscript for this book was created on a VAX-11/780 system using RUNOFF. The book was produced by Educational Services Development and Publishing in Marlboro, MA.

CONTENTS

Appendix L SBIA Stack Frame Organization and Bit Descriptions

PREFACE

The purpose of this manual is to assist the field service engineer in isolating elusive or intermittent faults in the VAX 8600/8650 kernel system.

Revision 3 of the Fault Isolation Manual reflects Error Handling Microcode Revision 1 with the addition of new EHM flowcharts and description, and the addition of new bits to the EBCS, EDPSR, EHMSTS, and IBESR registers. Revision 4 reflects the new Machine Check Handler.

The keep alive flowchart was corrected to reflect console software, and the snap file description was corrected to reflect the addition of 79 longwords to the snapshot.

Chapter 1 provides an overview of how the 8600/8650 detects and processes errors. This is the "big picture". It explains that:

- The error detection networks in each of the boxes, including the SBIA, report to the EBox.
- The EBox arbitrates and prioritizes the errors and generates a micro trap vector address.
- The Error Handling Microcode (EHM) captures the state of the machine, builds a stack frame, clears the error condition, rolls back the PCs, pushes the stack frame onto the interrupt stack, and calls the VMS machine check handler.
- VMS processes the error, queues the stack to be written to ERRLOG.SYS, and determines whether to REI, or bugcheck.
- Spear reads the event file and, depending which function is selected, computes system availability, summarizes the contents of the event file, translates specific entries, or analyzes the cause of specific entries.

Chapter 2 consists of a flowchart and a catalogue of error scenarios for the EBox, FBox, IBox, and MBox. The flowcharts are used to manually analyze stack frames. The result is a pointer to an error scenario. The scenario describes the nature of the error, provides a typical error signature, and suggests a probable cause.

Chapter 3 contains error scenarios for the SBIA and SBI errors. It will help the field engineer analyze and identify the most probable cause of SBIA error entries.

Chapter 4 contains a series of tables listing the console error messages.

Chapter 5 describes the system event file and how to translate the entries in the system event file using the error record formatter (ERF) or SPEAR.

There are 12 appendices:

A. Appendix A contains a flowchart of the Error Handling Microcode. It shows, in detail, how the EHM builds the stack, clears the error, rolls back the PCs, pushes the information on the interrupt stack, and calls VMS. The flow is detailed enough to allow the field engineer to use it to troubleshoot double error conditions.

B. Appendix B contains a brief description and a flowchart that describes how the console corrects control store parity errors.

C. Appendix C contains a flowchart of the VMS machine check handler. It shows the logging and the REI/Bugcheck decision making process.

D. Appendix D contains flows describing the keep alive fail and console ECC correction process. It includes the generation of snap files.

E. Appendix E contains detailed functional diagrams that describe the EBox error arbitration, prioritizing, and trap vector generation logic.

F. Appendix F contains a set of functional diagrams that describe each error detection network in the EBox.

G. Appendix G is the same as Appendix E except the diagrams pertain to the FBox.

H. Appendix H is the same as Appendix E except the diagrams pertain to the IBox.

I. Appendix I is the same as Appendix E except the diagrams pertain to the MBox.

J. Appendix J is the same as Appendix E except the diagrams pertain to the SBIA.

K. Appendix K contains bit definitions for the registers that make up the machine check stack frame.

L. Appendix L contains bit definitions for the registers that make up the SBIA stack frame

CHAPTER 1

SYSTEM FAULT ISOLATION OVERVIEW

INTRODUCTION

Both the VAX8600 and the VAX8650 are classified as a fault tolerant Processors. That is, they are designed to recover from most intermittent errors with minimum impact on system performance. The following examples illustrate the degree of error detection and recovery capability of these Processors.

- The Memory Array contents are ECC protected. If a single bit error is detected during an Array Read, the data is corrected (on the fly) before it is cached. Thus future references will produce correct data.

- Cache data is byte parity protected with ECC backup. If a Cache data parity error is detected the MBox will execute a Cache Data Correction Cycle. Thus when the operation is retried the data will be correct.

- Extensive parity protection has been designed into the processor data paths. In most cases if a data path parity error is detected, the processor is stalled, the error condition cleared, and the PCs and RLog rolled back to a state prior to the occurrence of the error. Thus, when the processor is restarted the instruction will be re-executed successfully.

- Control Store and Dispatch RAMs are ECC protected. If a Control Store or Dispatch RAM parity error is detected the processor is stalled and the Console is interrupted. The Console will read the bad RAM word, look up the ECC for that word in a table, perform ECC correction, write the word back into the RAM and restart the processor. Thus, in most cases, the process will successfully continue.

- If a short-term power failure occurs (10 minutes or less) the system provides battery back up power to the Array Refresh Circuitry. Thus when main power is restored the system will successfully continue.

FAULT DETECTION AND REPORTING OVERVIEW

Figure 1-1 describes the overall organization of error detection, error handling, and error reporting for both the VAX8600 and VAX8650 Processors. It shows the relationship between the Error Detection Networks, the Interrupt and Exception Arbitration Logic, the Error Handling Microcode, the VMS Machine Check Handler, the System Event File, and SPEAR.

In addition, Figure 1-1 shows that the Console is involved in RAM correction and Keep Alive Fail detection. The following paragraphs provide a brief description of each function.

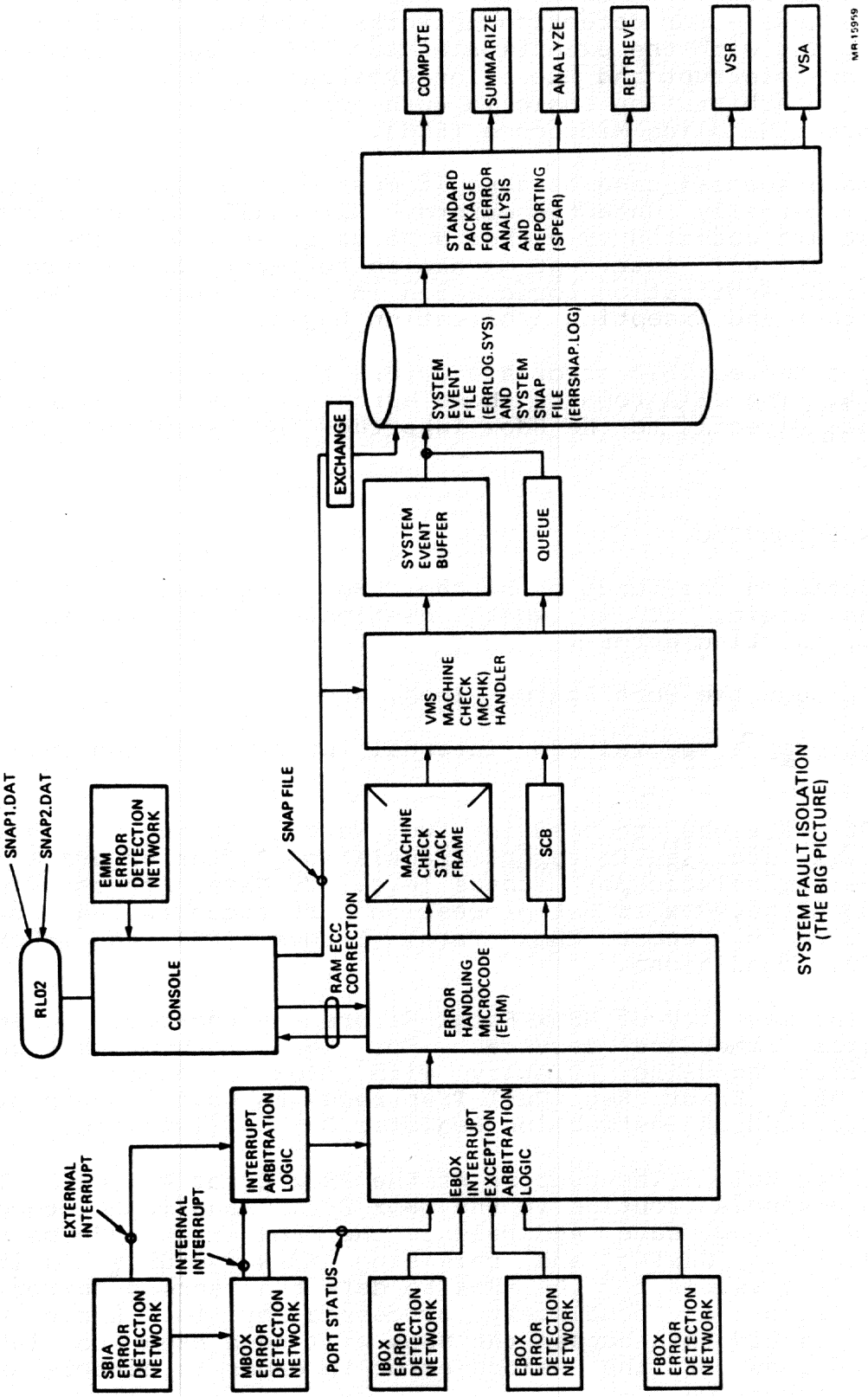


FIGURE 1-1 FAULT DETECTION AND REPORTING ORGANIZATION

ERROR DETECTION NETWORKS

As shown in Figure 1-1 each box has a separate error detection network. These error detection networks constantly monitor for error conditions, and with the exception of the SBIA, report errors directly to the EBox Interrupt and Exception Arbitration Logic. The Interrupt and Exception Arbitration Logic in turn generates a micro trap vector into the Error Handling Microcode (EHM).

The SBIA is a special case because it must monitor for externally as well as internally detected errors. Externally detected errors are errors that are detected by either a Nexus or an I/O device. The SBIA uses the external interrupt mechanism to report these errors to the EBox Interrupt Arbitration Logic which in turn reports them to the EBox Interrupt and Exception Arbitration Logic.

Internally detected SBIA error are errors that are detected internal to the SBIA. The SBIA reports these errors to the MBox which in turn, reports them directly to the EBox Interrupt and Exception Arbitration Logic.

MBOX ERROR REPORTING

Before describing the function of the EBox Interrupt and Exception Arbitration Logic, it is worth mentioning that the MBox has two methods of reporting errors:

- Through the Port Status Lines, and
- Through a special MBox Internal Interrupt Mechanism.

The Port Status Lines are used in two ways. During an MBox Port Request they are used to report Translation Buffer errors as well as other non-error related port status (e.g., TB Miss, Access Violation, etc.). When the MBox is not processing port requests the port status lines are used to report MBox Fatal Errors (FE) and MBox Error Register Full conditions.

MBox FE - The majority of MBox Fatal Errors are non-recoverable errors that require immediate service. For this reason they share the highest error handling priority with EBox detected errors. In contrast, MBox Error Reg Full Trap requests have a lower priority. They are serviced at Instruction Register Decode (IRD) time.

MBox Error Reg Full - The purpose of the MBox Error Reg Full Trap is to trap to a special routine in the EBox Error Handling Microcode that is designed to read, save, and release the MBox Error Address Register (MEAR) quickly. Saving and releasing MEAR quickly is important because it is possible for the MBox to detect a second error before the first error is completely processed by the Error Handling Microcode. Should this happen the MBox will need MEAR to latch the physical address of the second error in order to perform Array and Cache data correction.

MBOX INTERRUPTS

The MBox uses an Internal Interrupt mechanism to report all other non-fatal MBox errors. Like MBox Error Reg Full Traps, MBox Interrupts are serviced at Instruction Register Decode (IRD) time and therefore have a relatively low priority. However, because the RLog and PCs can be unwound and the operation retried, most of these errors will be recoverable.

EBOX INTERRUPT and EXCEPTION ARBITRATION LOGIC

This is a central focal point. All errors, with the exception of MBox Control Store and EBox Control Store parity errors report to the EBox Interrupt and Exception Arbitration Logic. This logic prioritizes the errors and generates a Micro-trap Vector Address. The Micro-trap Vector Address is the starting address of a special EBox Microcode routine that is designed specifically to handle error conditions.

ERROR HANDLING MICROCODE

The special Micro routine mentioned above is referred to as the Error Handling Microcode (EHM). The Error Handling Microcode reads the state of 25 major CPU Control and Status Registers and puts the result in the EBox Scratch Pad RAM; locations 17 through 2F. This data is referred to as the Machine Check Stack Frame.

In the process of building a Machine Check Stack Frame the Error Handling Microcode also clears the error condition and rolls back the RLog and PCs. This is done in preparation for retry of the failed operation. Finally the Error Handling Microcode sets up the VMS Machine Check Vector Address (SCBB+4) and calls the Interrupt/Exception Micro-routine. The Interrupt/Exception Micro-routine (not shown in Figure 1-1) pushes the Machine Check Stack Frame onto the Interrupt Stack and calls the VMS Machine Check Handler.

VMS MACHINE CHECK HANDLER

The VMS Machine Check Handler pops the Machine Check Stack Frame off the Interrupt Stack and puts it in a System Event (memory) Buffer. It then notifies the ERRFMT Process which, in turn, appends the buffer to the System Event File (ERRLOG.SYS).

In the process of building and queuing the System Event Buffer, the Machine Check Handler also checks the error rate and the severity of the error. If the error rate is excessive, or if the error condition is severe (i.e., non recoverable) then the Machine Check Handler will Bugcheck the process (i.e., the User or the System). Otherwise the Machine Check Handler will execute a REI and the system will retry the failed operation.

SPEAR (Standard Package for Error Analysis and Reporting)

SPEAR is a maintenance tool specifically designed to help Field Engineers sort and analyze the contents of System Event Files. There are two versions of Spear: SPEAR Basic and SPEAR Extended.

SPEAR Basic

SPEAR Basic is available on all sites that have a DEC Maintenance Contract. This version of SPEAR consists of five programs:

- Instruct - Instruct is a Computer Based Instructional Program that is designed to help new uses learn to use the SPEAR Library Programs. In addition to explaining the SPEAR Programs, Instruct also describes the organization of System Event Files and includes a review of some of the most common Troubleshooting Approaches.

- Compute - Compute is designed to use the contents of the System Event File to calculate system availability and effectiveness. The Compute report can be used (in part) to determine if the system is approaching the point where corrective maintenance will soon be required.

- Summarize - Summarize is designed to summarize the contents of the System Event File. The Summarize report can be used to determine whether the CPU or one of the I/O subsystems needs further investigation.

- Retrieve - Retrieve is a bit-to-text translator. It is designed to extract specific entries from the System Event File and produce either a Brief or Full translation of the event. This information can be used to investigate the cause of CPU and I/O failures.

- VSR (Venus Snap File Report Builder) - This program was added to the Basic SPEAR Library specifically to support VAX 8600/8650 Systems. Like Retrieve, VSR is a bit-to-text translator. VSR, however, is designed to translate SNAP Files. A SNAP file is a file built by the Console as a result of a Keep Alive Fail condition. Both SNAP Files and the Keep Alive Fail mechanism are described later.

SPEAR Extended

SPEAR Extended is only available at Remote Diagnosis Centers. In addition to the programs that are available in SPEAR Basic, Spear Extended includes:

- Analyze - Analyze is designed to analyze the contents of large System Event Files and identify the most probable cause of certain failures. Basically Analyze evaluates the events in a System Event File against a set of If-Then Isolation Theories. If the Events support a theory then the theory is displayed for consideration by the Engineer at the RD center.

- VSA (Venus Snap File Analysis Builder) - VSA is similar to Analyze except it is designed to analyze the contents of System SNAP Files.

KEEP ALIVE FAIL

The CPU has two Keep Alive Fail (KAF) Mechanisms. One KAF mechanism is used by the Console to monitor the operation of the CPU. Each time the EBox enters Instruction Register Decode (IRD) Time a signal is generated that sets a status bit in the Console. The Console checks the state of this bit every 300 milliseconds. If the bit is set, the system is operating normally. The Console then clears the bit and continues operation.

However, if the console finds that the EBox has failed to set the bit, it declares a Keep Alive Fail Condition. (i.e., the EBox failed to enter IRD Time during the last 300 Milliseconds.) At this point the CPU is considered dead.

As a result of the KAF condition the Console enters a special KAF Routine. The main purpose of the routine is to determine if either of the SNAP Files (SNAP1.DAT or SNAP2.DAT) are available to store SNAPSHOT Data. If they are, zthe KAF Routine will read the state of the CPU and put the result in a SNAP File buffer. The routine begins by reading each SDB Visibility Channel. The routine then goes on to read the Console Control and Status registers, the EMM Control and Status Registers, the EBox Scratch Pad RAM contents, the Internal Processor Registers, the PAMM contents, the last 64 longwords pushed on the Interrupt Stack, and the Control and Status Registers for SBIA0 and SBIA1.

When the KAF routine is finished building the buffer it verifies the contents of all loadable RAMs and then writes the SNAP File buffer out to the RL02. Finally, the KAF routine will attempt to re-boot the system.

If the re-boot is successful the SNAP File will be copied to the VMS Directory "SYS$SYSROOT:[SYSERR] and named ERRSNP.LOG;n If, however, the console was unable to re-boot the system then a second KAF condition will occur which will result in the generation of a second SNAP File (SNAP2.DAT). At this point the Console will loop waiting for manual intervention.

The other Keep Alive Fail mechanism is used by VMS to monitor the operation of the Console. The Console is programmed to update the Buffered Time Of Year (BTOY) Register every 10 milliseconds. Every 90 seconds VMS reads this register via the Time Of Day Register (TODR). If it finds that the register has been updated then all is OK. If, however, VMS determines that the register has not been updated it makes an entry in the System Event File and attempts to restart the Console.

CHAPTER 2

MANUAL MACHINE CHECK STACK FRAME ANALYSIS

MANUAL MACHINE CHECK STACK FRAME ANALYSIS

OVERVIEW

This Chapter is designed to help you analyze the contents of Machine Check Stack Frames. It assumes that you have a "valid" Machine Check Stack Frame in hand, and that you want to see which error scenario this method of analysis will lead to.

HOW TO USE THIS CHAPTER - First translate the EBox Control and Status Register (EBCS). Then, based on the contents of EBCS, follow the flow chart illustrated in Figure 2-2. The flow chart will eventually lead you to a page number that describes a typical error scenario that would produce a Stack Frame similar to the one you are analyzing.

Review the scenario. It describes the most common conditions under which that type of error will occur. It also suggests the most probable cause of the error; first at the module level and than at the component FRU level. (E.g., RAM and MCA callout.)

NOTE

In addition to the flow chart, Tables 1 through 4 may be used as a quick reference to the Error scenarios.

Table 1 Index of MBox Error Scenarios

Table 2 Index of EBox Error Scenarios

Table 3 Index of IBox Error Scenarios

Table 4 Index of FBox Error Scenarios

STANDARD EHM, CSL, and VMS ACTIONS

A lot of effort has gone into standardizing the way the Error Handling Microcode, the Console, and the VMS Machine Check Handler respond to CPU detected errors. This Section describes the standard role played by each. Special cases will be explained in the individual Error Scenarios.

EHM ACTION (Standard): Unless specified otherwise in the individual Error Scenarios the Error Handling Microcode will:

1. Stop IBox references.
2. Check and set the EHM ENTERED double-error trap mechanism.
3. Capture the state of the CPU and build a Stack Frame.
4. Clear the error condition and, if necessary correct GPR PEs.
5. Roll back the PCs in preparation for an instruction retry.
6. Load the VMQ with SCBB+4 (the MCHK Handler starting address).
7. Check and set the VMS ENTERED double-error trap mechanism.
8. Clear EHM ENTERED.
9. Call the Exception and Interrupt Handler (microcode).

The Interrupt and Exception Handler will:

1. Raise the IPL to 1F.
2. Push the Stack Frame onto the Interrupt Stack (memory).
3. Enable IBox References.
4. Start VMS at the MCHK Handler (SCBB+4).

NOTE

For specific details refer to the EHM Flows. (See Appendix A)

VMS ACTION (Standard): Unless specified otherwise in the individual Error Scenarios the VMS Machine Check Handler will:

1. Enter the Reason Code in the copy of EHMSTS <03:00>.
2. Check the error rate (See Table 5).
3. Check for Fatal Errors and set the MCHK Abort bit if approprate.
4. Transfer the Stack Frame to an Event Buffer.
5. Setup the buffer to be appended to SYS$SYSROOT:[SYSERR]:ERRLOG.SYS.
6. Check for abort bits.
7. Either REI or Bugcheck (Non-Fatal/User or Fatal/System.

If VMS decides to Bugcheck the User it will set up an Exception on the User Stack and execute an REI. If, however, VMS decides to Bugcheck the System it will print the contents of the Interrupt Stack on the Console, write the contents of main memory, the two Error Log Buffers, and the reason for the crash to a file called SYS$SYSTEM:SYSDUMP.DMP and then Halt. This will result in the Console detecting a Keep Alive Fail condition.

The Console will Snap Shot the state of the system and attempt to reboot VMS. If the reboot process is successful VMS will copy the file to SYS$SYSROOT:[SYSERR]ERRSNAP.LOG. If the file already exists the version number will be bumped by one. Also during the re-boot process the two Event Buffers saved during the Crash will be appended to the System Event file (SYS$SYSROOT:[SYSERR]ERRLOG.SYS).

NOTE

For specific details refer to the VMS MCK Flows. (See Appendix B)

Table 5 VMS Error Rate Thresholds

Error	Excessive Rate and Action Taken
EBox	3 Errors within 100 Milliseconds - Bugcheck
IBox	3 Errors within 100 Milliseconds - Bugcheck
MBox (FE)	2 Errors within 20 Milliseconds - Bugcheck
TB	3 Errors within 100 Milliseconds - Fatal System Bugcheck
MBox (1D)	3 (Non-Array/Non-Cache) MBox Errors within 100 Milliseconds - Bugcheck

Table 5 VMS Error Rate Thresholds (cont.)

Error	Excessive Rate and Action Taken
Cache	3 Errors (same Cache) within 100 Milliseconds - Turn off offending cache and print message on CTY
FBox	3 Errors within 100 Milliseconds - Turn FBox off and print message on CTY.

CONSOLE ACTION (RAM ECC Correction): When called (interrupted) to perform Control Store or Dispatch RAM correction the console proceeds as follows:

IBox DRAM and EBox, IBox, and FBox Control Store Correction:

1. Read the bad microword address (via the SDB)
2. Read the bad microword (via the SDB)
3. Calculate an ECC character for the bad microword
4. Get the correct ECC character for that microword
5. Exclusive "or" the two ECC characters (See note)
6. Correct the bad bit in the microword
7. Write the corrected word out to the RAM
8. Read and verify that the error was corrected
9. Return control to the Error Handling Microcode
10. The EHM will call the CSM to read four bytes of status from the console and put them in EScratch location T1D (CSES).

FBox DRAM Correction - The Console proceeds as outlined above but, because the FDRAM address is not latched at the time of error, the Console will reload and verify the entire RAM instead of just a single microword.

MBox Control Store Correction - The Console proceeds as outlined above but, because the MBox microwords are executed before they are checked, the Console will not attempt to recover from an MBox Control Store Parity Error. Instead it will leave the system hung until a KAF occurs.

NOTE

If the console is unable to correct the RAM parity error (i.e., multiple bit error) it will declare a KAF Condition and then re-initialize the CPU.

SAMPLE CTY OUTPUT FROM A FATAL BUG CHECK

The following is sample output from a Fatal-Bug-Check CTY Dump. Its purpose is to help you locate Machine Check register information when it is displayed on the in the Kernel/Interrupt Stack.

```
* FATAL BUG CHECK, VERSION = V4.1 MACHINECHK, Machine check while ...

CURRENT PROCESS = STARTUP

REGISTER DUMP

    R0 = 7FFE6440
    R1 = 7FFE6300
    R2 = 0000000F
    R3 = 7FF6C09F
    R4 = 7FFE640C
    R5 = 7FFE64B4
    R6 = 00000000
    R7 = 7FF4C748
    R8 = 7FFED052
    R9 = 7FFED25A
    R10= 7FFEDDD4
    R11= 7FFE33DC
    AP = 7FF341FC
    FP = 7FF341E8
    SP = 8049A794
    PC = 802FE3A2
    PSL= 041F0008

KERNEL/INTERRUPT STACK

    8049A79C  00000058  BYTCNT 1
    8049A7A0  40001803  EHMSTS 2
    8049A7A4  7FFE7DF0  EVMQSAV 3
    8049A7A8  00002000  EBCS 4
    8049A7AC  00000000  EDPSR 5
    8049A7B0  00200001  CSLINT 6
    8049A7B4  01006000  IBESR 7
    8049A7B8  00000000  EBXWD1 8
    8049A7BC  00000FE0  EBXWD2 9
    8049A7C0  7FFE643C  IVASAV 10
    8049A7C4  7FF6C3DA  VIBASAV 11
    8049A7C8  7FF6C3D6  ESASAV 12
    8049A7CC  7FF6C3D6  ISASAV 13
    8049A7D0  7FF6C3D6  CPC 14
    8049A7D4  84006004  MSTAT1 15
    8049A7D8  00004F10  MSTAT2 16
    8049A7DC  00060000  MDECC 17
    8049A7E0  04000100  MERG 18
    8049A7E4  00000003  CSHCTL 19
    8049A7E8  0000007C  MEAR 20
    8049A7EC  0000001F  MEDR 21
    8049A7F0  FFFFFFFF  FBXERR 22
    8049A7F4  FFFFFFFF  CSES 23
    8049A7F8  7FF6C3D6  PC 24
    8049A7FC  00C00000  PSL 25
```

MSTAT1 <29:26>

CODE	CYCLE	CODE	CYCLE
0000	NOP	1000	ABUS
0001	READ REG	1001	CP REFILL
0010	WRITE REG	1010	INVAL TB
0011	WRITEBACK	1011	TB CYCLE
0100	ABUS ARRAY WRT	1100	CP ARRAY WRT
0101	DATA CORRECTION	1101	CP WRITE
0110	CLEAR CACHE	1110	CP READ
0111	TB PROBE	1111	ABUS REFILL

MSTAT1 <31:30>

CODE	DESTINATION
00	IBUFFER
01	IMD
10	EMD
11	IBUFFER (LOAD TP)

IBESR <09:08>

CODE	DATA SOURCE
00	IBOX REGISTER SELECT
01	OPERAND SOURCE IS EMD
10	OPERAND SOURCE IS IBUFFER
11	OPERAND SOURCE IS IMD OR ID

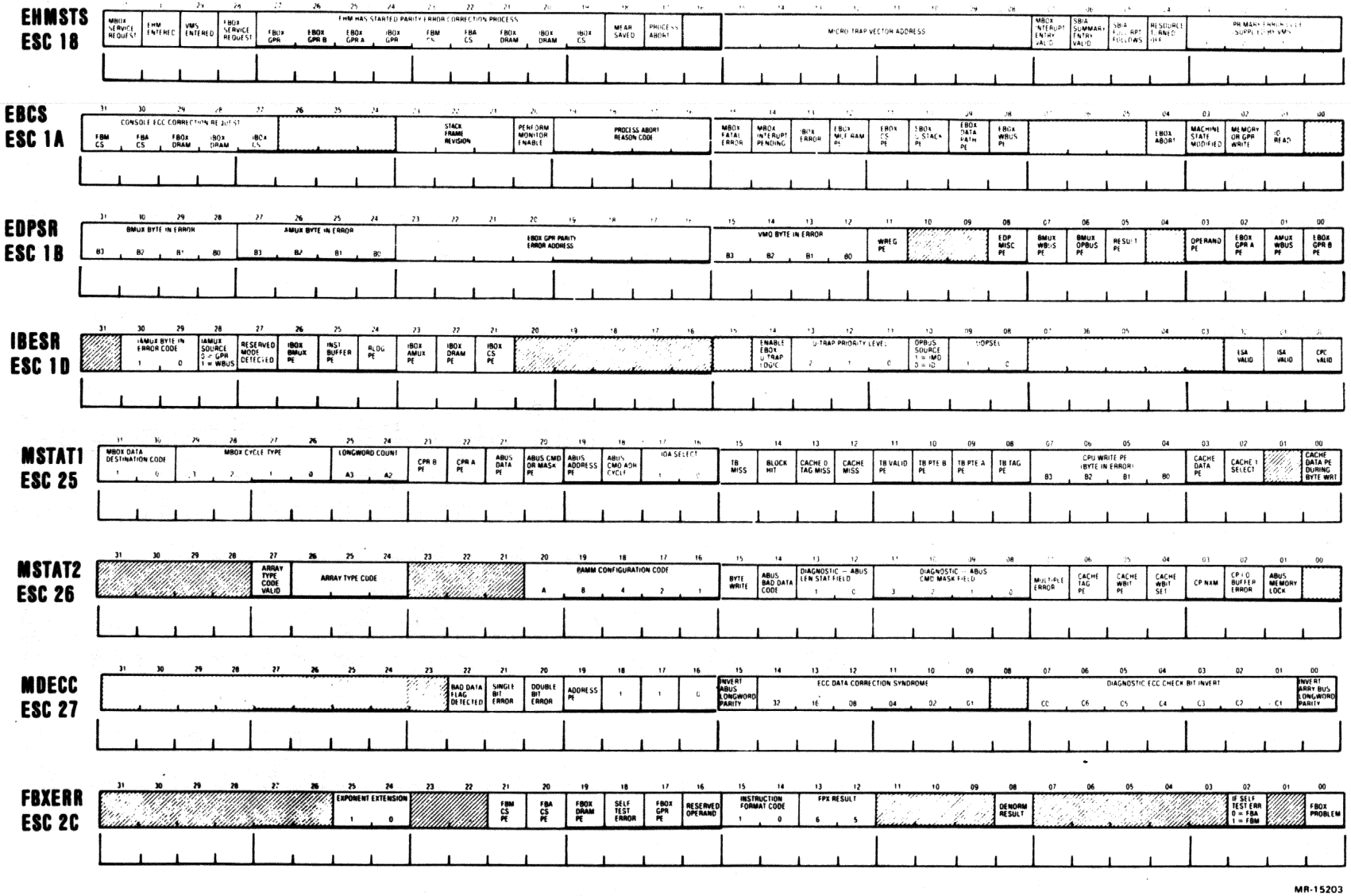
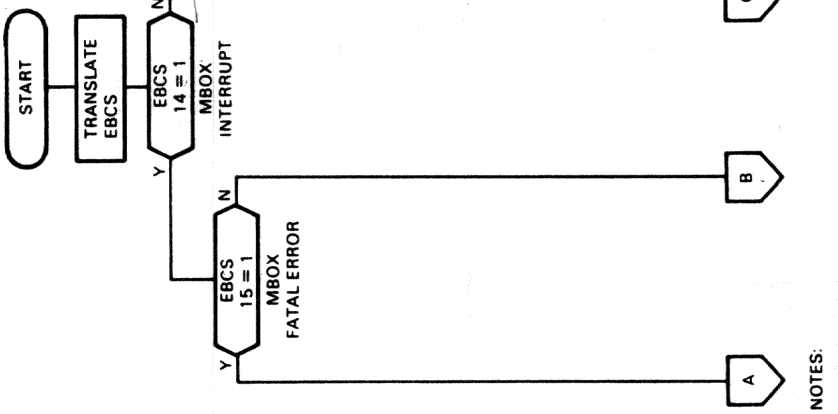


Figure 2-1 Abbreviated Machine Check Stack Frame Worksheet



NOTES:

1. THE MACHINE CHECK STACK FRAM CAN NOT BE ANALYZED USING THIS FLOW CHART.

2. TB ERROR SHOULD BE IGNORED UNLESS THERE ARE NOT OTHER ERROR INDICATIONS IN THE STACK FRAME. SEE ERROR SCENARIO: M01.

Figure 2-2 Machine Check Stack frame Analysis Flow Chart (Part 1 of 6)

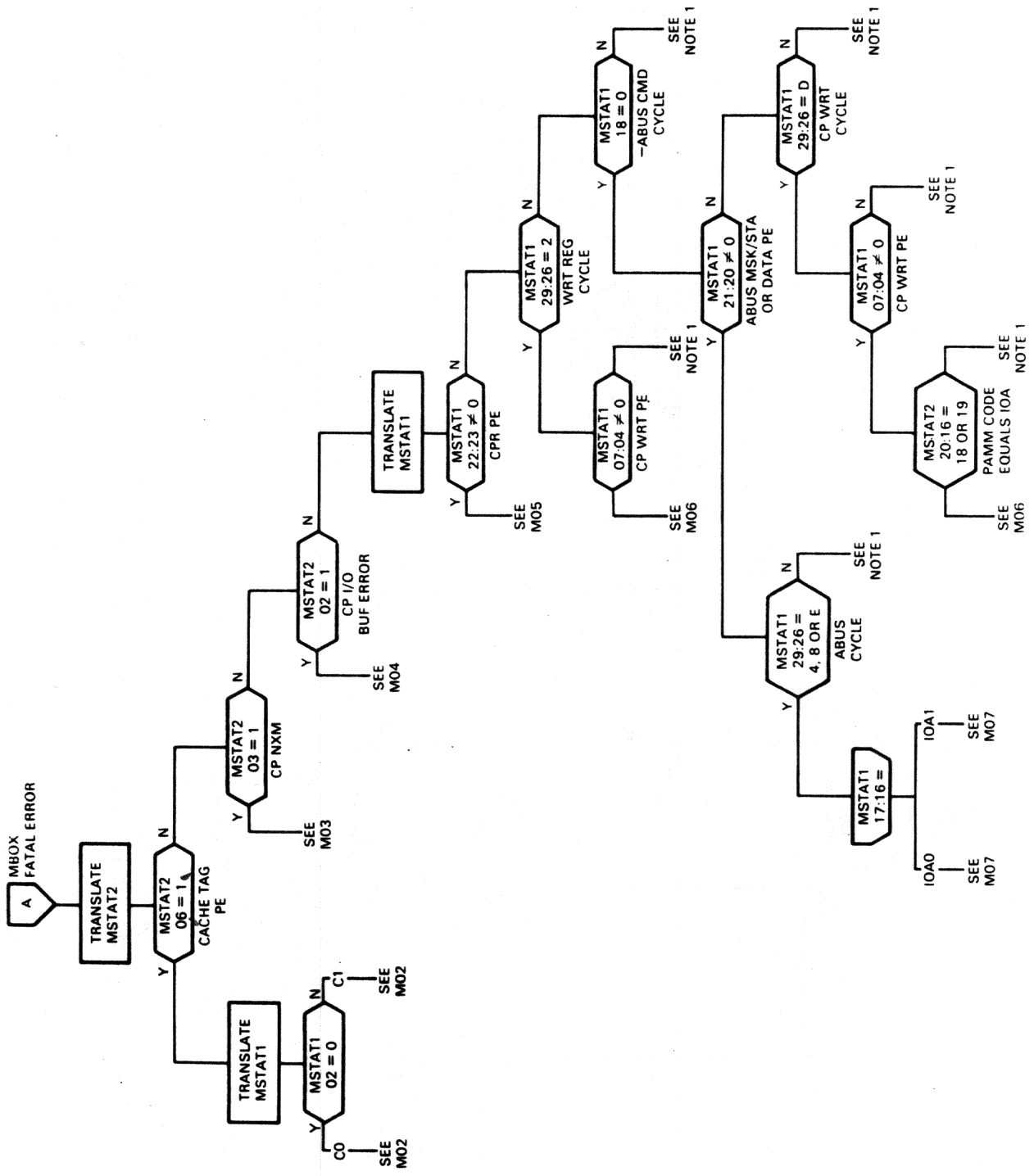
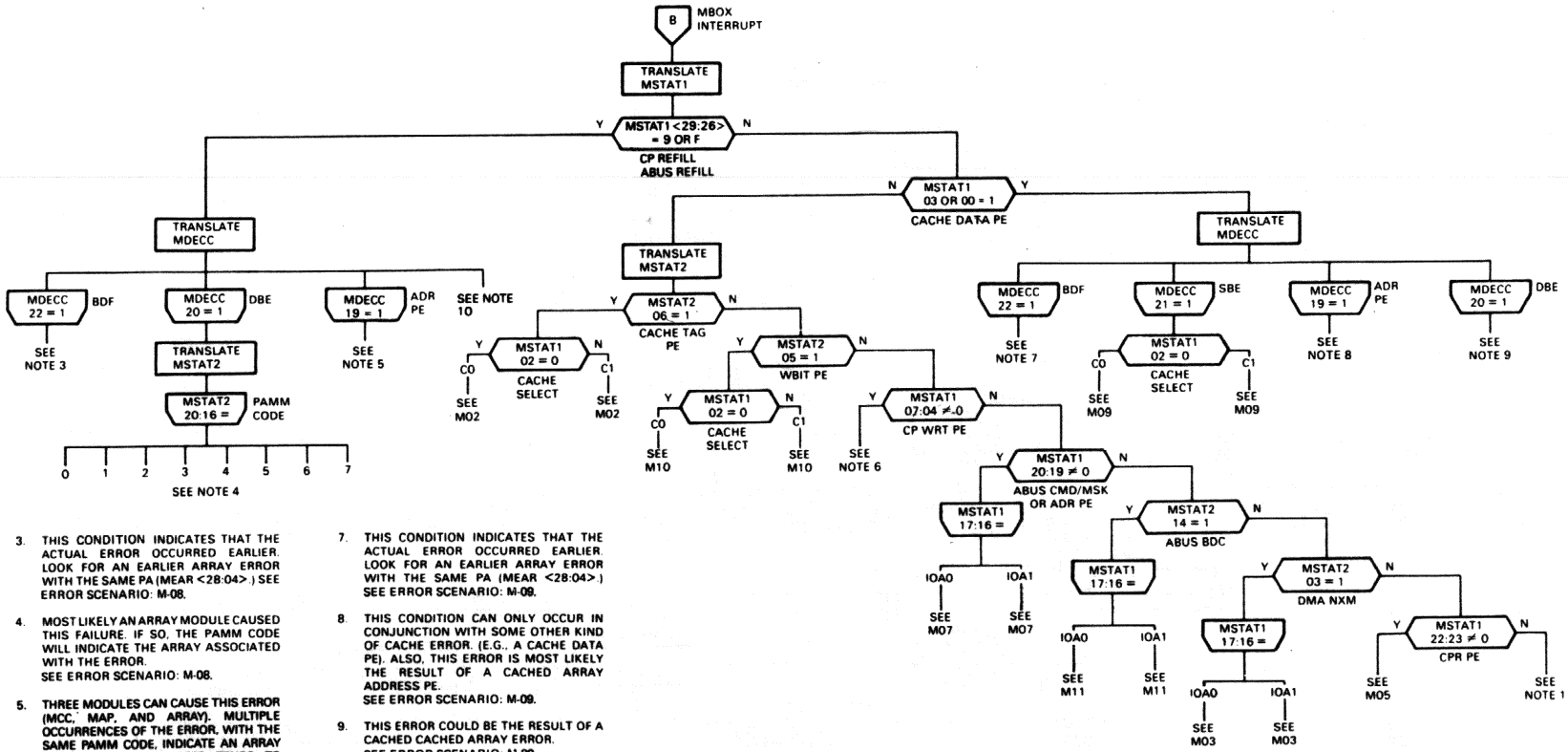


Figure 2-2 Machine Check Stack frame Analysis Flow Chart (Part 2 of 6)



3. THIS CONDITION INDICATES THAT THE ACTUAL ERROR OCCURRED EARLIER. LOOK FOR AN EARLIER ARRAY ERROR WITH THE SAME PA (MEAR <28:04>) SEE ERROR SCENARIO: M-08.

4. MOST LIKELY AN ARRAY MODULE CAUSED THIS FAILURE. IF SO, THE PAMM CODE WILL INDICATE THE ARRAY ASSOCIATED WITH THE ERROR.
SEE ERROR SCENARIO: M-08.

5. THREE MODULES CAN CAUSE THIS ERROR (MCC, MAP, AND ARRAY). MULTIPLE OCCURRENCES OF THE ERROR, WITH THE SAME PAMM CODE, INDICATE AN ARRAY MODULE. HOWEVER, VMS TENDS TO ACCESS THE UPPER ARRAY MORE THAN THE OTHR ARRAYS. THEREFORE, IF THE FAILURE IS ASSOCIATED WITH THE UPPER ARRAY SWAP IT WITH A LOWER ARRAY. IF THE ERROR PERSISTS, CHANCES ARE THE FAULT IS IN THE MCC OR MAP MODULE.
SEE ERROR SCENARIO: M-08.

6. THREE MODULES CAN CAUSE THE ERROR (IBD, IDP, AND MCD).
SEE ERROR SCENARIO: M-06.

7. THIS CONDITION INDICATES THAT THE ACTUAL ERROR OCCURRED EARLIER. LOOK FOR AN EARLIER ARRAY ERROR WITH THE SAME PA (MEAR <28:04>.)
SEE ERROR SCENARIO: M-09.

8. THIS CONDITION CAN ONLY OCCUR IN CONJUNCTION WITH SOME OTHER KIND OF CACHE ERROR. (E.G., A CACHE DATA PE). ALSO, THIS ERROR IS MOST LIKELY THE RESULT OF A CACHED ARRAY ADDRESS PE.
SEE ERROR SCENARIO: M-09.

9. THIS ERROR COULD BE THE RESULT OF A CACHED CACHED ARRAY ERROR.
SEE ERROR SCENARIO: M-09.

10. IF THERE ARE NO ERROR CONDITIONS SET IN MDECC THEN CHANCES ARE BIT 38 IS AT FAULT.

Figure 2-2 Machine Check Stack Frame Analysis Flow Chart (3 of 6)

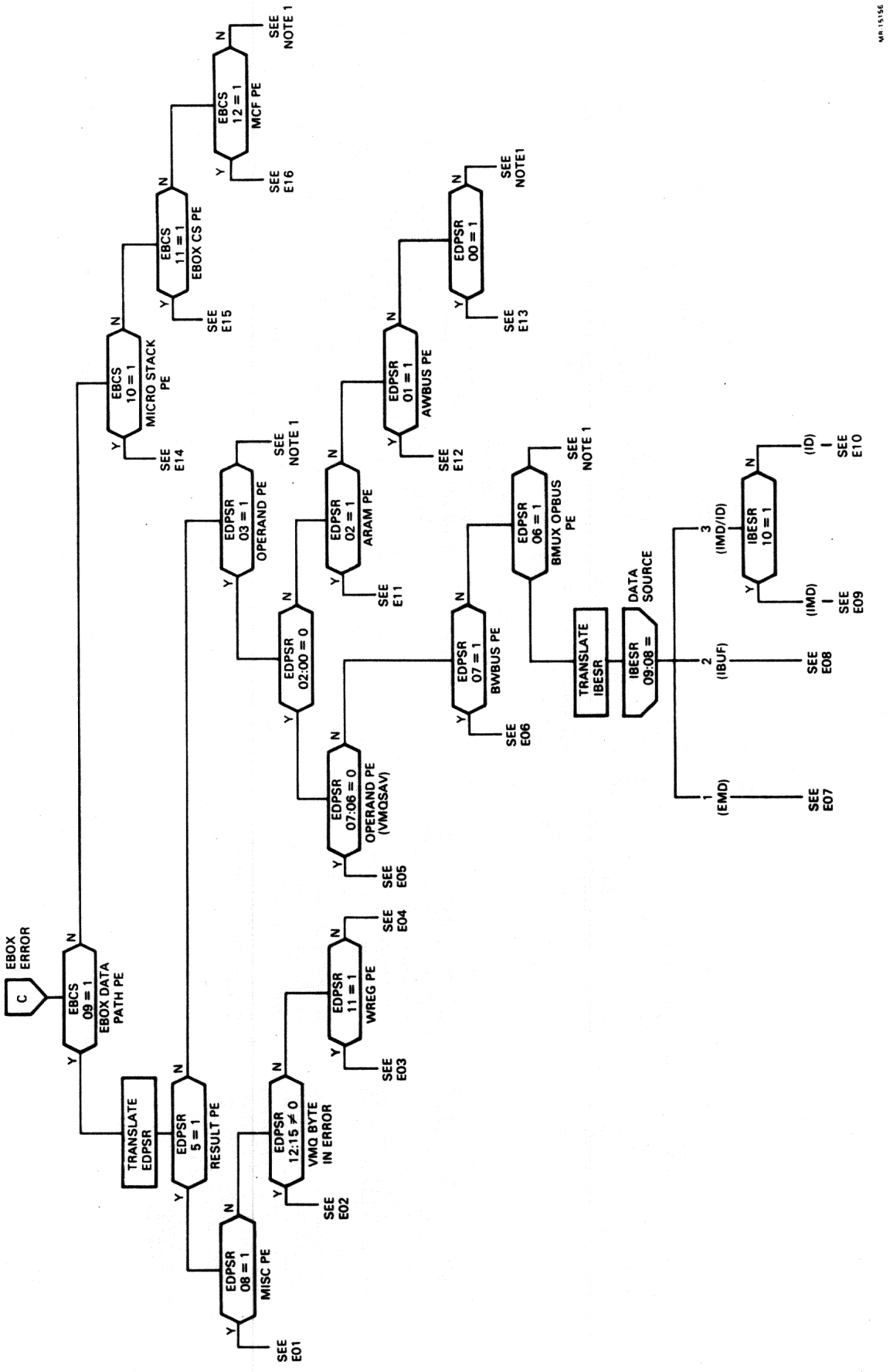


Figure 2-2 Machine Check Stack frame Analysis Flow Chart (Part 4 of 6)

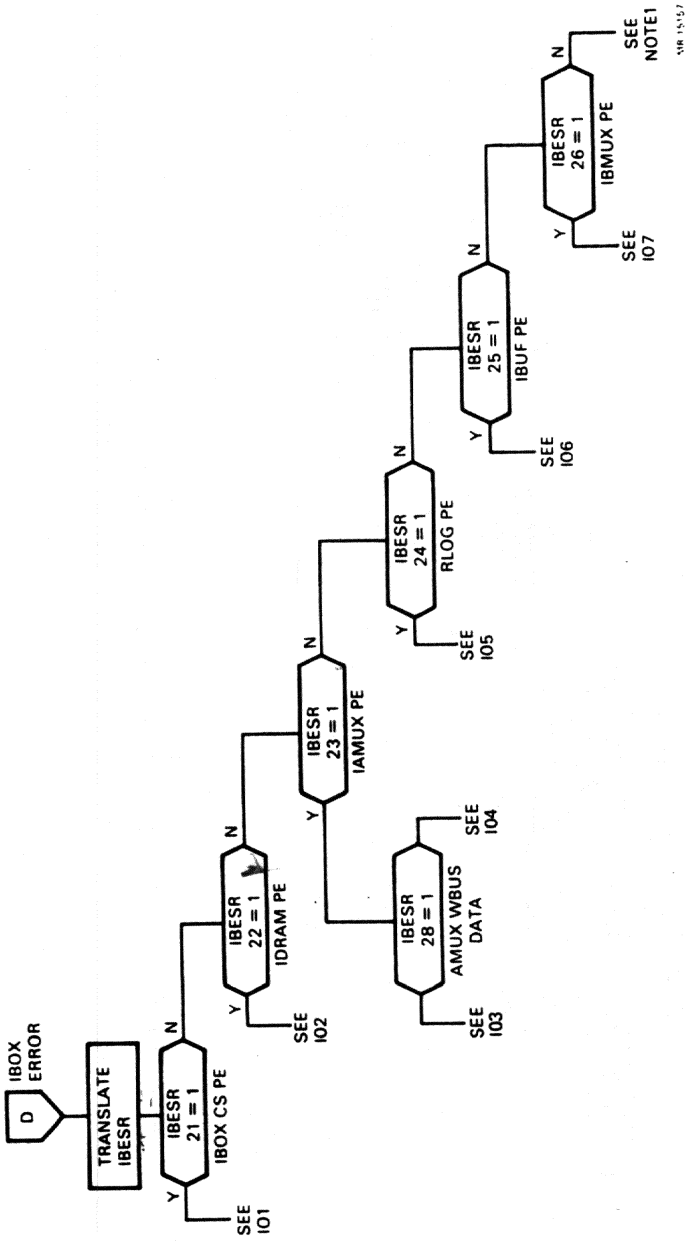


Figure 2-2 Machine Check Stack frame Analysis Flow Chart (Part 5 of 6)

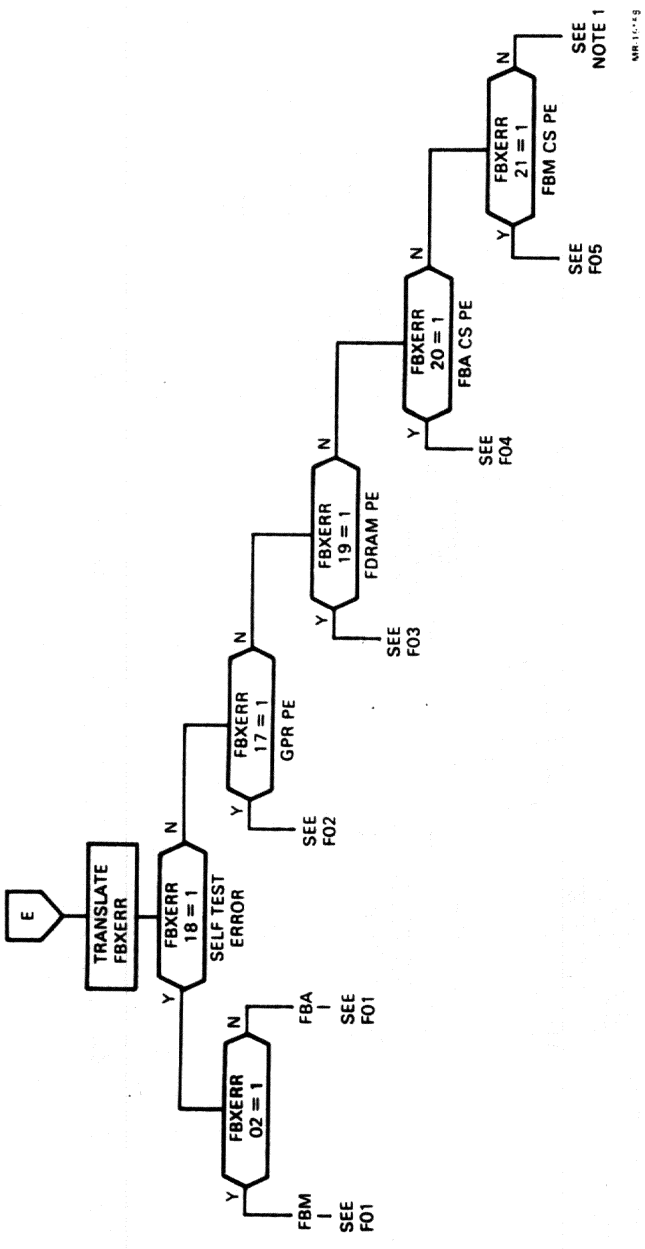


Figure 2-2 Machine Check Stack Frame Analysis Flow Chart (Part 6 of 6)

M-00 MBox Control Store Parity Error

OVERVIEW: The MBox Control Store consists of 256 "seventy-six bit" microwords. The microwords are stored in 20 256X4 RAMs on the MCC Module (MCCD-MCCG). These RAMs are addressed by the MMS MCA at T0 (MCC1).

At T2 of a Fetch Cycle the RAM outputs are loaded into the Microdata MCA's which cascade an accumulating parity check (over eighty bits) from one MCA to the next, culminating in MCCB PARITY D OUT H (MCCA-B) which, if true, latches MCC MBox CS PE H. MCC MBox CS PE H blocks further Microdata MCA clocks and also interrupts the Console via EBE CPU ERR <2:0> H = "7", a code which indicates a problem in the MBox Control Store.

Preserved within the Microdata MCA's are the failing Microword and also the micro PC of that Microword, which is "or'ed" into these MCA's at each fetch. Therefore the Console, after obtaining the contents of these MCA's via the SDB, has what it needs to attempt ECC correction. But unlike its action for errors on other Control Stores, the Console, after obtaining the syndrome and printing a message on the Console terminal, treats this error as a Keep Alive Fail Condition. A Snap Shot is taken, all RAMs are reloaded, etc.

Because some Microbits have VTERMS and other loads on the MCD and MAP Modules these Modules may cause a Microword parity error. In addition to bad Microword parity the following three conditions can cause an MBox Control Store Parity Error indication.

- Address Parity Error - The Microword was fetched from the wrong location.
- uSTACK Overflow - Too many words were pushed on the uSTACK
- uSTACK Underflow - Too many words were popped off the uSTACK

An address parity bit, also contained within the Microword and included in the overall parity bit calculation, is compared by the MMS MCA against parity calculated over the uPC from which the Microword was fetched. A mismatch results in a MBox CS PE.

If the three deep uSTACK in MMS MCA overflows or underflows, a CS PE also results; however the Microword frozen will be one past the one which issued the fatal stack command. The frozen uPC must be consulted to determine the uFLOW. Note that any error detected on an incoming DMA COMMAND asserts MCCM DMA ERR L which forces a pop of the uSTACK via MCCB DLY1 FORCE FF H.

These three errors "or" out of the MMS MCA as MCC1 STACK ERR H, a visibility bit captured in the SNAP FILE.

MANUAL MACHINE CHECK STACK FRAME ANALYSIS

ERROR SIGNATURE:

```
CSES<2:0>     =  "7" or MBox CS PE
CSES<28:16>   =  CS Error Address
CSES<15:8>    =  Syndrome (for SBE ='s phys. MCA bit # +1)
CSES<31>      =  Uncorrectable Error (Currently the Console
                 always sets this bit for MBox CS PE)
MCC1 STACK ERR H
```

PROBABLE CAUSE:

Module	Probability	Components
L0220/L0230/MCC	High	RAMS (see Table 1)
L0204/MCD	Low	VTerms
L0205/MAP	Low	VTerms

NOTE

Of the 80 MCA bits that are parity checked, 76 are RAM outputs. Syndromes, however, exist for all 88 bits in the MCA Shift Path; the extra 8 bits in the shift path hold the uPC of the frozen Microword.

Table 1 MBox Control Store RAM Callout

Syndrome	ULD	Phy	Signal Name	RAM
1	00	00	MCCG U NEXT ADR 0	E21
2	01	01	MCCG U NEXT ADR 1	E30
17	02	22	MCCF U NEXT ADR 2	E20
18	03	23	MCCF U NEXT ADR 3	E36
43	04	66	MCCF U NEXT ADR 4	E127
44	05	67	MCCF U NEXT ADR 5	E140
2D	06	44	MCCG U NEXT ADR 6	E96
2E	07	45	-MCCG U NEXT ADR 7	E96
38	08	55	MCCH U ABUS MFORK EN	E106
0D	09	12	MCCG U CLR FLGS	E21
1C	10	27	MCCG U BEN MASK 0	E13
25	11	36	MCCG U BEN MASK 1	E13
1E	12	29	MCCG U BEN MASK 2	E20
0E	13	13	MCCG U BEN MASK 3	E20
21	14	32	MCCG U BEN SEL 0	E10
23	15	34	MCCG U BEN SEL 1	E10
2A	16	41	MCCG U BEN SEL 2	E10
54	17	83	MCCH U FORCE ABS XFR	E130
11	18	16	-MCCH U PA LAT LD	E31
55	19	84	-MCCD U SEL CACHE	E130

Table 1 MBox Control Store RAM Callout (cont.)

Syndrome	ULD	Phy	Signal Name	RAM
47	20	70	MCCD U FORCE LAST WD	E130
53	21	82	MCCH U MIC PAR	E140
2B	22	42	MCCD U DS MUX SEL 0	E1
26	23	37	MCCD U DS MUX SEL 1	E1
24	24	35	MCCD U DSM VALID	E8
4B	25	74	MCCH U ABUS INH INC	E119
09	26	08	-MCCH U ARY STEP EN	E21
3A	27	57	MCCD U ARY START	E104
51	28	80	MCCF U ARY RD DAT EN	E119
34	29	51	-MCCF U ABUS LAT LD	E117
29	30	40	MCCE U ARY2 HLD	E10
3F	31	61	MCCE U ABUS MBOX OUT	E115
21	32	52	MCCE U ABUS ADR CNTL 0	E115
3D	33	60	MCCE U ABUS ADR CNTL 1	E115
41	34	64	-MCCG U ERR TRAP	E104
0B	35	10	MCCE U ABUS DR EN	E31
3F	36	62	MCCD U MD RES EN	E104
07	37	06	MCCD U DO MUX 0	E21
0C	38	11	MCCD U DO MUX 1	E22
1F	39	30	MCCF U ECC CHECK	E1
42	40	65	MCCG U A1 HLD	E117
15	41	20	MCCF U HLD ARY BUSY	E31
14	42	19	-MCCE U ABUS CLUP	E22
4C	43	75	MCCE U PA MUX SEL 0	E127
4E	44	77	MCCE U PA MUX SEL 1	E127
58	45	87	MCCE U PA MUX SEL 2	E119
4D	46	76	MCCF U INC WD CNT	E130
4A	47	73	MCCE U CLR WD CNT	E140
4F	48	78	MCCF U LD WD CNT	E127
12	49	17	MCCF U SEL WD CNT	E30
05	50	04	-MCCD U CACHE WR EN	E30
0A	51	09	MCCH U EN WR LRU	E30
16	52	21	MCCE U CLR CSH WV	E20
36	53	53	MCCE U ABUS RFL BR	E106
56	54	85	MCCF U MARK	E140
31	55	48	-MCCF U ARB SEL SEL	E117
13	56	18	MCCD U EN BYT MERGE	E22
22	57	33	MCCD U ABUS2 HLD	E8
28	58	39	MCCH CLR BD SEL	E13
1B	59	26	MCCH IOA SEL DIS	E13
37	60	54	MCCE U REG LOAD 0	E115
2C	61	43	MCCE U REG LOAD 1	E31
0E	62	13	-MCCH U EN ARY BUS	E22
50	63	79	MCCF U CB DR EN	E119
20	64	31	-MCCH U ABUS DATA EN	E8

Table 1 MBox Control Store RAM Callout (cont.)

Syndrome	ULD	Phy	Signal Name	RAM
	65		(NO RAM)	
	66		(NO RAM)	
27	67	38	-MCCH U RFL LAT LD	E8
06	68	05	MCCD U CYC TYP 0	E36
08	69	07	MCCD U CYC TYP 1	E36
10	70	15	MCCD U CYC TYP 2	E36
3C	71	59	MCCD U CYC TYP 3	E106
	72		(NO RAM)	
	73		(NO RAM)	
39	74	56	MCCG U STACK CTRL 0	E106
33	75	50	MCCG U STACK CTRL 1	E96
40	76	63	MCCF U IOA SEL 0	E117
3B	77	58	MCCE U IOA SEL 1	E96
1D	78	28	MCCE U RES CORR REQ	E1
32	79	49	MCCH U ADR PAR	E104

EHM ACTION: None

CSL ACTION: The Console displays the syndrome on the terminal and then declares a Keep Alive Fail Condition.

VMS ACTION: When VMS is rebooted it obtains the Snap File from the Console and logs it.

M-01 MBox TB Parity Error

OVERVIEW: The Translation Buffer is protected by four parity bits. The TB Valid Bit (which reflects the PTE Valid Bit in the memory page table) is protected by one parity bit. The PTE (which is divided into two parts: PTE B and PTE A) is protected by two parity bits. PTE B (which includes PA <24:09>) is protected by one bit; and PTE A (which includes (PA <29:25>) is protected by the other. The Modify bit, and the Protection Code <D:A> bits are protected by the third parity bit. Finally, the TB Tag (which contains VA <30:17>) is protected by the fourth parity bit.

If a TB parity error is detected during a CPU port request, the MBox will return a Port Status code of "8" (TB Parity Error) to the EBOX. If the MBox was processing an EBox Port request when the TB Parity Error was detected an immediate trap to the EHM will take place. Otherwise, the trap will be deferred until the EBox does a FORK or GET OPERAND.

NOTE

If an IBox Flush and Load CPC occurs before the EBox checks port status, the TB Error will not cause a trap to the EHM. The TB Error Status will, however, remain latched in the MBox until the EBox issues an (MCF) MBox Clear Error Regs. That won't happen until the EHM is called handle the next error. Thus, the (left over) TB Parity Error indication will show up in the next Stack Frame built by the EHM. Therefore, do not be confused by such a mixture of status.

ERROR SIGNATURE (TB Valid Parity Error):

MSTAT1 <11> = TB Valid Error

PROBABLE CAUSE:

Module	Probability	RAMS
L0205/MAP	High	E55, E60

ERROR SIGNATURE (TB PTE B Parity Error):

MSTAT1 <10> = TB PTE B Parity Error

PROBABLE CAUSE:

Module	Probability	RAMS
L0205/MAP	High	E136, E127, E123, E118 E113, E82, E78 If EBox port: E47, E34, If IBox port: E40, E28

MANUAL MACHINE CHECK STACK FRAME ANALYSIS

ERROR SIGNATURE (TB PTE A Parity Error):

MSTAT1 <09> = TB PTE A Parity Error

PROBABLE CAUSE:

Module	Probability	RAMS
L0205/MAP	High	E146, E140, E136 E774, E69, E64

ERROR SIGNATURE (TB TAG Parity Error):

MSTAT1 <08> = TB TAG Parity Error

PROBABLE CAUSE:

Module	Probability	RAMS
L0205/MAP	High	E145, E139, E135, E131 E126, E122, E117

EHM ACTION: Standard (See Introduction). In addition the EHM will clear this error condition by invalidating the entire Translation Buffer.

EHM VECTOR: 8 (EBox Request - VMQSAV contains the VA)
10 (OP Port Request - VASAV contains the VA)
18 (IBuffer Request - VIBASAV contains the VA)

CSL ACTION: None

VMS ACTION: Standard (See introduction)

M-02 MBox Cache Tag Parity Errors

OVERVIEW: When a read or write request is accepted by the MBox, bits <12:04> of the physical address referenced are used to index the Tag Storage for two Data Caches. A tag consists of four valid bits (one for each longword stored in the cache block), bits <28:13> of the physical address (where the longwords are stored), and a parity bit. If bits <28:13> of the address referenced match the address portion of the Tag and at least one valid bit is set, then there is at least a "block hit" and there may be a Cache Hit.

A Tag Parity Error in one cache is ignored if there is a "hit" in the other Cache; otherwise the attempted operation is steered toward the bad Cache. If the W Bit equals a zero for that Cache, the bad Tag will be overwritten. If the W BIT equals a one, Cache Writes will be inhibited and CPU requests will be aborted. In all cases DMA requests are forced around Cache to the Array.

A Cache Tag Parity Error on a CPU request with the tag W BIT = 0 or a Cache Tag Parity Error during a DMA request, regardless of the W BIT, results in an MBox IPL 1D interrupt. A Tag Error on a CPU request with the W BIT =1 results in an MBox FATAL ERROR microtrap in the EBox.

NOTE

Both Caches are "looked up" in parallel, checking for a "Hit". If both "Hit", then a Tag Error with the W BIT =1 is forced by the MBox. This results in an MBox Fatal Error.

ERROR SIGNATURE

MSTAT2	<06>	=	Cache Tag Parity Error
MSTAT1	<02>	=	Selected Cache
MEAR	<28:04>	=	Error Address
MSTAT2	<04>	=	Cache Written Bit (Note 1)
EBCS	<14>	=	MBox Interrupt
EBCS	<15>	=	MBox Fatal Error (Note 1)
MSTAT1	<18>	=	ABus C/A Cycle (Note 2)
MSTAT1	<17:16>	=	Selected Adapter (Note 2)
MSTAT1	<31:30>	=	CPU Port (Note 2)
MSTAT1	<29:26>	=	Cycle Type (Note 2)

Note

1. If MBox FE then error on CPU request with MSTAT2 <04> true.

2. If ABus C/A Cycle then error occurred on a DMA reference and MSTAT1 <17:16> are relevant and MSTAT1 <29:26> should = "8", ABus Cycle, or "4", ABus Array Write Cycle. If not ABus C/A Cycle, then the error occurred a CPU reference and MSTAT1 <31:30> are relevant and MSTAT1 <29:26> should = "E", CP Read Cycle, "D", CP Write Cycle, or "3", Write Back Cycle.

MANUAL MACHINE CHECK STACK FRAME ANALYSIS

PROBABLE CAUSE (if Caches 0 Select):

Module	Probability	Components (RAMS)
L0205/MAP	High	E148,E132,E128,E119,E142 E133,E129,E115,E86,E59,E63

PROBABLE CAUSE (if Caches 1 Select):

Module	Probability	Components (RAMS)
L0205/MAP	high	E141,E137,E124,E114,E149 E138,E125,E120,E89,E68,E73

EHM ACTION: If both Caches are on, the EHM sweeps the cache block selected by MEAR <12:4> in the good cache and then executes a Cache Clear which clears both Cache Tags at that index. If only one Cache is on the EHM simply Cache Clear's the bad Tag. Having helped VMS to avoid stumbling across a (potentially) fatal Tag error so that the error can be logged. The EHM rolls back the Instructions, builds a Stack Frame, and vectors to the VMS Machine Check Handler via SCBB+4.

EHM VECTOR: 6

CSL ACTION: None

VMS ACTION: VMS builds a full Machine Check Report, puts it a buffer, queues the buffer to be appended to the System Event File (ERRLOG.SYS), and if the W BIT is equal to 1, VMS takes a System Fatal Bugcheck: some unknown process has lost write data. Otherwise a counter is incremented and, if three errors have occurred within 100 milliseconds, the affected cache is shut off and a message is sent to the Console reporting the event.

M-03 MBox NXM Errors

OVERVIEW: When the MBox honors a CPU or DMA request, the physical address referenced is latched in the PA Latch on the MAP Module. Bits <29:20> of the PA Latch are used to address the PAMM (Physical Address Memory Map RAMS) which produces a five bit code for every megabyte of physical address space. The PAMM is configured by the console during Boot when memory and I/O space are sized. Unused megabytes of physical address space are assigned the NXM code (1F).

PAMM Code	Description
00-07	Select Arrays 0-7
08-17	Reserved
18-1B	Select ABus Adapters
1D-1E	Reserved
1F	Non-existent Memory (NXM)

CPU requests which result in the NXM code are aborted and an MBox Fatal Error microtrap is triggered in the EBox through vector 8.

DMA requests which result in a PAMM code of 1X (the most significant PAMM bit true) are also aborted and an MBox IPL 1D interrupt is requested. Note that SBIA's pass on SBI DMA requests to the MBox only when those requests pass an address check. That is, the address referenced must be within the range of Venus internal memory as indicated by bits <29:20> of the SBIA's Configuration Register which is loaded by the Console during Boot.

ERROR SIGNATURE:

```
MSTAT2 <03>      =  NXM
MEAR             =  Error Address
MSTAT2 <20:16>   =  the PAMM Code
EBCS <15>        =  MBox Fatal Error (If CPU Error)
MSTAT1 <17:16>   =  ABUS ADAPTER <1:0> (If DMA Error)
MSTAT1 <29:26>   =  Cycle Type (Note 1)
MERG <08>        =  Memory Management Enable (Note 2)
MSTAT1 <31:30>   =  CPU Port (If CPU Error)
SBIA ERRSUM <12,08,04,00>  =  MBox Detected (If DMA Error)
```

Note

1. For CPU errors this should be "E", CP READ CYCLE or "D", CP WRITE CYCLE. For DMA error this should be "8", ABUS CYCLE, or "4", ABUS ARRAY WRITE CYCLE

2. Helps to point toward CPU if 0

PROBABLE CAUSE (If CPU Error):

Module	Probability
L0205/MAP	High
L0220/L0230/MCC	Low
CPU	Low (NOTE 1)

Note

1. If MERG <08> equals a zero then the CPU may have produced a bad (nonexistent) address; check MEAR. MSTAT1 <31:30> will identify the Port and Box. Even if MERG <08> equals a one the EBox makes some non virtual (physical) references which bypass the TB (see if MEAR equals VMQ.SAV and MSTAT1 <31:30> equals a 2, indicating EBox Port).

PROBABLE CAUSE (If DMA Error):

Module	Probability
L0205/MAP	High
L0202/SBS	Med
L0203/SBA	Low
L0220/L0230/MCC	Low

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 6 (MBox Interrupt)
8 (MBox FE)

VMS ACTION: VMS builds a full Machine Check Report, puts it a buffer, queues the buffer to be appended to the System Event File (ERRLOG.SYS), and attempts one retry for CPU errors. If the retry fails or if two errors occur within 20 milliseconds, a System Fatal Bugcheck is taken. For DMA errors, the error is logged and a System Fatal Bugcheck is taken.

M-04 MBox CP IO Buffer Error

OVERVIEW: This error occurs when an ABus adapter detects any one of the following errors during a CP TO I/O reference:

- o Any ABus parity error detected on a CPU's ABus transaction
- o An illegal adapter address
- o An SBI Timeout (several variations)
- o An SBI Error Confirmation
- o A State Machine Control Store parity error.

The adapter returns ABus CPU BUF ERR H to the MBox which initiates an MBox Fatal Error micro-trap in the EBox.

ERROR SIGNATURE:

MSTAT2 <02>	=	CP IO Buffer Error
EBCS <15>	=	MBox Fatal Error
EBCS <14>	=	MBox Interrupt
SBIA ERRSUM <23>	=	CPU Buffer Error Lock (Note 1)
SBIA ERRSUM <31:26>	=	CPU Command/Length
SBIA ERRSUM <22>	=	CPU Address/Data Parity Error (Note 2)
SBIA ERRSUM <21>	=	CPU Control Parity Error (Note 2)
SBIA ERRSUM <20>	=	Address Error (Note 2)
SBIA ERRSUM <19>	=	Error on CPU Command/Address (Note 3)
SBIA ERRSUM <18>	=	State Machine Parity Error (Note 2)
SBIA SBIERR <12>	=	SBI Timeout (Note 2)
SBIA SBIERR <11:10>	=	CP Timeout Status <1:0> (timeout type)
SBIA SBIERR <08>	=	CP SBI Error Confirmation (Note 2)
SBIA TOADR	=	CPU (Longword) ABus Address

Note

1. Locks CPU error status for all CPU errors except ERRSUM <18>.
2. Different errors.
3. Sets for errors detected on the CPU's Command/Length/Address, not for errors on the Data/Mask/Status.
4. MEAR, and MSTAT2 <20:16> (PAMM CODE) are not valid for these errors. Also, if the Machine Check has not been processed by the VMS Machine Check Handler then MSTAT1 <17:16> (Selected Adapter) is not valid. For further information see the individual error write-ups.

EHM ACTION: The EHM rolls back the Instructions, builds a Machine Check Stack Frame, and vectors to the VMS Machine Check Handler via SCBB+4. See EHM flows for more detail.

EHM VECTOR: 6

VMS ACTION: VMS builds a full Machine Check Report, puts it in a buffer, queues the buffer to be appended to the System Event File (ERRLOG.SYS), and increments a counter: if two occur within 20 milliseconds or the retry fails (same PC), it takes a System Fatal Bugcheck.

EHM VECTOR: 8 (MBox FE)

VMS attempts retry for the first error only if certain conditions are met; otherwise it returns an error to the current process, which, if the processor mode is Kernel or Exec, will result in a System Fatal Bugcheck. For further information see the individual error write-ups.

M-05 MBox CPR (CCC) Parity Error

OVERVIEW: When the MBox accepts a CPU request (MCF), a modified form of the MCF is used to address the Cycle Parameter RAMS. The resulting eighteen functional CPR bits (there are also two parity bits) then help direct the resulting MBox operation. The CPR's are not used for DMA.

The parity bits are pre-computed and loaded into RAMs along with the CPR data bits by the Console (via the SDB and Microdata MCA's). MCCC U CPR PAR A H provides odd parity for ten functional bits and MCCC U CPR PAR B H provides odd parity for the other eight.

When a CPR Parity Error occurs, the operation that takes place in the MBox is unpredictable. An error therefore results in an MBox Fatal Error microtrap in the EBox through vector 8. Under extremely rare conditions, an error may be reported by MBox IPL 1D interrupt or Error Address Full Trap. If so then the EHM will set EHMSTS <17>, Process Abort, to serve as an Abort Flag for VMS.

ERROR SIGNATURE:

MSTAT1	<23> =	CPR PE B
MSTAT1	<22> =	CPR PE A
EBCS	<15> =	MBox Fatal Error (usually)
EBCS	<14> =	MBox Interrupt (always)

PROBABLE CAUSE (CPR B Parity Error):

Module	Probability	RAMS
L0220/L0230/MCC	high	E118 (4 inputs to parity tree)
		E139 (2 inputs)
		E129 (1 input)
		E126 (1 input)
		E116 (1 input)

PROBABLE CAUSE (CPR A Parity Error):

Module	Probability	RAMS
L0220/L0230/MCC	High	E129 (3 inputs)
		E126 (3 inputs)
		E116 (3 inputs)
		E139 (2 inputs)

EHM ACTION: EHM rolls back the Instructions (except when the error is reported via Error Address Full Trap) builds a Machine Check Stack Frame, and vectors to the VMS Machine Check Handler via SCBB+4.

EHM VECTOR: 4 (If detected while handling an ERF Trap Request)
6 (If detected while handling an MBox Interrupt)
8 (If detected while handling an MBox FE Trap Request

VMS ACTION: VMS builds a full Machine Check Report, puts it in a buffer, queues the buffer to be appended to the System Event File (ERRLOG.SYS), and takes a Fatal Bugcheck.

M-06 MBox Detected CPU Write Parity Errors

OVERVIEW: Result data destined for storage in the GPRs, Memory, or I/O Space are driven onto the WBus. After latching the result data from the WBus, the EBox generates byte parity, which is also driven onto the WBus. The WBus parity bits are sent directly to the MBox MCD Module, while the WBus data must pass through IDP, the DBUS, IBD, and over the MDBus before reaching MCD where a parity check occurs. All four bytes are parity checked regardless of the context sent via ICB (for OP Port Writes) and EBC (for EBox Writes). The context indicates to the MBox which bytes are valid write data.

If the IBox rotates the result data driven onto the MDBus, then the MCD Module (which receives a copy of the rotation count in ICB WRT ROT <1:0> H), rotates the parity bits (EBE WBUS OPAR <B3:B0> L) to properly realign them with their bytes.

If the EBox is doing an MBox Register write then bad parity results in an MBox Fatal Error microtrap in the EBox. Otherwise, bad parity results in an IPL 1D interrupt.

If a Cache Write is performed, the parity that is sent by the CPU is written , as is (good or bad), in the cache parity RAMs. Bytes not written during a byte write retain their original parity. The ECC character that is stored with data that fails the byte parity check (including when cache is off) will be generated to indicate Bad Data. If the error occurs on a byte not written, this ECC will never be accessed because the MBox must first detect a Cache parity error before it will attempt Cache correction.

NOTE

The EBox checks the parity of all data that it transmits on the WBus. A mismatch (WBus PE) will result in a process abort with the process abort code equal to 2 (EBCS <19:16> = 2).

ERROR SIGNATURE

MSTAT1	<07:04>	=	Byte(s) in Error
EBCS	<14>	=	MBox Interrupt
EBCS	<15>	=	MBox Fatal Error (Note 1)
MSTAT1	<29:26>	=	Cycle Type
MEAR	<29:02>	=	Error Address (physical) (Note 2)

NOTE

1. If MSTAT1 <29:26> equals "2" (MBox Register Write Cycle) it is a fatal error. If MSTAT1 <29:26> equal a "D" (CPU Write Cycle) then only an interrupt is requested.

2. If MSTAT1 <29:26> equal a "D", then this error may generate a Bad Data Flag error at this address. If MSTAT1 <29:26> equals a "2" then MEAR contains the address of the Register written.

PROBABLE CAUSE:

Module	Probability
L0208/IBD	High
L0206/IDP	High
L0204/MCD	Medium
DBus	Low
MDBus	Low
Parity Path to MBox	Low

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 8 (MBox FE)
6 (MBox Interrupt)

VMS ACTION: Standard (See Introduction). In addition, if the error involved an MBox Register Write VMS will execute a System Fatal Bugcheck.

M-07 MBox Detected ABus Parity Errors

OVERVIEW: Odd parity protects both the ABus Address/Data lines and the ABus Control lines. During Command/Address cycles, the Address/Data lines transmit the Address while the Control lines transmit the Command/Length. During Data cycles, the Address/Data lines transmit Data while the Control lines transmit the Mask/Status.

The MBox parity checks all ABus Control bits in the ABS MCA on the MCC module (MCC4). It parity checks the Address on the MAP module in the ADB MCA and the ADA MCA's (MAP1-2). It parity checks Data on the MCD module: byte parity generated on the Data latched in the Data Path MCA's MCD1-3 is collapsed into longword parity and compared against the longword parity latched from the ABus (MCD3).

For parity errors on a DMA Command/Length or Address, the MBox aborts the operation, returns MCC ABus DMA ERROR H to the requesting adapter, and requests an IPL 1D interrupt. For further information, see SBIA Detected DMA Errors.

For parity errors on DMA Write Data or Mask/Status, the write completes as if there were no error but an MBox Fatal Error micro-trap is triggered in the EBox. On rare occasions if a Control Parity Error is detected when the first longword is being processed for a DMA Write (only) an MBox interrupt will be requested. For parity errors on CPU read Data or Mask/Status, the Data is passed on unaltered to the CPU but an MBox Fatal Error micro-trap is triggered in the EBox.

ERROR SIGNATURE

MSTAT1	<21>	=	ABus Data PE
MSTAT1	<20>	=	ABus Control PE
MSTAT1	<19>	=	ABus Address PE
MSTAT1	<18>	=	ABus C/A Cycle (Note 1)
EBCS	<15>	=	MBox Fatal Error (Note 2)
EBCS	<14>	=	MBox Interrupt
MSTAT1	<17:16>	=	Selected Adapter
MSTAT1	<31:30>	=	CPU Port
MSTAT1	<29:26>	=	Cycle Type (Note 3)
MEDR	<31:00>	=	ABus Data

Note

1. If set, the error happened on the Command/Length or Address (i.e. if MSTAT1 <20> is set, then the Command/Length not Mask/Status was bad).

2. If set, then MSTAT1 <18> is reset.

3. If Cycle Type equals "E" (CP READ CYCLE) then MSTAT1 <31:30> indicate the port. If Cycle Type equals "8" (ABus CYCLE) or "4" (ABus Array Write) then the error was during DMA and the CPU PORT is not relevant.

4. If MSTAT1 <29:26> equals "E", then both MEAR and MSTAT2 <20:16> (PAMM Code) are not valid.

PROBABLE CAUSE (If Data Parity Error):

Module	Probability
L0202/SBS	High
L0203/SBA	High
L0204/MCD	High
ABus/Terminator	Low

PROBABLE CAUSE (If Control Parity Error):

Module	Probability
L0202/SBS	High
L0203/SBA	High
L0220/L0230/MCC	High
ABus/Terminator	Low

PROBABLE CAUSE (If Address Parity Error):

Module	Probability
L0202/SBS	High
L0203/SBA	High
L0205/MAP	High
ABus/Terminator	Low

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 8 (MBox FE)
6 (MBox Interrupt)

VMS ACTION: Standard (See Introduction). VMS will execute a System Fatal Bugcheck.

M-08 MBox Array ECC Errors

OVERVIEW: The ECC MCA hanging off the Array Bus on the MCD Module generates a seven bit ECC character that is stored with each longword during either an Array or Cache Write. The ECC MCA also produces a six bit syndrome as well as other error status and signals during array reads. Latched status is later gathered by the EHM into the register known as MDECC. Error signals sent onto the ERR MCA on MCC generate MCCM MBox INTR H.

The ECC character generation includes an address parity bit which is calculated on PA <28:04> of the address where the longword is to be stored. The ECC MCA handles the Address Parity Bit as if it were a thirty third data bit. When the longword is later read, a parity bit generated across PA <28:04> from where the longword was fetched is XOR'ed in for the cancel. A mismatch results in the unique "single bit" syndrome which says Address Parity Error. (This error can occur when data is read from the wrong location and later accessed.) During ECC generation the Bad Data Flag is also handled in a similar manner (i.e., The ECC MCA treats the Bad Data Flag as if it were thirty fourth data bit). The Bad Data Flag is set under the following conditions:

- A CPU WRITE with bad parity occurs (CPU writes always go to Cache unless the Cache is turned off).
- An attempt at Cache Correction fails; the data is either re-cached or, written to the Array if the operation was a writeback.
- A DMA Masked Write occurs to a Cache or Array location that has an uncorrectable ECC error.
- A CP Byte Write to a Cache (or Array location if the Cache is turned off) which has an uncorrectable ECC Error. A CP Byte Write only updates specified bytes in a longword).

Bad Data status is assumed zero when a longword is read and a mismatch results in the unique single-bit syndrome which says Bad Data.

To protect against failures in the control logic which gates data onto the Array Bus, two of the check bits are inverted by the ECC MCA before being driven onto the bus. These bits are reinverted when a longword is checked. This enables the ECC MCA to detect all ones or zero's on the bus and to latch and send ECC Fatal Error status to the ERR MCA.

During Refills, Array longwords are cached in parallel with being sent on to the read requester. Array longwords are cached with their original ECC. If an uncorrectable error is detected during an Array Read the Byte Parity is inverted before it is cached.

For uncorrectable ECC Errors during DMA reads the ERR MCA disables the ABus drivers and thus sends an all zero's response (bad longword parity) to the ABus Adapter. This will result in SBI RDS being sent on to the requesting NEXUS. Also the SBIA will latch an error in the ERRSUM Register <14,10,6,2>. If EBCS <14> is set, I/O status reporting the SBI RDS should be ignored.

ERROR SIGNATURE

EBCS	<14>	=	MBox Interrupt
MSTAT1	<29:26>	=	Cycle Type (Note: 1)
MSTAT1	<17:16>	=	Selected Adapter (Note: 1)
MSTAT1	<31:30>	=	CPU Port (Note: 1)
MEAR	<28:04>	=	Octaword in Error
MSTAT1	<25:24>	=	Longword in Error
MSTAT2	<20:16>	=	PAMM Code (Note: 2)
MSTAT2	<27>	=	Array Type Code Valid
MSTAT2	<27:24>	=	Array Type Code
MDECC	<22>	=	Bad Data Error (Note: 3)
MDECC	<21>	=	Data Single Bit Error
MDECC	<20>	=	Data Double Bit Error
MDECC	<19>	=	Data Address Parity Error
MDECC	<14:09>	=	Syndrome (Note: 4)

NOTE

1. If the cycle type equals "F" then the error happened during a DMA reference and MSTAT1 <17:16> is relevant. If the cycle type equals "9" then the error happened on a CPU reference and MSTAT1 <31:30> is relevant.

2. The array slot associated with the error.

3. Look for a previous error at the same MEAR <28:4> for the cause of this condition.

4. The data bit in error (or bad data or address parity error) and the corresponding syndrome is as follows.

SYNDROME	(MSB)	70	0421000	66666655555444443333322222111111
IN OCTAL	(LSB)	07	0000421	65432132654654326543265432654321
DATA BIT	(MSB)	BA	CCCCCCC	33222222222111111111110000000000
	(LSB)	DP	0654321	10987654321098765432109876543210

PROBABLE CAUSE (Address Parity Error):

Module	Probability
L0200/ARRAY	High (if same MSTAT2 <20:16> repeats)
L0220/L0230/MCC	High (if random MSTAT2 <20:16>)
L0205/MAP	Medium (if random MSTAT2 <20:16>)
L0204/MCD	Low

PROBABLE CAUSE (Double Bit Error):

Module	Probability
L0200/ARRAY	High
L0204/MCD	Medium
Array Bus/Terminator	Low
L0220/L0230/MCC	Low

PROBABLE CAUSE (CRD, Corrected Read Data):

Module	Probability
SMU	High
Array	Med
Array Bus/Terminator	Low
L0220/L0230/MCC	Low
L0204/MCD	Low

NOTE

Array, above may be the L0200, L0225, L0226, or L0235 modules. An SMU would only be applicable for the L0225 or L0235 modules.

EHM ACTION: Standard (See Introduction). SBE errors vector to SCBB+54.

EHM will set process abort, EHMSTS <17>, if the error occurred on the target word (Array Refill) and the error was not detected, handled, and cleared by the Box requesting the data.

VMS ACTION: For DB and BD Errors, VMS builds a full machine check report, puts it a buffer, queues the buffer to be appended to the system event file (ERRLOG.SYS), and, for an error on an unmodified page, attempts to bring in a fresh copy of the page from disk (remapping it and sending the bad page to the bad page list). If successful, VMS REI's. Otherwise, depending on the processor mode at the time of error, and who owns the bad page, it either aborts the current process or executes a system fatal bugcheck.

For parity errors VMS executes a system fatal bugcheck.

For SBE errors, VMS reads MEAR, MDECC, MSTAT1, and MSTAT2 from the EScratch (Machine Check Stack Frame) and stores them in a buffer. Sixteen errors are allowed to accumulate before a SBE error log entry is made. If three SBEs are detected within 10 millisecs, the SBEs are logged and SBE logging is disabled for 5 minutes.

NOTE

Generally, if an error results in a system crash, VMS MCHK puts the error into a VMS error buffer and takes a crash dump. The VMS error buffers are appended to the dump and processed at the next reboot.

M-09 MBox Cache Data Parity Errors

OVERVIEW: Each Cache data longword is stored with four byte parity bits and a seven bit ECC Character. Cache byte parity is checked in the Data Path MCA's on MCD1-3. The the results sent to the UFO MCA (MCDU) where it is latched as status. The result is also sent to the ERR MCA MCCM) where it generates an interrupt.

For errors detected during a CP Cache Read, the data and byte parity are sent to the CPU as is. For errors detected during an ABus Cache Read, an all zero's longword and parity bit are sent to the ABUS. In both cases the next operation that the MBox performs will be a Cache Data Correction Cycle. This operation will correct the entire Cache Block driving all four longwords along with their ECCs from the Check RAMs (each in turn) onto the Array Bus so that the ECC MCA can generate a correcting syndrome. Thus, when the retry occurs the data will have been corrected or else re-cached with inverted byte parity and ECC indicating Bad Data. For errors detected during a Byte Merge Write, correction is performed before the byte(s) are merged and written.

The MBox stores Cache correction status in MDECC. MDECC is also used to store Array correction status.

NOTE

> A zero syndrome can be due to a fault in a byte parity bit-- or due to a transient since correction involves re-reading the longword(s) for a second time. For further information, see Array Errors.

Bad parity sent onto the CPU should result in an EBox microtrap due to an IBox Error (EBCS <13>) or an EDP PE (EBCS <9>) If such a trap does not occur the EHM will set EHM.STS <17> (Process Abort.)

Bad parity sent to the ABus will result in SBI RDS being sent on to the requesting NEXUS. Also the SBIA will latch an error in the ERRSUM Register <14,10,6,2>. If EBCS <14> is set, I/O status reporting the SBI RDS should be ignored.

ERROR SIGNATURE

MSTAT1	<03>	=	Cache Read Data Parity Error (Note: 1)
MSTAT1	<00>	=	CP Byte Write Cache Data Parity Error (Note: 1)
EBCS	<14>	=	MBox Interrupt
MSTAT1	<29:26>	=	Cycle Type
MSTAT1	<31:30>	=	CPU Port (Note: 2)
MSTAT1	<17:16>	=	Adapter Select (Note: 2)
MSTAT1	<02>	=	Selected Cache
MEAR	<28:04>	=	Octaword in Error
MSTAT1	<25:24>	=	Longword in Error
MDECC	<22>	=	Bad Data Error (Note: 3)
MDECC	<21>	=	Data Single Bit Error
MDECC	<20>	=	Data Double Bit Error
MDECC	<19>	=	Data Address Parity Error (Note: 4)
MDECC	<14:09>	=	Syndrome

Note

1. Two different errors; the second only sets for errors during CP Byte Write Operations, not DMA Masked Writes.

2. If the Cycle Type equals "8" (ABus Cycle) then the error occurred on a DMA reference and MSTAT1 <17:16> are relevant. If the Cycle Type equals "E" (CPU Read Cycle), or "D" (CPU Write Cycle), or "3" (Write Back Cycle), then the error occurred on a CPU reference and MSTAT1 <31:30> are relevant.

3. Look for a previous error at the same MEAR <28:4> which may have caused the Bad Data Flag to be set.

4. Should only occur in conjunction with another error; either an error in a Data/Parity RAM or a cached Array Address Parity Error.

PROBABLE CAUSE (If: MSTAT1 <02> (Cache Sel) = 0 and MEAR <12> = 0)

Module	Probability	Components
L0204/MCD	High	RAMs (See Table 1)

PROBABLE CAUSE (If MSTAT1 <02> (Cache Sel) = 0 and MEAR <12> = 1):

Module	Probability	Components
L0204/MCD	High	RAMs (See Table 2)

PROBABLE CAUSE (If MSTAT1 <02> (Cache Sel) = 1 and MEAR <12> = 0):

Module	Probability	Components
L0204/MCD	High	RAMs (See Table 3)

PROBABLE CAUSE (If MSTAT1 <02> (Cache Sel) = 1 and MEAR <12> = 1):

Module	Probability	Components
L0204/MCD	High	RAMs (See Table 4)

Table 1 RAM Call out (MSTAT1 <02> = 0 and MEAR <12> = 0)

Syndrome	Bit	Signal Name	RAM
0	BP 0	MCDH GRP 0 BP 0	E52
0	BP 1	MCDH GRP 0 BP 1	E52
0	BP 2	MCDH GRP 0 BP 2	E52
0	BP 3	MCDH GRP 0 BP 3	E52
11	00	MCDF GRP 0 DATA 00	E11
12	01	MCDF GRP 0 DATA 01	E11
13	02	MCDF GRP 0 DATA 02	E11
14	03	MCDF GRP 0 DATA 03	E24
15	04	MCDE GRP 0 DATA 04	E24
16	05	MCDE GRP 0 DATA 05	E24
22	06	MCDE GRP 0 DATA 06	E32
23	07	MCDE GRP 0 DATA 07	E42
24	08	MCDD GRP 0 DATA 08	E11
25	09	MCDD GRP 0 DATA 09	E24
26	10	MCDD GRP 0 DATA 10	E32
32	11	MCDD GRP 0 DATA 11	E32
33	12	MCDC GRP 0 DATA 12	E32
34	13	MCDC GRP 0 DATA 13	E42
35	14	MCDC GRP 0 DATA 14	E42
36	15	MCDC GRP 0 DATA 15	E42
42	16	MCDB GRP 0 DATA 16	E113
43	17	MCDB GRP 0 DATA 17	E113
44	18	MCDB GRP 0 DATA 18	E113
45	19	MCDB GRP 0 DATA 19	E126
46	20	MCDA GRP 0 DATA 20	E126
54	21	MCDA GRP 0 DATA 21	E126
55	22	MCDA GRP 0 DATA 22	E138
56	23	MCDA GRP 0 DATA 23	E151
52	24	MCD9 GRP 0 DATA 24	E113
53	25	MCD9 GRP 0 DATA 25	E126
61	26	MCD9 GRP 0 DATA 26	E138
62	27	MCD9 GRP 0 DATA 27	E138
63	28	MCD8 GRP 0 DATA 28	E138
64	29	MCD8 GRP 0 DATA 29	E151
65	30	MCD8 GRP 0 DATA 30	E151
66	31	MCD8 GRP 0 DATA 31	E151

Table 2 RAM Call out (MSTAT1 <02> = 0 and MEAR <12> = 1)

Syndrome	Bit	Signal Name	RAM
0	BP 0	MCDH GRP 0 BP 0	E45
0	BP 1	MCDH GRP 0 BP 1	E45
0	BP 2	MCDH GRP 0 BP 2	E45
0	BP 3	MCDH GRP 0 BP 3	E45
11	00	MCDF GRP 0 DATA 00	E5
12	01	MCDF GRP 0 DATA 01	E5
13	02	MCDF GRP 0 DATA 02	E5
14	03	MCDF GRP 0 DATA 03	E18
15	04	MCDE GRP 0 DATA 04	E18
16	05	MCDE GRP 0 DATA 05	E18
22	06	MCDE GRP 0 DATA 06	E28
23	07	MCDE GRP 0 DATA 07	E36
24	08	MCDD GRP 0 DATA 08	E5
25	09	MCDD GRP 0 DATA 09	E18
26	10	MCDD GRP 0 DATA 10	E28
32	11	MCDD GRP 0 DATA 11	E28
33	12	MCDC GRP 0 DATA 12	E28
34	13	MCDC GRP 0 DATA 13	E36
35	14	MCDC GRP 0 DATA 14	E36
36	15	MCDC GRP 0 DATA 15	E36
42	16	MCDB GRP 0 DATA 16	E107
43	17	MCDB GRP 0 DATA 17	E107
44	18	MCDB GRP 0 DATA 18	E107
45	19	MCDB GRP 0 DATA 19	E120
46	20	MCDA GRP 0 DATA 20	E120
54	21	MCDA GRP 0 DATA 21	E120
55	22	MCDA GRP 0 DATA 22	E132
56	23	MCDA GRP 0 DATA 23	E144
52	24	MCD9 GRP 0 DATA 24	E107
56	25	MCD9 GRP 0 DATA 25	E120
61	26	MCD9 GRP 0 DATA 26	E132
62	27	MCD9 GRP 0 DATA 27	E132
63	28	MCD8 GRP 0 DATA 28	E132
64	29	MCD8 GRP 0 DATA 29	E144
65	30	MCD8 GRP 0 DATA 30	E144
66	31	MCD8 GRP 0 DATA 31	E144

Table 3 RAM Call out (MSTAT1 <02> = 1 and MEAR <12> = 0)

Syndrome	Bit	Signal Name	RAM
0	BP 0	MCDH GRP 1 BP 0	E53
0	BP 1	MCDH GRP 1 BP 1	E53
0	BP 2	MCDH GRP 1 BP 2	E53
0	BP 3	MCDH GRP 1 BP 3	E53
11	00	MCDF GRP 1 DATA 00	E12
12	01	MCDF GRP 1 DATA 01	E12
13	02	MCDF GRP 1 DATA 02	E12
14	03	MCDF GRP 1 DATA 03	E25
15	04	MCDE GRP 1 DATA 04	E25
16	05	MCDE GRP 1 DATA 05	E25
22	06	MCDE GRP 1 DATA 06	E33
23	07	MCDE GRP 1 DATA 07	E43
24	08	MCDD GRP 1 DATA 08	E12
25	09	MCDD GRP 1 DATA 09	E25
26	10	MCDD GRP 1 DATA 10	E33
32	11	MCDD GRP 1 DATA 11	E33
33	12	MCDC GRP 1 DATA 12	E33
34	13	MCDC GRP 1 DATA 13	E43
35	14	MCDC GRP 1 DATA 14	E43
36	15	MCDC GRP 1 DATA 15	E43
42	16	MCDB GRP 1 DATA 16	E114
43	17	MCDB GRP 1 DATA 17	E114
44	18	MCDB GRP 1 DATA 18	E114
45	19	MCDB GRP 1 DATA 19	E127
46	20	MCDA GRP 1 DATA 20	E127
54	21	MCDA GRP 1 DATA 21	E127
55	22	MCDA GRP 1 DATA 22	E139
56	23	MCDA GRP 1 DATA 23	E152
52	24	MCD9 GRP 1 DATA 24	E114
53	25	MCD9 GRP 1 DATA 25	E127
61	26	MCD9 GRP 1 DATA 26	E139
62	27	MCD9 GRP 1 DATA 27	E139
63	28	MCD8 GRP 1 DATA 28	E139
64	29	MCD8 GRP 1 DATA 29	E152
65	30	MCD8 GRP 1 DATA 30	E152
66	31	MCD8 GRP 1 DATA 31	E152

Table 4 RAM Call out (MSTAT1 <02> = 1 and MEAR <12> = 1)

Syndrome	Bit	Signal Name	RAM
0	BP 0	MCDH GRP 1 BP 0	E46
0	BP 1	MCDH GRP 1 BP 1	E46
0	BP 2	MCDH GRP 1 BP 2	E46
0	BP 3	MCDH GRP 1 BP 3	E46
11	00	MCDF GRP 1 DATA 00	E6
12	01	MCDF GRP 1 DATA 01	E6
13	02	MCDF GRP 1 DATA 02	E6
14	03	MCDF GRP 1 DATA 03	E19
15	04	MCDE GRP 1 DATA 04	E19
16	05	MCDE GRP 1 DATA 05	E19
22	06	MCDE GRP 1 DATA 06	E29
23	07	MCDE GRP 1 DATA 07	E37
24	08	MCDD GRP 1 DATA 08	E6
25	09	MCDD GRP 1 DATA 09	E19
26	10	MCDD GRP 1 DATA 10	E29
32	11	MCDD GRP 1 DATA 11	E29
33	12	MCDC GRP 1 DATA 12	E29
34	13	MCDC GRP 1 DATA 13	E37
35	14	MCDC GRP 1 DATA 14	E37
36	15	MCDC GRP 1 DATA 15	E37
42	16	MCDB GRP 1 DATA 16	E108
43	17	MCDB GRP 1 DATA 17	E108
44	18	MCDB GRP 1 DATA 18	E108
45	19	MCDB GRP 1 DATA 19	E121
46	20	MCDA GRP 1 DATA 20	E121
54	21	MCDA GRP 1 DATA 21	E121
55	22	MCDA GRP 1 DATA 22	E133
56	23	MCDA GRP 1 DATA 23	E145
52	24	MCD9 GRP 1 DATA 24	E108
53	25	MCD9 GRP 1 DATA 25	E121
61	26	MCD9 GRP 1 DATA 26	E133
62	27	MCD9 GRP 1 DATA 27	E133
63	28	MCD8 GRP 1 DATA 28	E133
64	29	MCD8 GRP 1 DATA 29	E145
65	30	MCD8 GRP 1 DATA 30	E145
66	31	MCD8 GRP 1 DATA 31	E145

The EHM will set Process Abort EHMSTS <17> if a non Single Bit Error occurred and the error was not detected handled, and cleared by the Box requesting the data.

VMS ACTION: Standard (See Introduction). For Address Parity Errors VMS executes a System Fatal Bugcheck. For Bad Data Errors see Array Errors.

M-10 MBox Cache W Bit Parity Error

OVERVIEW: Because the processor uses two Writeback Data Caches (i.e. a Cache can be updated without simultaneously updating an Array) it sets a W (Written) Bit in the Cache Tag of the selected Cache to flag when the corresponding Array Octaword is stale. This W Bit is stored with its own odd parity bit. Except that it causes MBox Interrupt, the MBox treats a W Bit PE exactly as if it were a set W Bit. That is, a Writeback will be performed if the selected Cache Block (Octaword) is about to be over written with data belong at a different physical address. Thus insuring that the most current data are always preserved. (If the W Bit had never actually been set, then the Array Octaword will be overwritten with data identical to that already stored there: the Writeback will be a NOP.)

ERROR SIGNATURE

MSTAT2 <05>	=	WBit PE
MSTAT2 <01>	=	Cache Select
MEAR <28:02>	=	Error Address

PROBABLE CAUSE (Cache 0 Select):

Module	Probability	RAMS
L0205/MAP	High	E77

PROBABLE CAUSE (Cache 1 Select):

Module	Probability	RAMS
L0205/MAP	High	E81

EHM ACTION: The EHM sweeps the faulty Cache Block (this is done to clear the error because the MBox may not have performed a Writeback followed by the Cache Invalidate), rolls back the instructions, builds a stack frame, and vectors to the Machine Check Handler via SCBB+4.

VMS ACTION: Standard (See Introduction)

M-11 MBox Detected ABus Bad Data Code

OVERVIEW: A two bit ABus field, ABus LEN STAT <1:0> H, is used to indicate Length during Command/Address cycles and Status during Mask/Data cycles.

The encodings used for STATUS are:

00 Good Data
11 Bad Data

The MBox always transmits Good Status for speed reasons, although if an SBIA detects the Bad Data code accompanying DMA Read Data from the MBox it will convert the Bad Data code into an SBI RDS to accompany that data back to the SBI Nexus.

For this error we must consider the reverse direction. That is, when an Adapter sends CP Read Data (not DMA Write Data) and Bad Data Status to the MBox. At present this occurs only when an SBIA converts an SBI RDS accompanying CP Read Data into the ABus Bad Data code to continue to accompany that data to the MBox.

It ought to be kept in mind that a DW780 can produce an RDS for a read access to any Unibus device capable of asserting the Unibus PB Line (this includes devices with parity protected registers).

A Bad Data code status bit is latched in the ABS MCA (which will actually latch the bit for any non zero Status value). Logic external to the MCA on the MCC module also detects the error and reports it to the ERR MCA which will request an MBox IPL 1D interrupt. It will also disable the MDBus drivers and thus provide all zero's Read Data (bad byte parity) to the requesting CPU port. As a result, this error may be reported by an EBox microtrap due to an IBox Error (EBCS <13>) or an EBox EDP PE (EBCS <09>); otherwise the error will be reported by MBox IPL 1D interrupt.

ERROR SIGNATURE

MSTAT2 <14>	=	ABus Bad Data Code
EBCS <14>	=	MBox Interrupt
EBCS <01>	=	IO Read (Note 1)
EHMSTS <17>	=	Process Abort (Note 2)
MSTAT1 <29:26>	=	"E", CP Read Cycle
MSTAT1 <31:30>	=	CPU Port
MSTAT2 <17:16>	=	Selected Adapter
MEDR	=	ABus Longword (the Read Data)
MSTAT1 <31:30>	=	CPU Port

NOTE

1. Can only set for RDS on reads to I/O space with PA <29> equal to a 1. PA <29> equals a 0 for reads to SBI memory. If EHMSTS <17> equals a 1, EBCS <01> may not have been captured by the error

2. See EHM ACTION

3. MEAR and MSTAT2 <20:16> (PAMM Code) are not valid for this error.

PROBABLE CAUSE:

Module	Probability
SBI NEXUS	High
L0202/SBS	Low
L0203/SBA	Low
L0220/L0230/MCC	Low

EHM ACTION: If the error is reported via MBox interrupt or Error Address Full Trap, EHM sets EHMSTS <17> (Process Abort) as an Abort Flag for VMS informing it that, because the usual EBox microtrap did not occur to prevent consuming the all zero's data, the current instruction probably cannot be retried. Regardless of reporting method, EHM rolls back the instructions, builds a stack frame, and vectors to the Machine Check Handler via SCBB+4.

VMS ACTION: VMS builds a full Machine Check Report, puts it a buffer, queues the buffer to be appended to the System Event File (ERRLOG.SYS), and increments a counter: if three errors occur within 100 milliseconds, VMS executes a System Fatal Bugcheck. Else, if EHMSTS <17> or EBCS <01> is set, VMS aborts the current process; otherwise it REI's. Note that if the processor mode is Kernel or Exec, which is most likely (unless external memory is in use) Process Abort becomes System Fatal Bugcheck.

E-00 EBox WBus Parity Error

OVERVIEW: This error occurs when the EBox Data Path (EDP) drives data onto the WBus, and the WReg longword parity generated on EDP does not match the WBus parity generated on EBE. When EDP is driving the WBus, the byte parity from EBE is sent to EDP where it is checked against the WReg longword parity.

If a WBus Parity Error occurs all copies of the GPR/SPs will have been corrupted with bad data. This will cause the EBox Error Handling Microcode to loop at location UPC 24, and the Console will detect a Keep Alive Fail Condition.

The EBox WBus Check is done on the EDP module, and the error signal WBUS ERR is sent to the EBD module where it's latched in the EBCS Register as WBus Parity Error.

NOTE

Because this type of an error will result in a Keep Alive Fail Condition it will never cause the EHM to generate a Machine Check Stack Frame.

ERROR SIGNATURE:

EBCS <08> = WBus Parity Error

PROBABLE CAUSE:

Module	Probability	Components
L0209/EDP	High	(See Table 1)
L0219/EBE	Medium	(See Table 2)
L0206/IDP	Medium	WBus Driver/Receiver
L0212/FBA	Medium	WBus Driver/Receiver
L0211/EBD	Very Low	EBCS Register

Table 1 EDP Component Callout

Byte	Bits	ALU	PDP	Misc
3	<31:24>	E102, E101	E3	E13
2	<23:16>	E87, E86	E3	E13
1	<15:08>	E70, E69	E3	E81
0	<07:00>	E53, E52	E3	E81

Table 2 EBE Component Callout

Byte	WBus Latches	Parity Generators
3	E169, E155, E43, E29	E157, E59
2	E169, E127, E43, E29, E85, E1	E115, E59
1	E169, E127, E43, E155, E1	E45, E31
0	E155, E127, E29, E1	E17, E31

EHM ACTION: The EBox microtraps through vector 8, and loops at UPC 24. The Console detects a Keep Alive Fail Condition, builds a Snap File, sends message to console terminal indicating WBus Parity Error, and re-boots the CPU. No Machine Check stack frame is generated.

VMS ACTION: After VMS is re-booted the Snap File is transferred to the VMS side of the system, renamed to ERRSNAP.LOG;n, and written in the SYS$SYSROOT:[SYSERR] directory. In addition an Processor Error Halt (Entry Type 16) is appended to the System Event File (ERRLOG.SYS).

E-01 EBox Result Parity Error (EDP Misc Error)

OVERVIEW: This error occurs when ALU result data is passed through the VMQ MUX, but NOT loaded into the VMQ Register, and the byte parity generated at the output of the VMQ MUX does not match the byte parity of the Condition Code ALU.

NOTE

Two EBox Result Errors are very similar; VMQ and MISC error. The difference is that the VMQ is loaded during a VMQ error, while the VMQ is not loaded during a MISC error.

The EBox Result Check is done on the EDP module PDP MCA, and the error signal RESULT PAR ERR is sent to the EBD module where it is latched in the EBCS Register as EBox Data Path PE. The EBox Abort flag is raised and latched into the EBCS Register on the EBD module. In the EDPSR Register, Result Parity Error and EDP Misc error bits are generated and latched on the EDP module PDP MCA.

ERROR SIGNATURE:

EBCS <09> = EBox Data Path Parity Error
EBCS <04> = EBox Abort Flag (Note 1)
EDPSR <05> = Result Parity Error
EDPSR <08> = EDP Misc Error (Note 2)

Note

1. The EBox Abort Flag will set if IRD LST CYC (uBEN field) is set when the error occurs. This flag indicates the error was detected too late to inhibit the PC from being updated and thus, prevents an instruction retry.

2. EDPSR <15:12> VMQ Byte in Error has no meaning for this error.

PROBABLE CAUSE:

Module	Probability	Components
L0209/EDP	High	(See Table 1)
L0211/EBD	Very Low	EBCS Register

Table 1 EDP Component Callout

Byte	Bits	ALU	PDP	Misc
3	<31:24>	E102, E101	E3	E13
2	<23:16>	E87, E86	E3	E13
1	<15:08>	E70, E69	E3	E81
0	<07:00>	E53, E52	E3	E81

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 8

CSL ACTION: None

VMS ACTION: Standard (See Introduction)

E-02 EBox Result Parity Error (VMQ)

OVERVIEW: This error occurs when the VMQ register is loaded and the result data and parity generated at the output of the VMQMUX does not match the data and byte parity generated at the output of the Condition Code ALU.

NOTE

Two EBox Result Errors are very similar; VMQ and MISC error. The difference is that the VMQ is loaded during a VMQ error, while the VMQ is not loaded during a MISC error.

The EBox Result Check is done on the EDP module PDP MCA, and the error signal RESULT PAR ERR is sent to the EBD module where it is latched in the EBCS Register as EBox Data Path PE. The EBox Abort flag is raised and latched into the EBCS Register on the EBD module. In the EDPSR Register, Result Parity Error and EDP Misc error bits are generated and latched on the EDP module PDP MCA.

ERROR SIGNATURE:

EBCS <9>	= EBox Data Path PE
EBCS <4>	= EBox Abort Flag (Note 1)
EDPSR <5>	= Result Check
EDPSR <15:12>	= VMQ Byte in Error

Note

1. The EBox Abort Flag will set if IRD LST CYC (uBEN field) is set when the error occurs. This flag indicates the error was detected too late to inhibit the PC from being updated and thus, prevents an instruction retry.

PROBABLE CAUSE:

Module	Probability	Components
L0209/EDP	High	(See Table 1)
L0211/EBD	Very Low	EBCS Register

Table 1 EDP Component Callout

Byte	Bits	ALU	PDP	Misc
3	<31:24>	E102, E101	E3	E13
2	<23:16>	E87, E86	E3	E13
1	<15:08>	E70, E69	E3	E81
0	<07:00>	E53, E52	E3	E81

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 8

CSL ACTION: None

VMS ACTION: Standard (See Introduction)

E-03 EBox Result Parity Errors (WReg)

OVERVIEW: This error occurs when the input parity to the WReg Mux does not match the output WReg Parity. There are three sources for WReg Mux input data and parity. There is only one error signature for all three error types. The following is a description of each operation:

1. WReg Shift Operation - This error occurs during an EBox Shifter operation. The Shifter receives 64 bits of input data, one longword from the AMux and one longword from the BMux. However, only 32 bits are passed on to the WReg with long word parity. The parity check is done between the parity of the eight bytes of input data, and the longword parity of the WReg, plus the parity of the unused data in the Shifter.

2. Formatter Operation - This error occurs during an EBox Format operation. The input data to the formatter is the BMux. A check is made between the byte parity at the BMux output and the WReg longword parity. The only format operations that are not parity checked are F FORMAT PACK+ and F FORMAT PACK-.

3. WReg Post ALU Shift Operation - This error occurs when the WReg Mux is passing ALU result data to the WReg. The byte parity generated at the output of the Condition Code ALU is compared against the WReg longword parity. If the WReg Mux performs a shift on the data, the bits shifted in or out are considered in the check.

The EBox Result Check is done on the EDP module PDP MCA, and the error signal RESULT PAR ERR is sent to the EBD module where it is latched in the EBCS Register as EBox Data Path PE. The EBox Abort flag is raised and latched into the EBCS Register on the EBD module. In the EDPSR Register, Result Parity Error and EDP Misc error bits are generated and latched on the EDP module PDP MCA.

ERROR SIGNATURE:

EBCS <09> = EBox Data Path Parity Error
EBCS <04> = EBox Abort Flag (Note 1)
EDPSR <05> = Result Parity Error
EDPSR <11> = WReg Parity Error

Note

1. The EBox Abort Flag will set if IRD LST CYC (uBEN field) is set when the error occurs. This flag indicates the error was detected too late to inhibit the PC from being updated and thus, prevents an instruction retry.

PROBABLE CAUSE:

Module	Probability	Components
L0209/EDP	High	(See Table 1)

L0211/EBD	Very Low	EBCS Register

Table 1 EDP Component Callout

Byte	Bits	ALU	SHF	PDP	Misc
3	<31:24>	E102, E101	E104, E88, E31, E5	E3	E13
2	<23:16>	E87, E86	E104, E88, E31, E5	E3	E13
1	<15:08>	E70, E69	E104, E88, E31, E5	E3	E81
0	<07:00>	E53, E52	E104, E88, E31, E5	E3	E81

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 8

CSL ACTION: None

VMS ACTION: Standard (See Introduction)

E-04 EBox Result Parity Error (VMQ Shift Operation)

OVERVIEW: This error occurs when the data and parity stored in VMQSAV does not match the new parity generated at the output of the VMQMUX. The VMQSAV parity is generated by the VMQMUX on a previous cycle. Data from the VMQSAV is routed back into in the VMQMUX for a shift operation (left 1 or right 2).

The EBox Result Check is done on the EDP module PDP MCA, and the error signal RESULT PAR ERR is sent to the EBD module where it is latched in the EBCS Register as EBox Data Path PE. The EBox Abort flag is raised and latched into the EBCS Register on the EBD module. In the EDPSR Register, Result Parity Error and EDP Misc error bits are generated and latched on the EDP module PDP MCA.
ERROR SIGNATURE:

EBCS <09> = EBox Data Path Parity Error
EBCS <04> = EBox Abort Flag (Note 1)
EDPSR <05> = Result Parity Error (Note 2)

Note

1. The EBox Abort Flag will set if IRD LST CYC (uBEN field) is set when the error occurs. This flag indicates the error was detected too late to inhibit the PC from being updated and thus, prevents an instruction retry.

2. EDPSR <15:12> VMQ Byte in Error has no meaning for this error.

PROBABLE CAUSE:

Module	Probability	Components
L0209/EDP	High	(See Table 1)
L0211/EBD	Very Low	EBCS Register

Table 1 EDP Component Callout

Byte	Bits	ALU	PDP	Misc
3	<31:24>	E102, E101	E3	E13
2	<23:16>	E87, E86	E3	E13
1	<15:08>	E70, E69	E3	E81
0	<07:00>	E53, E52	E3	E81

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 8

CSL ACTION: None

VMS ACTION: Standard (See Introduction)

E-05 EBox Operand Parity Error (VMQSAV)

OVERVIEW: This error occurs when the AMux is passing data from the VMQSAV Register to the ALU, and the byte parity stored with VMQSAV does not match the byte parity at the output of the AMux. The parity bits stored with VMQSAV are generated at the output of the VMQ MUX.

NOTE

A parity error in VMQSAV is indicated by a lack of any operand error bits set in EDPSR <7:0>.

The EBox Operand Check is done on the EDP Module PDP MCA, and the error signal OPR PAR ERR is sent to the EBD Module where it's latched in the EBCS Register as EBox Data Path PE. The EDPSR Register bits (Operand PE and AMux Byte in Error) are latched on the EDP Module PDP MCA.

ERROR SIGNATURE:

```
EBCS  <09>     = EBox Data Path Parity Error
EDPSR <03>     = Operand Parity Error (AMux or BMux)
EDPSR <07:05>
 and  <02:00> = "0" Indicates VMQSAV was Operand Source
EDPSR <27:24> = AMux Byte in Error
```

PROBABLE CAUSE:

Module	Probability	Components
L0209/EDP	High	(See Table 1)
L0211/EBD	Very Low	EBCS Register

Table 1 EDP Component Callout

Byte	Bits	ALU	PDP	Misc
3	<31:24>	E102, E101	E3	E13
2	<23:16>	E87, E86	E3	E13
1	<15:08>	E70, E69	E3	E81
0	<07:00>	E53, E52	E3	E81

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 8

CSL ACTION: None

VMS ACTION: Standard (See Introduction)

E-06 EBox Operand Error (B WBus)

OVERVIEW: This error occurs when the BMux is passing WBus data to the ALU (WBus Match), and the WBus byte parity to the BMux does not match the parity bits generated at the output of the BMux. WBus byte parity is generated by the EBox on the EBE Module. The BMux parity is generated and checked on the EDP Module.

WBus Match occurs in cases where GPR/SP data is read as operand data, and the previous cycle issued a write to the same location. When the WBus Match occurs, a bypass function takes place where the old (stale) data is not read from the SP RAMs, instead data is selected from the WBus.

When a WBus Match occurs, some data may come from the WBus and some from the GPR/SP. It Depends on the context of the data being written by the previous cycle and the context of the data being read. Therefore, it is possible to have a WBus Error and GPR/SP Error in the same cycle.

The EBox Operand Check is done on the EDP Module PDP MCA, and the error signal OPR PAR ERR is sent to the EBD Module where it's latched in the EBCS Register as EBox Data Path PE. The EDPSR Register bits (Operand PE, WBUS PE, and BMux Byte in Error) are latched on the EDP Module PDP MCA.

ERROR SIGNATURE:

EBCS <09>	= EBox Data Path Parity Error
EDPSR <03>	= Operand Parity Error (AMux or BMux)
EDPSR <07>	= B WBus PE (BMux)
EDPSR <31:28>	= BMux Byte in Error

PROBABLE CAUSE:

Module	Probability	Components
L0209/EDP	High	(See Table 1)
L0219/EBE	Medium	(See Table 2)
L0211/EBD	Very Low	EBCS Register

Table 1 EDP Component Callout

Byte	Bits	ALU	SHF	PDP	Misc
3	<31:24>	E102, E101	E104, E88, E31, E5	E3	E13
2	<23:16>	E87, E86	E104, E88, E31, E5	E3	E13
1	<15:08>	E70, E69	E104, E88, E31, E5	E3	E81
0	<07:00>	E53, E52	E104, E88, E31, E5	E3	E81

Table 2 EBE Component Callout

Byte	WBus Latches	Parity Generators
3	E169, E155, E43, E29	E157, E59
2	E169, E127, E43, E29, E85, E1	E115, E59
1	E169, E127, E43, E155, E1	E45, E31
0	E155, E127, E29, E1	E17, E31

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 8

CSL ACTION: None

VMS ACTION: Standard (See Introduction)

E-07 EBox OPBus Parity Error (EMD Data)

OVERVIEW: This error occurs when the EBox BMux is passing OPBus data to the EBox ALUs, and the input OPBus longword parity from the IBox does not match the parity generated on the output of the EBox BMux. There are several sources in the IBox that feed the OPBus.

The EMD supplies OPBus data to the EBox when the EBox requests the data from memory. EMD parity is not checked in the IBox, but parity for EMD data is passed from the MBox through the IBox to the EBox. The parity for EMD data checked in the EBox is created by collapsing MD Bus odd byte parity into OPBus odd longword parity.

The Data flows from the MBox MCD module MDP MCAs, to the IBox IBD module IBF MCAs to the IOP MCAs, and then to the EBox ALU and PDP MCAs.

NOTE

If MBox Interrupt EBCS <14> is set, the failure is a result of the MBox sending bad data. Therefore this error should be ignored. Instead check for one of the following errors:

MSTAT2 <14> - ABus Bad Data Code

MSTAT1 <03> - Cache Data Parity Error

MDECC <22> - Bad Data Error

MDECC <20> - Data Double Bit Error

MDECC <19> - Data Address Parity Error

ERROR SIGNATURE:

```
EBCS   <09>     = EDP Parity Error
EDPSR  <03>     = Operand Parity Error (AMux PE or BMux PE)
EDPSR  <06>     = B OPBus
IBESR  <09:08>  = "1" (uOPSEL) EMD
EBCS   <14>     = "0" No MBox Interrupt
```

PROBABLE CAUSE:

Module	Probability	Components
L0208/IBD	High	IBF, IOP
L0209/EDP	Medium	ALU, PDP
L0204/MCD	Low	MCD
L0212/FBA	Very Low	SOP, FXP, GXP
L0211/EBD	Very Low	EBCS Register

MANUAL MACHINE CHECK STACK FRAME ANALYSIS

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 8

CSL ACTION: None

VMS ACTION: Standard (See Introduction)

E-08 EBox OPBus Parity Error (String Data)

OVERVIEW: This error occurs when the EBox BMux is passing OPBus data to the EBox ALUs, and the input OPBus longword parity from the IBox does not match the parity generated on the output of the EBox BMux. There are several sources in the IBox that feed the OPBus.

The IBuffer supplies OPBus data to the EBox when the IBox is in string mode and requesting the data from memory. String Data Parity is not checked in the IBox, but the parity for string data is passed from the MBox through the IBox to the EBox. The parity for the string data checked in the EBox is created by collapsing MD Bus odd byte parity into OPBus longword odd parity.

The IBox uses only the parity bits of the bytes that are actually requested by the EBox to create the OPBus longword parity. The remaining bytes contribute parity bits with a value of "0".

The Data flows from the MBox MCD module MDP MCAs, to the IBox IBD module IBF MCAs to the IOP MCAs, and then to the EBox ALU and PDP MCAs.

NOTE

If MBox Interrupt EBCS <14> is set, the failure is a result of the MBox sending bad data. Therefore this error should be ignored. Instead check for one of the following errors:

MSTAT2 <14> - ABus Bad Data Code

MSTAT1 <03> - Cache Data Parity Error

MDECC <22> - Bad Data Error

MDECC <20> - Data Double Bit Error

MDECC <19> - Data Address Parity Error

ERROR SIGNATURE:

```
EBCS   <09>     = EDP Parity Error
EDPSR  <03>     = Operand Parity Error (AMux or BMux)
EDPSR  <06>     = B OPBus
IBESR  <09:08>  = "2" (uOPSEL) IBuffer
EBCS   <14>     = "0" No MBox Interrupt
```

MANUAL MACHINE CHECK STACK FRAME ANALYSIS

PROBABLE CAUSE:

Module	Probability	Components
L0208/IBD	High	IBF, IOP
L0209/EDP	Medium	ALU, PDP
L0204/MCD	Low	MCD
L0212/FBA	Very Low	SOP, FXP, GXP
L0211/EBD	Very Low	EBCS Register

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 8

CSL ACTION: None

VMS ACTION: Standard (See Introduction)

E-09 EBox OPBus Parity Error (IMD Data)

OVERVIEW: This error occurs when the EBox BMux is passing OPBus data to the EBox ALUs, and the input OPBus longword parity from the IBox does not match the parity generated on the output of the EBox BMux. There are several sources in the IBox that feed the OPBus.

The IMD supplies OPBus data to the EBox when the IBox requests the data from memory. IMD operand parity is not checked in the IBox, but parity for IMD data is passed from the MBox through the IBox to the EBox. The parity for IMD data checked in the EBox is created by collapsing MDBus odd byte parity into OPBus longword odd parity.

The Data flows from the MBox MCD module MDP MCAs, to the IBox IBD module IBF MCAs to the IOP MCAs, and then to the EBox ALU and PDP MCAs.

NOTE

1. Indirect address data in IMD takes a different path and is parity checked by the IBMux parity check logic.

2. If MBox Interrupt EBCS <14> is set, the failure is a result of the MBox sending bad data. Therefore this error should be ignored. Instead check for one of the following errors:

   MSTAT2 <14> - ABus Bad Data Code

   MSTAT1 <03> - Cache Data Parity Error

   MDECC <22> - Bad Data Error

   MDECC <20> - Data Double Bit Error

   MDECC <19> - Data Address Parity Error

ERROR SIGNATURE:

```
EBCS   <09>     = EDP Parity Error
EDPSR  <03>     = Operand Parity Error (AMux PE or BMux PE)
EDPSR  <06>     = B OPBus
IBESR  <10>     = "1" Source IMD
IBESR  <09:08>  = "3" (uOPSEL) IMD or ID see bit 10.
EBCS   <14>     = "0" No MBox Interrupt
```

PROBABLE CAUSE:

Module	Probability	Components
L0208/IBD	High	IBF, IOP
L0209/EDP	Medium	ALU, PDP
L0204/MCD	Low	MCD
L0212/FBA	Very Low	SOP, FXP, GXP
L0211/EBD	Very Low	EBCS Register

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 8

CSL ACTION: None

VMS ACTION: Standard (See Introduction)

E-10 EBox OPBus Parity Error (ID Data)

OVERVIEW: This error occurs when the EBox BMux is passing OPBus data to the EBox ALUs, and the input OPBus longword parity from the IBox does not match the parity generated on the output of the EBox BMux. There are several sources in the IBox that feed the OPBus.

Parity for operands supplied to the EBox from the IBox GPRs and Instruction Buffer go to the EBox via the ID Latch to the OPBus. The parity supplied to the EBox for this data is created by collapsing IAMux or IBMux odd byte parity into OPBus longword odd parity.

Data flows from the IAMux or IBMux Latch on the IDP Module IAD MCA, to the VA Latch and ID Latch on the IDP Module IVA and UPK MCA, and then over the OPBus to the EBox EDP Module ALU and PDP MCAs.

ERROR SIGNATURE:

```
EBCS  <09>     = EDP Parity Error
EDPSR <03>     = Operand Parity Error (AMux PE or BMux PE)
EDPSR <06>     = B OPBus
IBESR <10>     = "0" Source ID
IBESR <09:08> = "3" (uOPSEL) IMD or ID see bit 10.
```

PROBABLE CAUSE:

Modules	Probability	Components
L0206/IDP	High	IAD, IVA
L0209/EDP	Medium	ALU, PDP
L0206/IDP	Low	UPK, DPP
L0212/FBA	Very Low	SOP, FXP, GXP
L0211/EBD	Very Low	EBCS Register

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 8

CSL ACTION: None

VMS ACTION: Standard (See Introduction)

E-11 EBox Operand Parity Error (A RAM)

OVERVIEW: This error occurs when the AMux is passing GPRA/SP data to the ALU, and the parity generated at the AMux output does not match the GPRA/SP byte parity stored in the RAMs.

All data and parity written into the GPR/SP comes from the WBus. The EBox generates all WBus parity, even when the IBox and FBox are driving data on the WBus. When the EBox drives the WBus, it compares its internal parity to the parity on the WBus, if they do not match the EBox asserts WBus Parity Error.

The EBox Operand Check is done on the EDP Module PDP MCA, and the error signal OPR PAR ERR is sent to the EBD Module where it's latched in the EBCS Register as EBox Data Path PE. The EDPSR Register bits (Operand PE, A RAM PE, and AMux Byte in Error) are latched on the EDP module PDP MCA.

ERROR SIGNATURE:

EHMSTS	<25>	= EBox SP B to A
EBCS	<09>	= EBox Data Path Parity Error
EDPSR	<03>	= Operand Parity Error (AMux or BMux)
EDPSR	<02>	= A RAM PE (AMux PE)
EDPSR	<27:24>	= AMux Byte in Error

PROBABLE CAUSE:

Module	Probability	Components
L0209/EDP	High	(See Table 1)
L0219/EBE	Medium	(See Table 2)
L0211/EBD	Very Low	EBCS Register

Table 1 EDP Component Callout

Byte	Bits	RAMs	ALU	PDP	Misc
3	<31:24>	E500, E501, E903	E102, E101	E3	E13
2	<23:16>	E502, E503, E903	E87, E86	E3	E13
1	<15:08>	E616, E615, E903	E70, E69	E3	E81
0	<07:00>	E614, E613, E903	E53, E52	E3	E81

Table 2 EBE Component Callout

Byte	WBus Latches	Parity Generators
3	E169, E155, E43, E29	E157, E59
2	E169, E127, E43, E29, E85, E1	E115, E59
1	E169, E127, E43, E155, E1	E45, E31
0	E155, E127, E29, E1	E17, E31

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 8

CSL ACTION: None

VMS ACTION: Standard (See Introduction)

E-12 EBox Operand Error (A WBus)

OVERVIEW: This error occurs when the AMux is passing WBus data to the ALU (WBus Match), and the WBus byte parity to the AMux does not match the parity bits generated at the output of the AMux. WBus byte parity is generated by the EBox on the EBE module. The AMux parity is generated and checked on the EDP module.

WBus Match occurs in cases where GPR/SP data is read as operand data, and the previous cycle issued a write to the same location. When the WBus Match occurs, a bypass function takes place where the old (stale) data is not read from the SP RAMs, instead data is selected from the WBus.

When a WBus Match occurs, some data may come from the WBus and some from the GPR/SP. It Depends on the context of the data being written by the previous cycle and the context of the data being read. Therefore, it is possible to have a WBus Error and GPR/SP Error in the same cycle.

The EBox Operand Check is done on the EDP Module PDP MCA, and the error signal OPR PAR ERR is sent to the EBD Module where it's latched in the EBCS Register as EBox Data Path PE. The EDPSR Register bits (Operand PE, A WBus PE, and AMux Byte in Error) are generated and latched on the EDP module PDP MCA.

ERROR SIGNATURE:

```
EBCS   <09>     = EBox Data Path Parity Error
EDPSR  <03>     = Operand Parity Error (AMux or BMux)
EDPSR  <01>     = A WBus PE (AMux)
EDPSR  <27:24>  = AMux Byte in Error
```

PROBABLE CAUSE:

Module	Probability	Components
L0209/EDP	High	(See Table 1)
L0219/EBE	Medium	(See Table 2)
L0211/EBD	Very Low	EBCS Register

Table 1 EDP Component Callout

Byte	Bits	ALU	SHF	PDP	Misc
3	<31:24>	E102, E101	E104, E88, E31, E5	E3	E13
2	<23:16>	E87, E86	E104, E88, E31, E5	E3	E13
1	<15:08>	E70, E69	E104, E88, E31, E5	E3	E81
0	<07:00>	E53, E52	E104, E88, E31, E5	E3	E81

Table 2 EBE Component Callout

Byte	WBus Latches	Parity Generators
3	E169, E155, E43, E29	E157, E59
2	E169, E127, E43, E29, E85, E1	E115, E59
1	E169, E127, E43, E155, E1	E45, E31
0	E155, E127, E29, E1	E17, E31

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 8

CSL ACTION: None

VMS ACTION: Standard (See Introduction)

E-13 EBox Operand Parity Error (B RAM)

OVERVIEW: This error occurs when the BMux is passing GPRB/SP data to the ALU and the parity generated at the BMux output does not match the GPRB/SP byte parity stored in the RAMs.

All data and parity written into the GPR/SP comes from the WBus. The EBox generates all WBus parity, even when the IBox and FBox are driving data on the WBus. When the EBox drives the WBus, it compares its internal parity to the parity on the WBus, if they do not match the EBox asserts WBus Parity Error.

The EBox Operand Check is done on the EDP Module PDP MCA, and the error signal OPR PAR ERR is sent to the EBD Module where it's latched in the EBCS Register as EBox Data Path PE. The EDPSR Register bits (Operand PE, B RAM PE, and BMux Byte in Error) are latched on the EDP module PDP MCA.

ERROR SIGNATURE:

```
EHMSTS <26>     = EBox SP A to B
EBCS   <09>     = EBox Data Path Parity Error
EDPSR  <03>     = OPERAND Parity Error (AMux or BMux)
EDPSR  <00>     = B RAM PE (BMux PE)
EDPSR  <31:28> = BMux Byte in Error
```

PROBABLE CAUSE:

Module	Probability	Components
L0209/EDP	High	(See Table 1)
L0219/EBE	Medium	(See Table 2)
L0211/EBD	Very Low	EBCS Register

Table 1 EDP Component Callout

Byte	Bits	RAMs	ALU	PDP	Misc
3	<31:24>	E715, E714, E907	E102, E101	E3	E13
2	<23:16>	E713, E712, E907	E87, E86	E3	E13
1	<15:08>	E815, E814, E907	E70, E69	E3	E81
0	<07:00>	E813, E812, E907	E53, E52	E3	E81

Table 2 EBE Component Callout

Byte	WBus Latches	Parity Generators
3	E169, E155, E43, E29	E157, E59
2	E169, E127, E43, E29, E85, E1	E115, E59
1	E169, E127, E43, E155, E1	E45, E31
0	E155, E127, E29, E1	E17, E31

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 8

CSL ACTION: None

VMS ACTION: Standard (See Introduction)

E-14 EBox Micro Stack Parity Error

OVERVIEW: This error occurs when the EBox Micro-Stack is read (Popped) and the Micro-Stack location contains incorrect parity. The Micro-Stack parity is generated on the microsequencer (MIC MCA) when data is pushed onto the Micro-Stack.

The EBox Micro stack Parity Error signal is generated on the CSB module and sent to the EBD module where it's latched in the EBCS Register.

ERROR SIGNATURE:

EBCS <10> = EBox Ustack Parity Error

PROBABLE CAUSE:

Module	Probability	Components
L0216/CSB	High	RAMs - E160, E166, E172, E179
L0216/CSB	Medium	Checker
L0211/EBD	Very Low	EBCS Register

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 8

CSL ACTION: None

VMS ACTION: Standard (See Introduction)

E-15 EBox Control Store Parity Error

OVERVIEW: This error occurs when a microword is read out of the Control Store RAMs into the data latches. EBox Control Store is spread across two modules: CSA and CSB. There are two parity checks, one for the CSA module and one for the CSB module. The error signals are or'ed together on CSB to produce EBox CS PE. EBox CS parity error is then sent to the EBD module where it's latched in the EBCS Register.

EBox Control Store Parity Errors are handled differently then are Control Store Parity Errors for the other boxes. The EBox will stall inhibiting clocks to the data latches and thus freeze the data and address in error. The EBox will then interrupt the console using the CPU ERROR CODE lines <2:0> from the EBE module.

The console will receive the interrupt and determine that it was caused by an EBox Control Store Parity Error. The Console will then read the bad microword and address over the SDB and try ECC correction on the microword. If the correction fails, the Console will re-initialize the CPU and notify VMS. If the correction succeeds (single bit error), then the console will write the corrected word back into the EBox CS RAMs. Thenit will read the corrected word out of the RAMS and re-check the parity to verify that the RAM location is correct.

ERROR SIGNATURE:

```
EBCS <11>     = EBox Control Store Parity Error
CSES <31>     = Correctable Error
                  0 = Error corrected
                  1 = Unable to correct error
CSES <28:16> = Control Store Address
CSES <15:08> = Syndrome
CSES <02:00> = "1" EBox Control Store Parity Error
```

NOTE

If ECC Correction fails the Console will re-initialize the CPU and notify VMS.

PROBABLE CAUSE:

Module	Probability	Components	CSA/CSB PE	
L0216/CSB	High	RAMs (See Table 1)		CSB
L0215/CSA	High	RAMs (See Table 2)	CSA	
L0209/EDP	Medium	Receivers	CSA	
L0206/IDP	Low	Receiver		CSB
L0210/EBC	Low	Receivers	CSA	
L0207/ICA	Low	Receivers	CSA	
L0213/FBM	Low	Receivers	CSA	
L0211/EBD	Very Low	EBCS Reg/Receivers	CSA	CSB
L0217/L0231/CLK	Very Low	Receiver	CSA	CSB
L0219/EBE	Very Low	CPU Error Code		

Table 1 CSB Control Store RAM Call Out (See Note 1)

Syn	Phy	ULD	Sig Name	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
1	00	00	UJUMP 00	E95	E100	E106	E111	E116	E123	E128	E134
2	01	01	UJUMP 01	E95	E100	E106	E111	E116	E123	E128	E134
3	02	02	UJUMP 02	E95	E100	E106	E111	E116	E123	E128	E134
4	03	03	UJUMP 03	E95	E100	E106	E111	E116	E123	E128	E134
5	04	04	UJUMP 04	E94	E99	E105	E110	E115	E122	E127	E123
6	05	05	UJUMP 05	E94	E99	E105	E110	E115	E122	E127	E123
7	06	06	UJUMP 06	E94	E99	E105	E110	E115	E122	E127	E123
8	07	07	UJUMP 07	E94	E99	E105	E110	E115	E122	E127	E123
9	08	08	UJUMP 08	E93	E98	E104	E109	E114	E121	E126	E132
A	09	09	UJUMP 09	E93	E98	E104	E109	E114	E121	E126	E132
B	10	10	UJUMP 10	E93	E98	E104	E109	E114	E121	E126	E132
C	11	11	UJUMP 11	E93	E98	E104	E109	E114	E121	E126	E132
D	12	12	UJUMP 12	E66	E59	E54	E49	E43	E38	E31	E25
E	13	13	USUB 00	E66	E59	E54	E49	E43	E38	E31	E25
F	14	14	USUB 01	E66	E59	E54	E49	E43	E38	E31	E25
10	15	15	UBEN 00	E66	E59	E54	E49	E43	E38	E31	E25
11	16	16	UBEN 01	E67	E60	E55	E50	E44	E39	E32	E26
12	17	17	UBEN 02	E67	E60	E55	E50	E44	E39	E32	E26
13	18	18	UBEN 03	E67	E60	E55	E50	E44	E39	E32	E26
14	19	19	UBEN 04	E67	E60	E55	E50	E44	E39	E32	E26
15	20	37	USRC1 7	E68	E61	E56	E51	E45	E40	E33	E27
16	21	82	DATA PAR	E68	E61	E56	E51	E45	E40	E33	E27
17	22	3	USRC1 6	E68	E61	E56	E51	E45	E40	E33	E27
18	23	35	USRC1 5	E68	E61	E56	E51	E45	E40	E33	E27
19	24	34	USRC1 4	E69	E62	E57	E52	E46	E41	E34	E28
1A	25	33	USRC1 3	E69	E62	E57	E52	E46	E41	E34	E28
1B	26	32	USRC1 2	E69	E62	E57	E52	E46	E41	E34	E28
1C	27	31	USRC1 1	E69	E62	E57	E52	E46	E41	E34	E28
1D	28	47	UDEST 3	E65	E58	E53	E48	E42	E37	E30	E24
1E	29	63	UMCF 0	E65	E58	E53	E48	E42	E37	E30	E24
1F	30	64	UMCF 1	E65	E58	E53	E48	E42	E37	E30	E24
20	31	65	UMCF 2	E65	E58	E53	E48	E42	E37	E30	E24
21	32	61	UOPSEL 0	E91	E96	E102	E107	E112	E119	E124	E130
22	33	62	UOPSEL 1	E91	E96	E102	E107	E112	E119	E124	E130
23	34	39	USRC2 1	E91	E96	E102	E107	E112	E119	E124	E130
24	35	38	USRC2 0	E91	E96	E102	E107	E112	E119	E124	E130
25	36	49	UDEST 5	E92	E97	E103	E108	E113	E120	E125	E131
26	37	48	UDEST 4	E92	E97	E103	E108	E113	E120	E125	E131
27	38	30	USRC1 0	E92	E97	E103	E108	E113	E120	E125	E131
28	39	40	USRC2 2	E92	E97	E103	E108	E113	E120	E125	E131

NOTE

1. The numbers above the Estate (E ###) numbers correspond to the bits <18:16> in CSES.

Table 1A CSB Control Store RAM Terminating Module

Syn	Phy	ULD	Signal Name	Module	Slot	Chip	Pin
1	00	00	UJUMP 00	CSB	4	E140	P01
2	01	01	UJUMP 01	CSB	4	E140	P14
3	02	02	UJUMP 02	CSB	4	E140	P15
4	03	03	UJUMP 03	CSB	4	E140	P02
5	04	04	UJUMP 04	CSB	4	E139	P14
6	05	05	UJUMP 05	CSB	4	E139	P15
7	06	06	UJUMP 06	CSB	4	E139	P02
8	07	07	UJUMP 07	CSB	4	E139	P01
9	08	08	UJUMP 08	CSB	4	E138	P01
A	09	09	UJUMP 09	CSB	4	E138	P14
B	10	10	UJUMP 10	CSB	4	E138	P02
C	11	11	UJUMP 11	CSB	4	E138	P15
D	12	12	UJUMP 12	CSB	4	E20	P14
E	13	13	USUB 00	CSB	4	E20	P15
F	14	14	USUB 01	CSB	4	E20	P01
10	15	15	UBEN 00	CSB	4	E20	P02
11	16	16	UBEN 01	CSB	4	E21	P14
12	17	17	UBEN 02	CSB	4	E21	P15
13	18	18	UBEN 03	CSB	4	E21	P02
14	19	19	UBEN 04	CSB	4	E21	P01
15	20	37	USRC1 7	EDP	10	A54	
16	21	82	UPAR 0	CSB	4	E22	P02
17	22	36	USRC1 6	EDP	10	B06	
18	23	35	USRC1 5	EDP	10	B56	
19	24	34	USRC1 4	EDP	10	B10	
1A	25	33	USRC1 3	EDP	10	A56	
1B	26	32	USRC1 2	EDP	10	A01	
1C	27	31	USRC1 1	EDP	10	A52	
1D	28	47	UDEST 3	EDP	10	A70	
1E	29	63	UMCF 0	IDP	14	A27	
1F	30	64	UMCF 1	IDP	14	A12	
20	31	65	UMCF 2	IDP	14	A03	
21	32	61	UOPSEL 0	IDP	14	C75	
22	33	62	UOPSEL 1	IDP	14	C85	
23	34	39	USRC2 1	EDP	10	A58	
24	35	38	USRC2 0	EDP	10	A59	
25	36	49	UDEST 5	EDP	10	A68	
26	37	48	UDEST 4	EDP	10	A62	
27	38	30	USRC1 0	EDP	10	A64	
28	39	40	USRC2 2	EDP	10	A63	

NOTE

All terminators should measure 56 ohms to -2 volts.

Table 2 CSA Control Store RAMs Call Out (See Note 1)

Syn	Phy	ULD	Sig Name	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
29	40	60	ULIT 5	E82	E75	E69	E62	E56	E50	E43	E37
2A	41	59	ULIT 4	E82	E75	E69	E62	E56	E50	E43	E37
2B	42	58	ULIT 3	E82	E75	E69	E62	E56	E50	E43	E37
2C	43	57	ULIT 2	E82	E75	E69	E62	E56	E50	E43	E37
2D	44	25	UAMX 0	E83	E76	E70	E63	E57	E51	E44	E38
2E	45	52	USCK 2	E83	E76	E70	E63	E57	E51	E44	E38
2F	46	51	USCK 1	E83	E76	E70	E63	E57	E51	E44	E38
30	47	50	USCK 0	E83	E76	E70	E63	E57	E51	E44	E38
31	48	68	UMCF 5	E84	E77	E71	E64	E58	E52	E45	E39
32	49	67	UMCF 4	E84	E77	E71	E64	E58	E52	E45	E39
33	50	66	UMCF 3	E84	E77	E71	E64	E58	E52	E45	E39
34	51	83	PAR	E84	E77	E71	E64	E58	E52	E45	E39
35	52	91	SPARE 0	E85	E78	E72	E65	E59	E53	E46	E40
36	53	90	SPARE 1	E85	E78	E72	E65	E59	E53	E46	E40
37	54	89	SPARE 2	E85	E78	E72	E65	E59	E53	E46	E40
38	55	88	SPARE 3	E85	E78	E72	E65	E59	E53	E46	E40
39	56	81	UMISC 3	E86	E79	E73	E66	E60	E54	E47	E41
3A	57	80	UMISC 2	E86	E79	E73	E66	E60	E54	E47	E41
3B	58	79	UMISC 1	E86	E79	E73	E66	E60	E54	E47	E41
3C	59	78	UMISC 0	E86	E79	E73	E66	E60	E54	E47	E41
3D	60	87	SPARE 4	E32	E33	E27	E28	E34	E29	E35	E36
3E	61	86	SPARE 5	E32	E33	E27	E28	E34	E29	E35	E36
3F	62	85	CC SYNC	E32	E33	E27	E28	E34	E29	E35	E36
40	63	84	NODEST	E32	E33	E27	E28	E34	E29	E35	E36
41	64	29	USMI 0	E87	E80	E74	E67	E61	E55	E48	E42
42	65	26	UBMX 0	E87	E80	E74	E67	E61	E55	E48	E42
43	66	28	UVMQK 1	E87	E80	E74	E67	E61	E55	E48	E42
44	67	27	UVMQK 0	E87	E80	E74	E67	E61	E55	E48	E42
45	68	46	UDEST 2	E119	E125	E132	E138	E144	E151	E157	E164
46	69	45	UDEST 1	E119	E125	E132	E138	E144	E151	E157	E164
47	70	24	UALU 4	E119	E125	E132	E138	E144	E151	E157	E164
48	71	23	UALU 3	E119	E125	E132	E138	E144	E151	E157	E164
49	72	22	UALU 2	E118	E124	E131	E137	E143	E150	E156	E163
4A	73	69	UMARK 0	E118	E124	E131	E137	E143	E150	E156	E163
4B	74	21	UALU 1	E118	E124	E131	E137	E143	E150	E156	E163
4C	75	20	UALU 0	E118	E124	E131	E137	E143	E150	E156	E163
4D	76	75	UCCK 3	E117	E123	E130	E136	E142	E149	E155	E162
4E	77	74	UCCK 2	E117	E123	E130	E136	E142	E149	E155	E162
4F	78	73	UCCK 1	E117	E123	E130	E136	E142	E149	E155	E162
50	79	72	UCCK 0	E117	E123	E130	E136	E142	E149	E155	E162
51	80	77	UDT 1	E116	E122	E129	E135	E141	E148	E154	E161
52	81	76	UDT 0	E116	E122	E129	E135	E141	E148	E154	E161
53	82	71	UFBOX 1	E116	E122	E129	E135	E141	E148	E154	E161
54	83	70	UFBOX 0	E116	E122	E129	E135	E141	E148	E154	E161
55	84	43	USRC2 5	E115	E121	E128	E134	E140	E147	E153	E160

Table 2 CSA Control Store RAMs Call Out (cont.) (See Note 1)

Syn	Phy	ULD	Signal Name	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
56	85	42	USRC2 4	E115	E121	E128	E134	E140	E147	E153	E160
57	86	41	USRC2 3	E115	E121	E128	E134	E140	E147	E153	E160
58	87	44	UDEST 0	E115	E121	E128	E134	E140	E147	E153	E160
59	88	56	ULIT 1	E114	E120	E127	E133	E139	E146	E152	E159
5A	89	55	ULIT 0	E114	E120	E127	E133	E139	E146	E152	E159
5B	90	54	ULITCTL 1	E114	E120	E127	E133	E139	E146	E152	E159
5C	91	53	ULITCTL 0	E114	E120	E127	E133	E139	E146	E152	E159

NOTE

1. The numbers above the Estate (E ###) numbers correspond to the bits <18:16> in CSES.

Table 2A CSA Control Store RAM Terminating Module

Syn	Phy	ULD	Signal Name	Module	Slot	Chip	Pin
29	40	60	ULIT 5	EDP	10	A89	
2A	41	59	ULIT 4	EDP	10	A04	
2B	42	58	ULIT 3	EDP	10	A78	
2C	43	57	ULIT 2	EDP	10	A88	
2D	44	25	UAMX	EDP	10	B64	
2E	45	52	USCK 2	EDP	10	A76	
2F	46	51	USCK 1	EDP	10	A84	
30	47	50	USCK 0	EDP	10	A86	
31	48	68	UMCF 5	CSA	3	E20	P01
32	49	67	UMCF 4	CSA	3	E20	P15
33	50	66	UMCF 3	CSA	3	E20	P02
34	51	83	UPAR	CSA	3	E20	P14
35	52	91	SPARE 0	CSA	3	A12	
36	53	90	SPARE 1	CSA	3	A06	
37	54	89	SPARE 2	CSA	3	A08	
38	55	88	SPARE 3	CSA	3	A38	
39	56	81	UMISC 3	EDP	10	B66	
3A	57	80	UMISC 2	EDP	10	B60	
3B	58	79	UMISC 1	EDP	10	A55	
3C	59	78	UMISC 0	EDP	10	A46	
3D	60	87	SPARE 4	CSA	3	A01	
3E	61	86	SPARE 5	CSA	3	A04	
3F	62	85	CC SYNC	ICA	13	B78	
40	63	84	NODEST	ICA	13	A05	
41	64	29	USMI	EBD	6	A40	

Table 2A CSA Control Store RAM Terminating Module (cont.)

Syn	Phy	ULD	Signal Name	Module	Slot	Chip	Pin
42	65	26	UBMX	EDP	10	B58	
43	66	28	UVMQK 1	EDP	10	B68	
44	67	27	UVMQK 0	EDP	10	B91	
45	68	46	UDEST 2	EDP	10	A57	
46	69	45	UDEST 1	EDP	10	A50	
47	70	24	UALU 4	EDP	10	A44	
48	71	23	UALU 3	EDP	10	B82	
49	72	22	UALU 2	EDP	10	B38	
4A	73	69	UMARK	CLK	11	B45	
4B	74	21	UALU 1	EDP	10	B30	
4C	75	20	UALU 0	EDP	10	B32	
4D	76	75	UCCK 3	EDP	10	B18	
4E	77	74	UCCK 2	EDP	10	B20	
4F	78	73	UCCK 1	EDP	10	A20	
50	79	72	UCCK 0	EDP	10	B22	
51	80	77	UDT 1	EBC	5	C69	
52	81	76	UDT 0	EBC	5	C80	
53	82	71	UFBOX 1	FBM	7	B69	
54	83	70	UFBOX 0	FBM	7	B64	
55	84	43	USRC2 5	EDP	10	B54	
56	85	42	USRC2 4	EDP	10	B12	
57	86	41	USRC2 3	EDP	10	A73	
58	87	44	UDEST 0	EDP	10	A48	
59	88	56	ULIT 1	EDP	10	B03	
5A	89	55	ULIT 0	EDP	10	A80	
5B	90	54	ULITCTL 1	EDP	10	A91	
5C	91	53	ULITCTL 0	EDP	10	A94	

NOTE

All terminators should measure 56 ohms to -2 volts.

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 8

CSL ACTION: Standard (See Introduction) In addition, since the EBox hardware interrupts the console directly, the console will (via the SDB) generate CSPE Reset on the CSB module. This will cause the EBox microcode to re-start the EHM at vector 8.

VMS ACTION: Standard (See Introduction)

E-16 EBox MCF RAM Parity Error

OVERVIEW: This error occurs when even parity is detected at the output of the EBox Memory Control Field (MCF) RAMs on the EBC module. These RAMs are accessed every time a new EBox microword is read. All unused locations are loaded with even parity, so addressing an unused location will result in a parity error.

The MCF RAM parity check is done on the EBC module, and the error signal is sent to the EBD module where it's latched in the EBCS Register.

ERROR SIGNATURE:

EBCS <12> = EBox MCF RAM Parity Error
EBCS <04> = EBox Abort Flag (Note 1)

Note

1. The EBox Abort Flag will set if IRD LST CYC (uBEN field) is set when the error occurs. This flag indicates the error was detected too late to inhibit the PC from being updated and thus, prevents an instruction retry.

PROBABLE CAUSE:

Module	Probability	Components
L0210/EBC	High	RAMs - E101, E102, E103, E104
L0210/EBC	Medium	Latches/Checkers
L0211/EBD	Very Low	EBCS Register

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 8

CSL ACTION: None

VMS ACTION: Standard (See Introduction)

I-01 IBox Control Store Parity Error

OVERVIEW: IBox Control Store parity is checked at the output of the control store data latches. The IBox Control Store has odd parity across its 51-bit field.

An IBox Control Store Parity Error will inhibit further clocking of both the Control Store Data Latches and the UPC Save register. At the same time it will cause a microtrap to the EHM. The EHM will begin building a Machine Check Stack Frame, determine that it was called to handle an IBox CS PE, and set EBCS <27>. This in turn will interrupt the Console by asserting CPU ERROR <2:0> lines from the EBE module. The Console will correct the parity error and return control to the EHM. The EHM will continue building the Stack Frame and call the VMS Machine Check Handler.

Data flows from the ICA Module CS RAMs, to the CS Data Latches, to the CS Parity Checkers and then to the IBox IDP, IBD, and ICB Modules. The Control Store Parity Error signal (ICS PE) goes from the ICA Module to the EBE Module where it is latched in IBESR <21>.

ERROR SIGNATURE:

```
EBCS  <13>     = IBox Error
IBESR <21>     = ICS Parity Error
CSES  <31>     = Correctable Error
                   0 = Error corrected
                   1 = Unable to correct error
CSES  <28:16>  = Control Store Address
CSES  <15:08>  = Syndrome
CSES  <02:00>  = "2" ICS Parity Error
```

PROBABLE CAUSE:

Modules	Probability	Components
L0207/ICA	High	RAMs (see Table 1)
L0207/ICA	Medium	CS Data Latches/Parity Checker
L0206/IDP	Medium	Receivers
L0214/ICB	Low	Receivers
L0208/IBD	Low	Receivers
L0219/EBE	Very Low	IBESR Register

Table 1 IBox Control Store RAM Callout

Syndrome	Phy	ULD	Signal Name	RAM
1	00	24	GPR SEL 0	E16
2	01	25	GPR SEL 1	E16
3	02	26	GPR SEL 2	E16
4	03	15	AMUX SEL 0	E16
5	04	16	AMUX SEL 1	E13
6	05	17	AMUX SEL 2	E13
7	06	21	CTX CTL 0	E13
8	07	22	CTX CTL 1	E13
9	08	23	CTX CTL 2	E14
A	09	18	BMUX SEL 0	E15
B	10	19	BMUX SEL 1	E15
C	11	20	BMUX SEL 2	E15
D	12	37	UMISC 0	E15
E	13	38	UMISC 1	E14
F	14	39	UMISC 2	E14
10	15	40	UMISC 3	E14
11	16	10	IFORK CTL 0	E36
12	17	11	IFORK CTL 1	E36
13	18	12	IFORK CTL 2	E36
14	19	13	UTRAP CTL 0	E35
15	20	14	UTRAP CTL 0	E35
16	21	08	UBEN CTL 0	E35
17	22	09	UBEN CTL 1	E35
18	23	41	CYCLE ID 0	E36
19	24	42	CYCLE ID 1	E12
1A	25	29	OPV CTL 0	E12
1B	26	30	OPV CTL 1	E12
1C	27	33	UMCF 0	E12
1D	28	34	UMCF 1	E11
1E	29	35	UMCF 2	E11
1F	30	36	UMCF 3	E11
20	31	43	UNSTALL	E11
21	32	45	DIAG UNSTALL	E40
22	33	27	UCODE WREQ	E40
23	34	28	REG MODE	E40
24	35	00	NA 0	E40
25	36	01	NA 1	E39
26	37	02	NA 2	E39
27	38	03	NA 3	E39
28	39	04	NA 4	E39
29	40	05	NA 5	E37
2A	41	06	NA 6	E37
2B	42	07	NA 7	E38
2C	43	31	UNPACK CTL 0	E38
2D	44	32	UNPACK CTL 1	E38

Table 1 IBox Control Store RAM Callout (cont.)

Syndrome	Phy	ULD	Signal Name	RAM
2E	45	46	IBOX MARK	E38
2F	46	44	INH IBF SHIFT	E37
30	47	50	ICS OPAR	E37
31	48	47	UMISC2 0	E17
32	49	48	UMISC2 1	E17
33	50	49	UMISC2 2	E17

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 10, 18, 1E, or 1F

CSL ACTION: Standard (See Introduction)

VMS ACTION: Standard (See Introduction)

I-02 IBox IDRAM Parity Error

OVERVIEW: The IBox DRAM has odd parity across its 20-bit field. When a DRAM parity error occurs the failing Address and Data will be held in the ERR/LD ADRS register, and a microtrap to the EHM will occur.

The EHM will begin building a Machine Check Stack Frame, determine that it was called to handle an IBox DRAM PE, and set EBCS <28>. This in turn will interrupt the console by asserting CPU ERROR <2:0> lines from the EBE Module. The Console will correct the parity error and return control to the EHM. The EHM will continue building the Stack Frame and call the VMS Machine Check Handler.

Data flows from the IBD Module DRAMs to the DRAM Data Latches, and then to two IOPA MCAs (E7, E20) which do the parity checking. The parity error signal from IOP MCA is fed through the DBS MCA, to the ICB Module, and then to the EBox EBE Module where it is latched in IBESR <22>.

ERROR SIGNATURE:

```
EBCS   <13>     = IBox Error
IBESR  <22>     = IDRAM Parity Error
CSES   <31>     = Correctable Error
                     0 = Error corrected
                     1 = Unable to correct error
CSES   <28:16>  = Control Store Address
CSES   <15:08>  = Syndrome
CSES   <02:00>  = "3" IDRAM Parity Error
```

PROBABLE CAUSE:

Modules	Probability	Components
L0208/IBD	High	RAMs (see Table 1)
L0208/IBD	Medium	DRAM Data Latches/Checkers(IOPA)
L0219/EBE	Very Low	IBESR Register

Table 1 IBox DRAM Callout

Syndrome	Bit	Signal Name	Dram
1	D00	NAT ADRS 00	E82
2	D01	NAT ADRS 01	E53
3	D02	NAT ADRS 02	E55
4	D03	NAT ADRS 03	E80
5	D04	NAT ADRS 04	E56
6	D05	NAT ADRS 05	E54
7	D06	NAT FPA	E81
8	D07	NAT OPAR	E52
9	D08	NAT LAST	E88
A	D09	NAT BDEST NXT	E79

Table 1 IBox DRAM Callout (cont.)

Syndrome	Bit	Signal Name	Dram
B	D10	NAT SUSPEND	E78
C	D11	NAT CTL 0	E74
D	D12	NAT CTL 1	E75
E	D13	NAT REF 0	E50
F	D14	NAT REF 1	E76
10	D15	NAT TYPE 0	E51
11	D16	NAT TYPE 1	E77
12	D17	NAT CTX 0	E95
13	D18	NAT CTX 1	E96
14	D19	NAT CTX 2	E89

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 10, 18, 1E, or 1F

CSL ACTION: Standard (See Introduction)

VMS ACTION: Standard (See Introduction)

I-03 IBox IAMux WBus PARITY

OVERVIEW: This error occurs when a WBus Match occurs in the IBox and a WBus byte parity error is detected at the output of the IBox AMux. Parity is only checked on the bytes consumed.

A WBus Match will occur when the GPR data required for an operation is in the process of being updated (written) by the previous operation. Instead of waiting for the previous operation to complete the GPR write cycle the data is taken directly off the WBus.

The EBox generates all WBus and GPR parity bits on the EBE Module. IAMux EC <1:0> indicates the most significant byte in error.

The IAMux Parity Error signal (IAMux PE), IAMux EC <1:0>, and IAMux WBus DATA go from the IDP module to the EBE module where they are latched in IBESR.

ERROR SIGNATURE:

```
EBCS  <13>    = IBox Error
IBESR <30:24> = IAMux EC <1:0> (Byte in Error)
IBESR <28>    = "1" IWBus Data
IBESR <23>    = IAMux Parity Error
```

PROBABLE CAUSE:

Modules	Probability	Components
L0206/IDP	High	IAD, DPPA (see Table 1)
L0219/EBE	Medium	WBus Parity Generators
L0219/EBE	Very low	IBESR Register

Table 1 IAD MCA Callout

IAMux EC <1:0>	Data Bit	Byte	Slice	IAD MCA
0	0-3	0	0	E1
0	4-7	0	1	E2
1	8-11	1	2	E1
1	12-15	1	3	E2
2	16-19	2	4	E5
2	20-23	2	5	E7
3	24-27	3	6	E8
3	28-31	3	7	E8

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 10, 18, 1E, or 1F

CSL ACTION: Standard (See Introduction)

MANUAL MACHINE CHECK STACK FRAME ANALYSIS

VMS ACTION: Standard (See Introduction)

I-04 IBox IAMux GPR Parity Error

OVERVIEW: This error occurs when the IBox AMux is passing GPR data to the Data Path Adder, and the parity generated at the output of the IAMux does not match the GPR Byte parity stored in the RAMs. Parity is only checked on the bytes consumed. Parity is not checked when a GPR is used as an index register [Rx].

The EBox generates all WBus and GPR parity bits on the EBE Module. IAMux EC <1:0> indicates the most significant byte in error.

The IAMux Parity Error signal (IAMux PE), and IAMux EC <1:0> go from the IDP module to the EBE module where they are latched in IBESR.

ERROR SIGNATURE:

```
EBCS  <13>     = IBox Error
IBESR <28>     = "0" GPR Data
IBESR <23>     = "1" IAMux Parity Error
IBESR <30:29>  = IAMux EC <1:0> (Byte in Error)
```

PROBABLE CAUSE:

Modules	Probability	Components
L0206/IDP	High	RAMs (see Table 1)
L0206/IDP	Medium	IAD, DPPA
L0219/EBE	Low	WBus Parity Generator
L0219/EBE	Very low	IBESR Register

Table 1 IBox GPR RAM Callout

IAMux EC <1:0>	Data Bit	Byte	Slice	RAM	IAD MCA
0	0-3	0	0	E16 E20	E1
0	4-7	0	1	E19 E20	E2
1	8-11	1	2	E29 E24	E1
1	12-15	1	3	E32 E24	E2
2	16-19	2	4	E65 E58	E5
2	20-23	2	5	E69 E58	E7
3	24-27	3	6	E74 E71	E8
3	28-31	3	7	E75 E71	E8

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 10, 18, 1E, or 1F

CSL ACTION: Standard (See Introduction)

VMS ACTION: Standard (See Introduction)

I-05 IBox RLog Parity Error

OVERVIEW: RLog parity is checked during an RLog unwind operation. The RLog is written during autoincrement, autodecrement, autoincrement deferred, EXC only, and for the first specifier of all instructions regardless of the addressing mode.

Data flows from the ICB Module RLog RAMs, to the RLog Latches, to the parity checkers, and to the IDP Module DPPB and SPA MCAs. The RLog Parity Error signal (RLog PE) goes from the ICB Module to the EBE Module where it is latched into IBESR.

ERROR SIGNATURE:

EBCS <13> = IBox Error
EBCS <03> = State Modified
IBESR <24> = RLog Parity Error

PROBABLE CAUSE:

Modules	Probability	Components
L0214/ICB	High	RAMs (E162, E163, E164)
L0214/ICB	Medium	RLog Latches, Generator, Checker
L0206/IDP	Low	DPPB, SPA
L0219/EBE	Very low	IBESR Register

EHM ACTION: The EHM set Machine State Modified EBCS <03>, and then processes the error in the normal manner. Standard (See Introduction)

EHM VECTOR: 1F

CSL ACTION: None

VMS ACTION: Standard (See Introduction)

I-06 IBox IBuffer Parity Error

OVERVIEW: IBuffer Parity is checked on the Opcode Byte 0, on the Specifier Byte 1, and on the bytes <6,4,3,2> when selected by the RMode Finder during optimization.

Opcode <B0>, or Specifier <B1> Parity is checked when the byte is valid. If a error is detected in IBuf bytes <B1:B0>, the setting of the buffer valid flags are inhibited, and Opcode Valid into the Opcode Buffer is negated.

RMode Parity Error detection is enabled only when a successful instruction optimization is performed. The byte selected by the RMode Finder is checked, and if a parity error is detected, the setting of the buffer valid flags are inhibited.

Data flows from the MBox MCD Module MDP MCAs, to the IBox IBD Module IBF MCAs, and then to the IOP MCAs for parity checking. The IBuf Parity error signal from the IOP MCA is fed through the DBS MCA, to the ICB Module, and then to the EBox EBE Module where it is latched in IBESR.

NOTE

If MBox Interrupt EBCS <14> is set, the failure is a result of the MBox sending bad data. Therefore this error should be ignored. Instead check for one of the following errors:

MSTAT2 <14> - ABus Bad Data Code

MSTAT1 <03> - Cache Data Parity Error

MDECC <22> - Bad Data Error

MDECC <20> - Data Double Bit Error

MDECC <19> - Data Address Parity Error

ERROR SIGNATURE:

EBCS <13> = IBox Error
IBESR <25> = IBuf Parity Error
EBCS <14> = "0" No MBox Interrupt

PROBABLE CAUSE:

Modules	Probability	Components
L0208/IBD	High	IBF
L0208/IBD	Medium	IOPB, IOPC
L0204/MCD	Low	MDP
L0219/EBE	Very Low	IBESR Register

MANUAL MACHINE CHECK STACK FRAME ANALYSIS

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 10, 18, 1E, or 1F

CSL ACTION: Standard (See Introduction)

VMS ACTION: Standard (See Introduction)

I-07 IBox IBMux Parity Error

OVERVIEW: IBMux odd parity is checked on Instruction Stream Operand Data from IBUF bytes <5:1>, and on Indirect Addresses from the IMD. The parity that is checked at the output of the IBMux Latch originates from the MD BUS.

Data flows from the MBox MCD Module MDP MCAs, to the IBox IBD Module IBF MCAs and DBS MCAs, then to the IDP Module IAD MCAs. The IBMux Parity Error signal (IBMux PE) goes from the IDP Module to the EBE Module where it is latched in IBESR.

NOTE

If MBox Interrupt EBCS <14> is set, the failure is a result of the MBox sending bad data. Therefore this error should be ignored. Instead check for one of the following errors:

MSTAT2 <14> - ABus Bad Data Code

MSTAT1 <03> - Cache Data Parity Error

MDECC <22> - Bad Data Error

MDECC <20> - Data Double Bit Error

MDECC <19> - Data Address Parity Error

ERROR SIGNATURE:

EBCS <13> = IBox Error
IBESR <26> = IBMux Parity Error
EBCS <14> = "0" No MBox Interrupt

PROBABLE CAUSE:

Modules	Probability	Components
L0208/IBD	High	IBF, DBS
L0206/IDP	Medium	IAD, DPPB
L0204/MCD	Low	MDP
L0219/EBE	Very Low	IBESR Register

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 10, 18, 1E, or 1F

CSL ACTION: Standard (See Introduction)

VMS ACTION: Standard (See Introduction)

F-01 FBox Self Test Error

OVERVIEW: The FBox runs a self test micro-diagnostic when it is not busy executing FBox instructions. (I.e., the CPU is executing non-floating point accelerator instructions (e.g., MOV, TSTB).

If a Self Test Error is detected, the Self Test Error flag (FBXERR <18>) will set. When the Self Test Error flag sets FBXERR <02> will indicate which of the two FBox Modules failed the Self Test. If FBXERR <02> is set then the FBM Module failed. Otherwise the FBA Module failed.

When an FBox detects an error it generates a signal called FBox Problem. When the EBox requests the result of an FBox operation it will detect FBox Problem. FBox Problem goes from the FBA module (FBR MCA) to the EBD module, where it causes a EBox micro-trap to location 2, the entry vector for the FBox Service routine.

The FBox Service routine will determine whether FBox Problem was set because of a hardware error or because of some other FBox exception condition (i.e., Divide by Zero). If the FBox detected an error the FBox Service Routine will set FBox Service Request (EHSR <28>) and call the EHM.

ERROR SIGNATURE (FBA Module):

EHMSTS <28> = FBox Service Request (Note 1)
FBXERR <00> = FBox Problem
FBXERR <18> = Self Test Error
FBXERR <02> = FBA Module

PROBABLE CAUSE (FBA Module):

Module	Probability
L0212/FBA	High
L0213/FBM	Low

ERROR SIGNATURE (FBM Module):

EHMSTS <28> = FBox Service Request (Note 1)
FBXERR <00> = FBox Problem
FBXERR <18> = Self Test Error
FBXERR <02> = FBM Module

PROBABLE CAUSE (FBM Module):

Module	Probability
L0213/FBM	High
L0212/FBA	Low

Note

1. FBXERR is only valid if EHMSTS <28> is set

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 2

CSL ACTION: None

VMS ACTION: Standard (See Introduction)

F-02 FBox GPR Parity Error

OVERVIEW: The Scratchpad RAMs in the FBox use odd parity. They can be written from the WBus (SOP MCA) or from the Fraction Adder (FAD MCA). The parity bits from the WBus and Fraction Adder are selected for input to the RAMs on the RRC MCA. The output parity from the RAMs is checked on the FBR MCA. The GPR PE error signal is latched in FBox Register 1 on the FBR MCA.

When an FBox detects an error it generates a signal called FBox Problem. When the EBox requests the result of an FBox operation it will detect FBox Problem. FBox Problem goes from the FBA module (FBR MCA) to the EBD module, where it causes a EBox micro-trap to location 2, the entry vector for the FBox Service routine.

The FBox Service routine will determine whether FBox Problem was set because of a hardware error or because of some other FBox exception condition (i.e., Divide by Zero). If the FBox detected an error the FBox Service Routine will set FBox Service Request (EHSR <28>) and call the EHM.

ERROR SIGNATURE:

EHMSTS <28> = FBox Service
FBXERR <00> = FBox Problem (Note 1)
FBXERR <17> = GPR Parity Error

Note

1. FBXERR is only valid if EHMSTS <28> is set

PROBABLE CAUSE:

Modules	Probability	Component
L0212/FBA	High	RAMs
L0212/FBA	Medium	RRC, FAD, SOP, FBR
L0219/EBE	Low	WBus Parity Generator

EHM ACTION: The EHM reloads the FBox Scratch Pads and then processes the error in the standard manner. (See Introduction)
EHM VECTOR: 2

CSL ACTION: None

VMS ACTION: Standard (See Introduction)

F-03 FBox FDRAM Parity Error

OVERVIEW: The FDRAM in the FBox consists of 8 bits <7:0>. The FDRAMs are located on the FBM module. Odd parity checking on FDRAM data is done on the MCB MCA. The FDRAM parity error signal from MCB MCA is latched in FBox Register 2 on the FBA module FBR MCA.

When an FBox detects an error it generates a signal called FBox Problem. When the EBox requests the result of an FBox operation it will detect FBox Problem. FBox Problem goes from the FBA module (FBR MCA) to the EBD module, where it causes a EBox micro-trap to location 2, the entry vector for the FBox Service routine.

The FBox Service routine will determine whether FBox Problem was set because of a hardware error or because of some other FBox exception condition (i.e., Divide by Zero). If the FBox detected an error the FBox Service Routine will set FBox Service Request (EHSR <28>) and call the EHM.

EHM begin building a Machine Check Stack Frame, determine that it was called to handle an FBox DRAM PE and set EBCS <29>. This in turn will cause a Console interrupt by asserting the CPU ERROR <2:0> lines from the EBE module. The Console will reload the FDRAM and return control to the EHM. The EHM will continue building the Stack Frame and then call the VMS Machine Check Handler.

ERROR SIGNATURE:

```
EHMSTS <28>     = FBox Service Request (Note 1)
FBXERR <00>     = FBox PROBLEM
FBXERR <19>     = FDRAM PE
CSES   <31>     = Correctable Error
                    0 = Error corrected
                    1 = Unable to correct error
CSES   <02:00>  = "4" FDRAM Parity Correction
```

Note

1. FBXERR is only valid if EHMSTS <28> is set

PROBABLE CAUSE:

Module	Probability	Component
L0213/FBM	High	RAMs (See Table 1)
L0213/FBM	Medium	MCB
L0212/FBA	Low	FBR

Table 1 DRAM Callout

Bit	Signal Name	RAM
0	FORK ADDR 0	E39
1	FORK ADDR 1	E38
2	FORK ADDR 2	E39
3	FORK ADDR 3	E38
4	FORK ADDR 4	E39
5	FORMAT 0	E38
6	FORMAT 1	E39
7	FDRAM PARITY	E38

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 2

CSL ACTION: None

VMS ACTION: Standard (See Introduction)

F-04 FBox FBA CS Parity Error

OVERVIEW: Control Store Parity in the FBox is checked by the MCA that actually uses the microcode bits. The partial parity outputs from these MCAs are routed through to the ACC MCA which makes the final check for odd parity. If a parity error exists the failing address is held on the MSQ MCA, so the console can perform an ECC correction process on the location. The FBA CS PE error signal from ACC MCA is sent to the FBA Module (FBR MCA) where it is latched in FBox Register 3.

When an FBox detects an error it generates a signal called FBox Problem. When the EBox requests the result of an FBox operation it will detect FBox Problem. FBox Problem goes from the FBA module (FBR MCA) to the EBD module, where it causes a EBox micro-trap to location 2, the entry vector for the FBox Service routine.

The FBox Service routine will determine whether FBox Problem was set because of a hardware error or because of some other FBox exception condition (i.e., Divide by Zero). If the FBox detected an error the FBox Service Routine will set FBox Service Request (EHSR <28>) and call the EHM.

The EHM will begin building a Machine Check Stack Frame, determine that it was called to handle an FBA CS PE and set EBCS <30>. This in turn will cause a Console interrupt by asserting the CPU ERROR <2:0> from the EBE Module. The Console will correct the parity error and return control to the EHM. The EHM will continue building the Stack Frame and then call the VMS Machine Check Handler.

ERROR SIGNATURE:

```
EHMSTS <28>    = FBox Service Request
FBXERR <20>    = FBA CS Parity Error (Note 1)
CSES   <31>    = Correctable Error
                  0 = Error corrected
                  1 = Unable to correct error
CSES <28:16> = Control Store Address
CSES <15:08> = Syndrome
CSES <02:00> = "5" (FBA CS Parity Error)
```

Note

1. FBXERR is only valid if EHMSTS <28> is set

2. In all cases where ECC correction fails the Console will re-initialize the CPU and notify VMS.

PROBABLE CAUSE:

Module	Probability	Components
L0212/FBA	HIGH	RAMs (See Table 1)
L0212/FBA	LOW	ACB, ACC, ACL, ALN,FBR, MSQ, RRC

Table 1 FBox (FBA) Control Store RAM Callout

Syndrome	Phy	ULD	Signal Name	RAM	MCA
1	00	17	FA19 UROTK 0	E17	ACB E28
2	01	21	FA22 UAUXK 1	E68	ALN E14
3	02	31	FA21 UFADK 1	E68	ACL E63
4	03	23	FA18 UFARAK 0	E74	RRC E17
5	04	38	FA21 UEALU 2	E48	FBR E7
6	05	04	FA23 UJUMP 4	E23	MSQ E37
7	06	40	FA20 UEALU 4	E57	FBR E7
8	07	15	FA22 UHMX 2	E65	ALN E14
9	08	26	FA19 USOPK 1	E65	ACC E3
A	09	09	FA20 UBEN 0	E48	MSQ E37
B	10	07	FA23 UJUMP 7	E49	MSQ E37
C	11	14	FA22 UHMX 1	E57	ALN E14
D	12	33	FA22 UFADROTK 0	E69	ACL E63
E	13	32	FA21 UFADSIG	E69	ACL E63
F	14	22	FA22 USIGNIF	E66	ACB E28
10	15	13	FA22 UHMX 0	E66	ALN E14
11	16	00	FA23 UJUMP 0	E25	MSQ E37
12	17	36	FA21 UEALU 0	E12	FBR E7
13	18	05	FA23 UJUMP 5	E25	MSQ E37
14	19	16	FA19 UKHMX	E60	ACB E28
15	20	34	FA21 UFADROTK 1	E58	ACL E63
16	21	24	FA18 UFARAK 1	E58	RRC E17
17	22	47	FA18 UWRT SPAD	E12	RRC E17
18	23	18	FA18 UROTK 1	E60	ACB E28
19	24	37	FA21 UEALU 1	E54	FBR E7
1A	25	29	FA19 UBSIDE 1	E18	ACB E28
1B	26	20	FA22 UAUXK 0	E56	ALN E14
1C	27	30	FA21 UFADK 0	E56	ACL E63
1D	28	44	FA18 USEL RA 0	E10	RRC E17
1E	29	46	FA18 USEL RA 2	E10	RRC E17
1F	30	28	FA19 UBSIDE 0	E18	ACB E28
20	31	39	FA20 UEALU 3	E54	FBR E7
21	32	10	FA20 UBEN 1	E49	MSQ E37
22	33	35	FA21 UFADROTK 2	E49	ACL E63
23	34	06	FA23 UJUMP 6	E20	MSQ E37
24	35	01	FA23 UJUMP 1	E20	MSQ E37
25	36	45	FA18 USEL RA 1	E13	RRC E17
26	37	41	FA19 UDW PARITY	E9	FBR E7
27	38	02	FA23 UJUMP 2	E30	MSQ E37
28	39	42	FA19 UDRTY 0	E1	ACC E3
29	40	27	FA20 USHFTX	E8	ACC E3
2A	41	11	FA20 UBEN 2	E1	MSQ E37
2B	42	12	FA20 UBEN 3	E2	MSQ E37

Table 1 FBox (FBA) Control Store RAM Callout (cont.)

Syndrome	Phy	ULD	Signal Name	RAM	MCA
2C	43	08	FA22 UJUMP 8	E13	MSQ E37
2D	44	03	FA23 UJUMP 3	E30	MSQ E37
2E	45	19	FA18 UROTK 2	E9	ACB E28
2F	46	25	FA20 USOPK 0	E8	ACC E3
30	47	43	FA19 UDRTY 1	E2	ACC E3

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 2

CSL ACTION: None

VMS ACTION: Standard (See Introduction)

F-05 FBox FBM Control Store Parity Error

OVERVIEW: Control Store Parity in the FBox is checked by the MCA that actually uses the micro-code bits. The partial parity outputs from these MCAs are fed through to the MPZ MCA which makes the final check for odd parity. If a parity error exists the failing address is held on the MSQ MCA, so the console can perform an ECC correction process on the location. The FBM CS PE error signal from MPZ MCA is sent to the FBA Module FBR MCA where it is latched in FBox Register 3.

When an FBox detects an error it generates a signal called FBox Problem. When the EBox requests the result of an FBox operation it will detect FBox Problem. FBox Problem goes from the FBA module (FBR MCA) to the EBD module, where it causes a EBox micro-trap to location 2, the entry vector for the FBox Service routine.

The FBox Service routine will determine whether FBox Problem was set because of a hardware error or because of some other FBox exception condition (i.e., Divide by Zero). If the FBox detected an error the FBox Service Routine will set FBox Service Request (EHSR <28>) and call the EHM.

The EHM will begin building a Machine Check Stack Frame, determine that it was called to handle an FBM CS PE and set EBCS <30>. This in turn will cause a Console interrupt by asserting the CPU ERROR <2:0> from the EBE Module. The Console will correct the parity error and return control to the EHM. The EHM will continue building the Stack Frame and then call the VMS Machine Check Handler.

ERROR SIGNATURE:

```
EHMSTS <28>    = FBox Service Request
FBXERR <00>    = FBox Problem (Note 1)
FBXERR <21>    = FBM Control Store Parity Error
CSES   <31>    = Correctable Error
                   0 = Error corrected
                   1 = Unable to correct error (Note 2)
CSES <28:16> = Control Store Address
CSES <15:08> = Syndrome
CSES <02:00> = "6" FBM Control Store Parity Error
```

Note

1. FBXERR is only valid if EHMSTS <28> is set

2. In all cases where ECC correction fails the Console will re-initialize the CPU and notify VMS.

PROBABLE CAUSE:

Module	Probability	Components
L0213/FBM	High	RAMs (See Table 1)
L0213/FBM	Low	MAX, MCL, MPR, MPZ, MSQ

Table 1 FBox (FBM) Control Store RAM Callout (12-Mar-85)

Syndrome	Phy	ULD	Signal Name	RAM	MCA
1	00	20	FM15 MPLR SEL 2	E13	MPR E9
2	01	25	FM15 CRY CTL 0	E18	MCL E42
3	02	15	FM14 MHLD SEL 2	E18	MPR E9
4	03	16	FM14 MHLD SEL 3	E16	MPR E9
5	04	34	FM14 MSMX CTL 1	E15	MAX E63
6	05	17	FM14 MCAND SEL	E14	MPR E9
7	06	14	FM14 MHLD SEL 1	E15	MPR E9
8	07	13	FM14 MHLD SEL 0	E16	MPR E9
9	08	22	FM15 QBMUX SEL	E14	MPR E9
A	09	18	FM14 MPLR ENAB	E13	MPR E9
B	10	24	FM15 LDACC 1	E22	MPZ E20
C	11	23	FM15 LDACC 0	E22	MPZ E20
D	12	21	FM14 COUNTER CLR	E12	MPR E9
E	13	19	FM15 MPLR SEL 1	E12	MPR E9
F	14	26	FM15 CRY CTL 1	E26	MCL E42
10	15	27	FM15 CRY CTL 2	E28	MCL E42
11	16	30	FM16 MRMX SEL 0	E59	MAX,MCL,E63,E42
12	17	31	FM16 MRMX SEL 1	E48	MAX,MCL,E63,E42
13	18	32	FM16 MRMX SEL 2	E48	MAX,MCL,E63,E42
14	19	35	FM16 LDFRMA	E28	MAX E63
15	20	33	FM16 MSMX CTL 0	E59	MAX E63
16	21	39	FM16 PARITY	E26	MPZ E20
17	22	28	FM16 EXAC LD 0	E23	MAX E63
18	23	29	FM16 EXAC LD 1	E23	MAX E63
19	24	10	FM17 BEN 1	E50	MSQ E37
1A	25	01	FM18 JUMP 1	E57	MSQ E37
1B	26	03	FM17 JUMP 3	E57	MSQ E37
1C	27	12	FM17 BEN 3	E44	MSQ E37
1D	28	00	FM17 JUMP 0	E53	MSQ E37
1E	29	36	FM17 MUL SYNC	E49	MPZ E20
1F	30	09	FM17 BEN 0	E44	MSQ E37
20	31	11	FM17 BEN 2	E49	MSQ E37
21	32	04	FM18 JUMP 4	E53	MSQ E37
22	33	05	FM18 JUMP 5	E50	MSQ E37
23	34	02	FM17 JUMP 2	E58	MSQ E37
24	35	08	FB18 JUMP 8	E58	MSQ E37
25	36	07	FM18 JUMP 7	E51	MSQ E37
26	37	38	FM18 SPARE 1	E40	MPZ E20
27	38	06	FM18 JUMP 6	E51	MSQ E37

```
28      39      37      FM18 SPARE 0        E40     MPZ  E20
```

EHM ACTION: Standard (See Introduction)
EHM VECTOR: 2

CSL ACTION: None

VMS ACTION: Standard (See Introduction)

CHAPTER 3

SBIA MANUAL STACK FRAME ANALYSIS

OVERVIEW

When the VMS Error Handler is called to process either an SBI Alert, an SBI Fault, or an SBI Error conditions it will generate an SBIA Error Record (Entry Code 13). The VMS Error Handler will also generate an SBIA Error Record for certain MBox Fatal Error Machine Checks. See the VMS MCHK Flow Chart (2 of 5) for the specific conditions under which this will happen.

The VMS Error Handler will appended the SBIA Error record to the System Event File ERRLOG.SYS. Thus, using either ANALYZE/ERROR or Spear, the record can be translated (See Chapter 5, Example 4) and analyzed.

The purpose of this Chapter is to help you analyze and identify the most probable cause of SBIA Error Entries. To do this you will need to translate the entry and extract three registers:

1. ERRSUM - Refered to as IOA ES in the Translated Entry
2. SBIERR - Refered to as SBIA ER in the Translated Entry
3. SBISTS - Refered to as SBIA FS in the Translated Entry

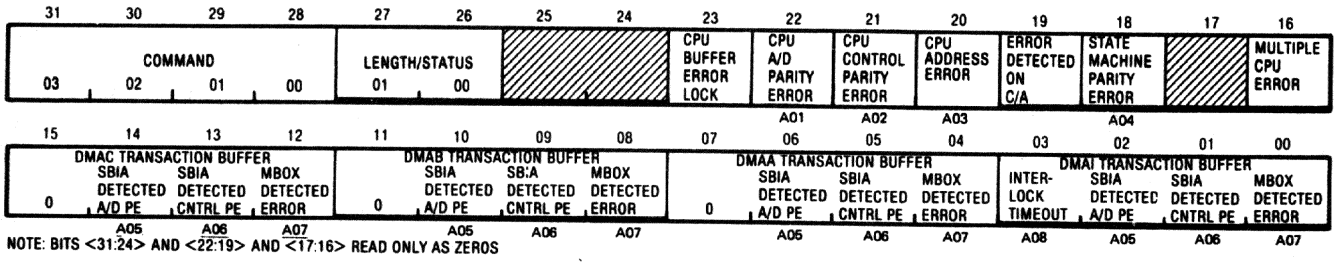
Match the contents each register with the corresponding registers shown below. Underneath the Error Status bit in the registers below there is a alphanumeric index (e.g., A-08). Use the index and Table 1 to identify the page in this Chapter that describes an error scenario that would produce a error record similar to the one you are analyzing.

Review the senario. It describes the most common conditions under which that type of error will occur. It also suggests the most probable cause of the error.

ERRSUM (IOA ES)

ERRSUM — **ERROR SUMMARY REGISTER**

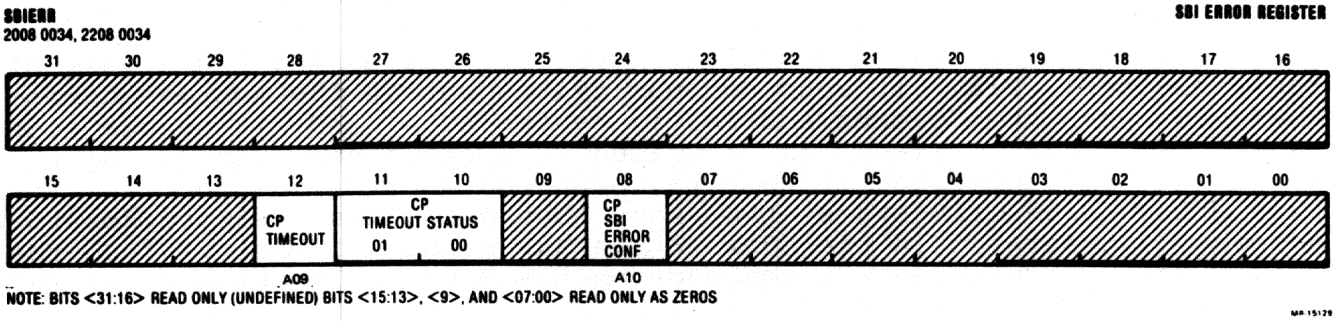
2008 0008, 2208 0008



NOTE: BITS <31:24> AND <22:19> AND <17:16> READ ONLY AS ZEROS

MR-15124

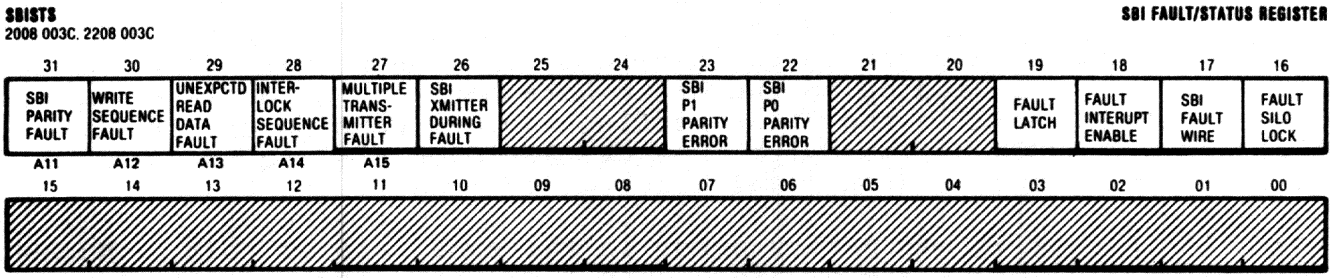
SBIERR (SBIA ER)



NOTE: BITS <31:16> READ ONLY (UNDEFINED) BITS <15:13>, <9>, AND <07:00> READ ONLY AS ZEROS

MR-15129

SBISTS (SBIA FS)



NOTE: BITS <31:20> AND <17:00> READ ONLY

MR-15131

Table 1 Index of SBIA and SBI Error Scneraios

A-01 SBIA Detected ABus Address/Data on CPU Reference

OVERVIEW: When the MBox passes a CPU Read Command/Address or a CPU Write Command/Address and Write Data over the ABus to be loaded into the CPU Buffer portion of the DC022 Register File on the SBA Module of the selected SBIA, it send odd parity to protect the Address during the cycle when the Command/Address is on the ABus and it sends odd parity (if the Command is a Write) to protect the Data during the following cycle when the Write Data is on the ABus. The Address and the Write Data timeshare the same ABus A/D lines. Parity for the Address is generated in the MBox on the MAP module. Parity for the Write Data is produced in the MBox on the MCD Module which takes the four original byte parity bits which had arrived from the EBox via the WBus accompanying the CPU Write Data and collapses them into one longword parity bit.

The SBIA first unloads the Address and parity bit from the DC022's onto the File Info Bus from where they are loaded into the File Data Latch on the SBS Module for the check (the Address is then passed on to the Command/Address Latch). It then unloads the Write Data (if any) and performs another check in exactly the same way (the Write Data is then passed on to the Write Data Latch). A parity error in either case results in the SBIA aborting the CPU Command and returning ABus CPU BUF ERROR H along with ABus CPU BUF DONE H to the MBox which, in turn, initiates an MBox Fatal Error micro trap in the EBox.

The SBIA latches either parity error in the same status bit, ERRSUM <22>. The two errors are distinguished from each other by ERRSUM <19> which sets only for errors detected on the Command/Address. This distinction must be made to determine which MBox source (MAP or MCD) produced the data/parity. It also latches the ABus Address in the SBI Timeout Address Register.

Note that for Write Data Parity Errors the CPU may have originated an error which propagated through the MBox, latching MBox status to indicate a CPU Write Parity Error in MSTAT1 <7:4>. If so, frozen MBox status will be unable to record MSTAT2 <2>, CP IO BUF ERR, although an MBox Fatal Error micro trap will still occur in the EBox, latching EBCS <15>, MBox FE. MSTAT2 <7>, Multiple Error, will also set, and MSTAT2 <20:16> will indicate the I/O Adapter.

ERROR SIGNATURE:

```
SBIA ERRSUM <22>    = CPU A/D PTY ERR
SBIA ERRSUM <19>    = ERR on CPU C/A (Note 1)
SBIA ERRSUM <23>    = CPU BUF Error Lock (Note 2)
SBIA TOADR <27:00> = ABus Address (A longword address)
MSTAT2 <2>          =  CP IO BUF ERR
EBCS <15>           =  MBox FE
```

Notes

1. If set the A/D PE was on the Address, not the Write Data

2. Locks ERRSUM <31:26> and the SBIA TOADR Register

NOTE

MEAR, MSTAT2 <20:16> (PAMM Code), and MSTAT1 <17:16> (Selected Adapter) obtained by EHM are not valid. However, before logging the error VMS overwrites MSTAT2 <17:16> with the correct Adapter code.

PROBABLE CAUSE:

Module	Probability
L0205/MAP	High (if ERRSUM <19> is set)
L0204/MCD	High (if ERRSUM <19> is reset)
L0202/SBS	High
L0203/SBA	High
ABus/Terminator	Low

EHM ACTION: EHM, entered as the result of an MBox Fatal Error microtrap, rolls back the instructions, builds a stack frame, and vectors to the Machine Check Handler via SCBB+4. See EHM flows for more detail.

VMS ACTION: After appending the SBIA ERRSUM Register to the Machine Check Stack Frame, setting EHMSTS <6> to note that it has done so, and logging the error, VMS checks the trapped instruction against a table of instructions capable of performing multiple I/O reads. If it gets a match, because some I/O Registers can be read only once (hence the instruction is not safe to retry), or if the instruction performs a read/modify/write and the error occurs on the write, VMS will abort the current image. Otherwise, because the failed reference was aborted by the SBIA before SBI or SBIA internal space could be affected, it REI's. Note that, if the processor mode at the time of the error was Kernal or Exec as it currently always should be, Image Abort becomes System Fatal Bugcheck.

A-02 SBIA Detected ABus Control PE On CPU Reference

OVERVIEW: When the MBox passes a CPU Read Command/Address or a CPU Write Command/Address and Write Data over the ABus to be loaded into the CPU Buffer portion of the DC022 Register File on the SBA Module of the selected SBIA, it send odd Control Parity to protect the Command/Length during the cycle when the Command/Address is on the ABus and it sends odd Control Parity (if the Command is a Write) to protect the Mask/Status sent during the following cycle when the Write Data is on the ABus. The Command/Length and the Mask/Status timeshare the same ABus Control lines. Control Parity is generated in the MBox by the ABS MCA on the MCC module.

The SBIA first unloads the Command/Length and parity bit from the DC022's onto the File Info Bus from where they are loaded into the File Data Latch on the SBS Module for the check (the Command/Length is then passed on to the Command/Address Latch). It then unloads the Mask/Status (if any) and performs another check in exactly the same way (the Mask/Status is then passed on to the Write Data Latch). A parity error in either case results in the SBIA aborting the CPU Command and returning ABus CPU BUF ERROR H along with ABus CPU BUF DONE H to the MBox which initiates an MBox Fatal Error micro trap in the EBox.

The SBIA latches either parity error in the same status bit, ERRSUM <21>. The two errors are distinguished from each other by ERRSUM <19> which sets only for errors detected on the Command/Address. It also latches the ABus Command and Length in ERRSUM <31:26>.qq

ERROR SIGNATURE:

SBIA ERRSUM <21>	=	CPU CNTRL PTY ERR
SBIA ERRSUM <23>	=	CPU BUF Error Lock (Note 1)
SBIA ERRSUM <31:28>	=	ABus Command Code
SBIA ERRSUM <27:26>	=	ABus Command/Length
SBIA ERRSUM <19>	=	ERR ON CPU C/A (Note 2)
MSTAT2 <02>	=	CP IO BUF ERR
EBCS <15>	=	MBox FE

Notes

1. Locks ERRSUM <31:26> and the SBIA SBI TOADR Register
2. If set the CNTRL PE was on the Command/Length, not the Mask/Status.

NOTE

MEAR, MSTAT2 <20:16> (PAMM Code), and MSTAT1 <17:16> (Selected Adapter) obtained by EHM are not valid. However, before logging the error VMS overwrites MSTAT2 <17:16> with the correct Adapter Code.

PROBABLE CAUSE:

Module	Probability
L0220/L0230/MCC	High
L0202/SBS	High
L0203/SBA	High
ABus/Terminator	Low

EHM ACTION: EHM, entered as the result of an MBox Fatal Error microtrap, rolls back the instructions, builds a stack frame, and vectors to the Machine Check Handler via SCBB+4. See EHM flows for more detail.

VMS ACTION: After appending the SBIA ERRSUM Register to the Machine Check Stack Frame, setting EHMSTS <6> to note that it has done so, and logging the error, VMS checks the trapped instruction against a table of instructions capable of performing multiple I/O Reads. If it gets a match, because some I/O Registers can be Read only once (hence the instruction is not safe to retry), or if the instruction performs a Read/modify/write and the error occurs on the write, VMS will abort the current image. Otherwise, because the failed reference was aborted by the SBIA before SBI or SBIA internal space could be affected, it REI's. Note that, if the processor mode at the time of the error was Kernal or Exec as it currently always should be, Image Abort becomes System Fatal Bugcheck.

A-03 SBIA Detected Address Error on CPU Reference

OVERVIEW: When a CPU Read or Write request arrives at the MBox, the physical address being referenced is used to address the PAMM (Physical Address Memory Map RAMs located on the MAP Module). If the resulting PAMM Code selects an ABus Adapter, the MBox takes the CPU Command/Address and any Write Data and sends them over the ABus to be loaded into the CPU Buffer portion of the DC022 Register File on the SBA Module of the selected SBIA.

The SBIA unloads the Command/Address from the DC022's onto the File Info Bus from where it is loaded into the File Data Latch on the SBS Module for a parity check. The Command/Address is then passed on to the Command/Address Latch where the address is decoded. If the address is an Adapter Local Address (not for the SBI) but is referencing a nonexistent adapter register, then the SBIA aborts the CPU Command and returns ABus CPU BUF ERROR H along with ABus CPU BUF DONE H to the MBox which initiates an MBox Fatal Error micro trap in the EBox.

Note that the SBIA will also detect an Address Error on CPU initiated SBI transactions when Control/Status <30>, Enable SBI Cycles Out, is reset. Control/Status <30> is normally set by the Console during boot.

The SBIA latches Address Error in ERRSUM <20>. It also latches the ABus Address in its TOADR Register.

ERROR SIGNATURE:

SBIA ERRSUM <20>	=	Adrs Err
SBIA ERRSUM <23>	=	CPU BUF Error Lock (Note 1)
SBIA ERRSUM <19>	=	ERR ON CPU C/A
SBIA TOADR <27:00>	=	the ABus Address (A Longword Address)
MSTAT2 <02>	=	CP IO BUF ERR
EBCS <15>	=	MBox FE

Note

1. Locks ERRSUM <31:26> and the SBIA TOADR Register

NOTE

MEAR, MSTAT2 <20:16> (PAMM Code), and MSTAT1 <17:16> (Selected Adapter) obtained by EHM are not valid. However, before logging the error VMS overwrites MSTAT2 <17:16> with the correct Adapter Code.

PROBABLE CAUSE:

Module	Probability
Software	High
L0202/SBS	Medium
L0205/MAP	Low
L0203/SBA	Low

EHM ACTION: The EHM, entered as the result of an MBox Fatal Error microtrap, rolls back the instructions, builds a stack frame, and vectors to the Machine Check Handler via SCBB+4. See EHM flows for more detail.

VMS ACTION: After setting EHMSTS <05> to note that a full SBIA log is to follow, VMS logs the error and aborts the current image. Note that, if the processor mode at the time of the error was Kernal or Exec as it currently always should be, Image Abort becomes System Fatal Bugcheck.

A-04 SBIA Detected State Machine PE

OVERVIEW: A parity protected State Machine on the SBA module oversees all CP to I/O references which transmit to the SBI. It steps through the "states" of the SBI protocol: ARB for the SBI (and start the SBI Timeout Counter), transmit Command (and perhaps Write Data), await ACK(s), await Read Data (and restart Timeout Counter), ACK Read Data, and return to Idle. In addition to the usual SBI Reads and Writes, it is invoked for Interrupt Summary Read and Quad Clear, with all four functions stepping through some different "states".

If a State Machine parity error is detected during an active CP to I/O reference, the reference is aborted (an SBI Fault may result), and a CP IO BUFF ERR is reported to the MBox. But a parity error can also occur just as the reference finishes or while the State Machine is in the Idle State. To insure that these errors are also reported an SBIA interrupt is requested. (This may occur in parallel with the MBox Fatal Error microtrap request which was caused by the CP IO BUFF ERR).

ERROR SIGNATURE:

SBIA ERRSUM <18>	=	STATE MACH PTY ERR
MSTAT2 <02>	=	CP IO BUF ERR (Note 1)
EBCS <15>	=	MBox FE (Note 1)

Note

1. If the error is detected during a CP to I/O reference

NOTE

MEAR, MSTAT2 <20:16> (PAMM Code), and MSTAT1 <17:16> (Selected Adapter) obtained by EHM are not valid. However, before logging the error VMS overwrites MSTAT2 <17:16> with the correct adapter code.

PROBABLE CAUSE:

Module	Probability
L0203/SBA	High
L0202/SBS	Low

EHM ACTION: Unless the error occurs during CP to I/O Reference (which causes the usual Machine Check Stack Frame to be build because of the CP IO BUFF ERROR) the EHM is not entered. Instead, the error is reported by an adapter IPL 1C interrupt.

VMS ACTION: For Machine Checks, after appending the SBIA Error Summary Register to the Machine Check stack frame and setting EHMSTS <6> to note that it has done so, VMS logs the error and aborts the current image. Note that Image Abort, if the processor mode at the time of the error was Kernal or Exec as it currently always should be, becomes System Fatal Bugcheck. For errors reported by adapter interrupt, after making a full log of SBIA status Registers, VMS takes a System Fatal Bugcheck.

A-05 SBIA Detected ABus Data PE on DMA Read Data

OVERVIEW: The MBox provides odd parity for each longword of DMA Read Data that it loads over the ABus into the DC022 Register File on the SBA module. For Cache Reads this longword parity is produced by XORing the four Cache byte parity bits. For Array Reads longword parity is generated by the ECC MCA. All such parity production occurs on the MCD module. For Cache Data Parity Errors and ECC Uncorrectable Array Errors the MBox provides an all zero's longword and a zero parity bit. A signal from the MCC module controls driving the longword and parity onto the ABus.

The SBIA unloads the Control Field and Parity Bit from the DC022's onto the File Info Bus from there the Control Field and Parity Bit are loaded into the File Data Latch on the SBS module for the check. When a parity error is detected, the SBIA generates good parity and sends the bad data with good parity along with Read Data Substitute (RDS) to the reading Nexus. The SBIA will post an interrupt requests and thus notify the CPU of the parity error.

Depending on which one of the four DMA buffers in the DC022 Register File is being used, the parity error is latched as one of four status bits in the SBIA Error Summary Register. This latching locks the entire error field associated with the active buffer in the Error Summary Register. It also locks the DMAx Command/Address and DMAx ID Registers (where x corresponds to the active buffer). This preserves the SBI Command/Address associated with the error along with the SBI ID of the Nexus which issued that Command/Address.

ERROR SIGNATURE:

SBIA ERRSUM <14,10,6,2>	=	SBIA Detect A/D (Which bit is set depends on the DC022 Buffer being used.)

PROBABLE CAUSE:

Module	Probability
L0204/MCD	High
L0203/SBA	High
L0202/SBS	High
ABus/Terminator	Low
L0220/L0230/MCC	Low

EHM ACTION: Except for Cache Data Parity Errors and Uncorrectable Array ECC Errors (which cause a Machine Check) the EHM is not entered. Instead, the error is reported by an adapter IPL 1C interrupt.

VMS ACTION: VMS logs the error, clears adapter status, and continues. Retries are left to the Device Driver.

A-06 SBIA Detected ABus Control PE on DMA Read Data (Mask/Status)

OVERVIEW: In response to a DMA Read the MBox always sends an ABus Control Field consisting of an all zero's Status Sub-field, an all zero's Mask Sub-field, and Odd Parity along with each longword. The Control Field and the longword are loaded into the DC022 Register File on the SBA Module. Although the SBIA does not use the Control Field it does check the field for Odd Parity. The ABS MCA on the MCC module generates the Control Parity.

The SBIA unloads the Control Field and Parity Bit from the DC022's onto the File Info Bus from there the Control Field and Parity Bit are loaded into the File Data Latch on the SBS module for the check. When a parity error is detected, the SBIA generates good parity and sends the bad data with good parity along with Read Data Substitute (RDS) to the reading Nexus. The SBIA will post an interrupt requests and thus notify the CPU of the parity error.

Depending on which one of the four DMA buffers in the DC022 Register File is being used, the parity error is latched as one of four status bits in the SBIA Error Summary Register. This latching locks the entire error field associated with the active buffer in the Error Summary Register. It also locks the DMAx Command/Address and DMAx ID Registers (where x corresponds to the active buffer). This preserves the SBI Command/Address associated with the error along with the SBI ID of the Nexus which issued that Command/Address.

ERROR SIGNATURE:

SBIA ERRSUM <13,9,5,1>	=	SBIA Detect Cntrl (Which bit is set depends on the dc022 Buffer being used.)

PROBABLE CAUSE:

Module	Probability
L0220/L0230/MCC	High
L0203/SBA	High
L0202/SBS	High
ABus/Terminator	Low

EHM ACTION: The EHM is not entered. Instead, the error is reported by an adapter IPL 1C interrupt.

VMS ACTION: VMS logs the error, clears adapter status, and continues. Retries are left to the Device Driver.

A-07 SBIA Detected DMA Errors (Command/Lenght, Address and NXM)

OVERVIEW: The SBIA latches one of four status bits (which bit depends on the active DC022 Buffer) in its Error Summary Register and requests an interrupt whenever the MBox replies with both MCC DMA ERROR H and MCC DMA DONE [N] H in response to one of the SBIA's DMA requests. The MBox does this when it detects one of the following conditions.

1. An ABus Control Parity Error on the "DMA Command/Length"
2. An ABus Address/Data Parity Error on the "DMA Address"
3. A "NXM" on the DMA Address

The MBox aborts the bad DMA command, sets status dependent on the error type, and requests an IPL 1D interrupt.

The latched status bit in the SBIA locks the entire error field in the Error Summary Register associated with the active buffer. It also locks the DMAx Command/Address and DMAx ID Registers (where x corresponds to the active buffer). This preserves the SBI Command/Address which got the error along with the SBI ID of the Nexus which issued that Command/Address.

ERROR SIGNATURE:

MSTAT1 <18>	=	ABus C/A Cycle
MSTAT1 <20>	=	ABus Cntl PE (A Command/Length PE) (Note 1)
MSTAT1 <19>	=	ABus Address PE (Note 1)
MSTAT2 <3>	=	NXM (Note 1)
MSTAT1 <29:26>	=	"8" ABus Cycle or "4" ABus Array Write Cycle
SBIA ERRSUM <12,8,4,0>	=	MBox Detect (Note 2)

Notes

1. Three possible errors.
2. Which bit is set depends on the DC022 Buffer being used.

PROBABLE CAUSE:

Module	Probability
L0202/SBS	High
L0203/SBA	High
L0205/MAP	High (If Address Parity Error)
L0220/L0230/MCC	High (If Command/Length Parity Error)
ABus/Terminator	Low

EHM ACTION: The EHM which is entered as a result of the MBox interrupt, rolls back the instructions, builds a stack frame, and vectors to the Machine Check Handler via SCBB+4. See EHM flows for more detail.

VMS ACTION: VMS logs the error, clears adapter status (clearing the adapter IPL 1C interrupt), and takes a System Fatal Bugcheck.

A-08 SBIA DMA Interlock Timeout

OVERVIEW: A DMA SBI Interlocked Read directed to VAX8600/8650 internal memory and accepted by an SBIA must be followed by an (ACK'ed) SBI Interlocked Write within 512 SBI cycles. If this does not happen the SBIA will detect an Interlock Timeout, drop the ABus Lock Wire if it is driving it (i.e. if the Interlocked Read was accepted by the MBox), latch status, and request an interrupt.

The latched status bit locks the entire error field in the Error Summary Register associated with the DMAI buffer (i.e. Error Summary <3:0>). It also locks the DMAI Command/Address and DMAI ID Registers to preserve the SBI Command/Address which got the error along with the SBI ID of the Nexus which issued that Command/Address.

This timeout can occur, for example, if the MBox does not return MCC ABUS DMA DONE [N] H to the adapter signalling that it has completed the Interlock Read. The requesting Nexus will detect an SBI Read Data Timeout and won't issue the Interlock Write.

ERROR SIGNATURE:

SBIA ERRSUM <3> = DMA INTLK TMOUT

PROBABLE CAUSE:

Module	Probability
SBI Nexus	High
L0202/SBS	Medium
L0203/SBA	Low
L0220/L0230/MCC	Low

EHM ACTION: The EHM is not entered. Instead, the error is reported by IPL 1C adapter interrupt.

VMS ACTION: VMS logs the error and continues. Retries are left to the I/O Driver.

A-09 SBIA Detected SBI Timeout Error on CPU Reference

OVERVIEW: When a CPU Read or Write request arrives at the MBox, the physical address being referenced is used to address the PAMM (Physical Address Memory Map RAMs located on the MAP Module). If the resulting PAMM Code selects an ABus Adapter, the MBox takes the CPU Command/Address and any Write Data and sends them over the ABus to be loaded into the CPU Buffer portion of the DC022 Register File on the SBA Module of the selected SBIA.

The SBIA unloads the Command/Address from the DC022's onto the File Info Bus from where it is loaded into the File Data Latch on the SBS Module for a parity check. The Command/Address is then passed on to the Command/Address Latch where the Address is decoded. If the decoded Address is not a local address (directed to an SBIA Register) or is a local address which allows, when accessed, an SBI function such as Interrupt Summary Read or Quad Clear to be performed, the SBIA State Machine is activated to perform the SBI protocol.

The State Machine starts the 512 SBI cycle Timeout Counter and begins arbitrating for the SBI. If it cannot get it before the timer expires, it aborts the SBI function, latches status, and returns ABus CPU BUF ERROR H along with ABus CPU BUF DONE H to the MBox.

If it is able to get the bus (the SBIA is usually assigned a high TR Level (low priority) for these CPU requests), it passes the ABus Command/Address (after translation into SBI format) along with the Mask during the first transmit cycle and any Write Data during a second transmit cycle onto the SBI. (For Quad Clear the SBI Command/Address is derived from the ABus Write Data while the SBI Mask derives from the ABus Mask; there will also be a second Write Data transmit cycle. For Interrupt Summary Read the SBIA itself produces the SBI Command: forced 0's.)

The State Machine then waits for an SBI "ACK" (SBI CNF <1:0> = 01) from the addressed Nexus occurring two SBI cycles after each transmit cycle, signalling acceptance of the Command/Address And any Write Data. If it gets either a No Response (CNF <1:0> = 00) or a Busy (CNFf<1:0> = 10) Confirmation to any transmit, it then re-arbitrates for the SBI and transmits the whole thing all over again. If this occurs often enough for the Timer to expire (it is still running), the State Machine will abort the function, latch status, and return ABus CPU BUF ERROR H along with ABus CPU BUF DONE H to the MBox.

NOTE

Interrupt Summary Read does not require an "ACK"; instead the State Machine simply checks for good SBI parity during what would would normally be the "ACK" cycle, then passes the received information through a priority encoder into the DC022's and issues ABus CPU BUF DONE H. If the received parity is bad, the State Machine will attempt to retransmit the Interrupt Summary Read. If this occurs often enough for the Timer to expire, the State Machine will abort the function, latch status, and return ABus CPU BUF ERROR H along with ABus CPU BUF DONE H to the MBox.)

If the function being performed is a Read, then after the Command/Address has been ACK'ed the State Machine will restart the Timer and await the Read Data. If the Data does not arrive before the Timer expires, the State Machine will abort, latch status, and return ABus CPU BUF ERROR H along with ABus CPU BUF DONE H to the MBox, initiating an MBox Fatal Error micro trap in the EBox.

Note that if the CPU attempts to access a nonexistent SBI Nexus, the SBIA will detect this error. Therefore software can cause this error.

The SBIA latches an SBI Timeout in SBIERR <12> and the Timeout type in SBIERR <11:10>. It also latches the ABus Address (which, except in the case of Quad Clear, should indicate the SBI address referenced) in its TOADR Register.

ERROR SIGNATURE:

SBIA SBIERR <12>	=	CPU Timeout
SBIA SBIERR <11:10>	=	CPU Timeout Status (Note 1)
SBIA ERRSUM <23>	=	CPU BUF Error Lock (Note 2)
SBIA TOADR <27:00>	=	the ABus Address (A Longword Address)
MSTAT2 <02>	=	CP IO BUF ERR
EBCS <15>	=	MBox FE

Notes

1. 00 = Device No Response on Last Access (includes couldn't get bus)

2. 01= Device Busy on Last Access

3. 10= Waiting for Read Data

4. Locks ERRSUM <31:26>, SBIERR <12,8> and the SBIA TOADR Register

NOTE

MEAR, MSTAT2 <20:16> (PAMM Code), and MSTAT1 <17:16> (Selected Adapter) obtained by EHM are not valid. However, before logging the error VMS, overwrites MSTAT2 <17:16> with the correct Adapter Code.

PROBABLE CAUSE:

Module	Probability
Software	High
L0202/SBS	Medium
Nexus	Medium
L0203/SBA	Low
L0220/L0230/MCC	Low
CPU	Low

EHM ACTION: The EHM, entered as the result of an MBox Fatal Error microtrap, rolls back the instructions, builds a stack frame, and vectors to the Machine Check Handler via SCBB+4. See EHM flows for more detail.

VMS ACTION: After setting EHMSTS <05> to note that a full SBIA log is to follow, VMS logs the error and aborts the current Image. (However, if VMS can determine that the error occurred on a read to a BRRVR in a DW780, it will REI instead). Note that, if the processor mode at the time of the error was Kernal or Exec as it currently always should be, Image Abort becomes System Fatal Bugcheck.

A-10 SBIA Detected SBI Error Confirmation on CPU Reference

When a CPU Read or Write request directed beyond the VAX8600/8650 internal memory arrives at the MBox, the MBox assembles an ABus Command/Address and sends it along with any Write Data and the Write Data Mask over the ABus to be loaded into the CPU Buffer portion of the DC022 Register File on the SBA Module of the selected SBIA. The ABus Command is produced by the ABS MCA on the MCC module. The Write Data Mask is produced by the STAT MCA on the MCC Module from the context and address of the CPU request; the STAT MCA then passes this Mask to the ABS MCA for transmission to the ABus.

The SBIA unloads the Command/Address from the DC022's onto the File Info Bus from where it is loaded into the File Data Latch on the SBS Module for a parity check. The Command/Address is then passed on to the Command/Address Latch where the Address is decoded. The Write Data and Mask are similarly unloaded and parity checked and are then passed onto the Write Data Latch.

If the decoded Address is not a local address (directed to an SBIA Register) or is a local address which allows, when accessed, an SBI function such as Quad Clear to be performed, the SBIA State Machine is activated to pass the ABus Command/Address/Mask (after translating the Command into SBI format) during the first transmit cycle and any Write Data during the second transmit cycle onto the SBI. (For Quad Clear operations the SBI Command/Address is derived from the ABus Write Data while the SBI Mask, transmitted during both the C/A and the following Word cycle, derives from the ABus Mask; there is also a second Write Data transmit cycle.)

The State Machine then waits for an SBI "ACK" (SBI CNF <1:0> = 01) from the addressed Nexus occurring two SBI cycles after the Command/Address/Mask was transmitted, signalling acceptance of the Command. However, if after detecting good SBI parity, the receiving Nexus decodes the Command to be an illegal function, it will return an Error Confirmation (SBI CNF <1:0> = 11) instead of an "ACK". The State Machine will then abort the SBI function, latch status, and return ABus CPU BUF ERROR H along with ABus CPU BUF DONE H to the MBox, initiating an MBox Fatal Error micro trap in the EBox.

Note that SBI Nexuses return an Error Confirmation for other than a longword write to their internal Registers (i.e. the four bit SBI Mask transmitted with the Write Command was not all ones). A DW780 returns an Error Confirmation for any attempt at other than a word or byte access of UNIBUS Space. Therefore software can cause this error.

The SBIA latches an SBI Error Confirmation in SBIERR <08>. It also latches the ABus Address (which, except in the case of Quad Clear, should indicate the Nexus referenced) in its TOADR Register.

ERROR SIGNATURE:

SBIA SBIERR <08>	=	CP SBI Error Conf
SBIA ERRSUM <23>	=	CPU BUF Error Lock (Note 1)
SBIA TOADR	=	the ABus Address (A Longword Address)
MSTAT2 <02>	=	CP IO BUF ERR
EBCS <15>	=	MBox FE

Note

1. Locks ERRSUM <31:26>, SBIERR <12,08> and the SBIA TOADR Register.

NOTE

MEAR, MSTAT2 <20:16> (PAMM Code), and MSTAT1 <17:16> (Selected Adapter) obtained by EHM are not valid. However, before logging the error VMS, overwrites MSTAT2 <17:16> with the correct Adapter Code.

PROBABLE CAUSE:

Module	Probability
Software	High
L0202/SBS	Medium
Nexus	Medium
L0203/SBA	Low
L0220/L0230/MCC	Low
CPU	Low

EHM ACTION: The EHM, entered as the result of an MBox Fatal Error microtrap, rolls back the instructions, builds a stack frame, and vectors to the Machine Check Handler via SCBB+4. See EHM flows for more detail.

VMS ACTION: After setting EHMSTS <05> to note that a full SBIA log is to follow, VMS logs the error and aborts the current image. Note that, if the processor mode at the time of the error was Kernal or Exec as it currently always should be, Image Abort becomes System Fatal Bugcheck.

A-11 SBI Parity Fault

OVERVIEW: There is one SBI field, B <31:00>, protected with even parity by the SBI P1 parity bit. There is also a group of three SBI fields, TAG <3:0>, ID <4:0>, and M <3:0>, protected with even parity by the SBI P0 parity bit. All Nexuses including the SBIA latch these fields and parity check them once every SBI cycle. SBI null cycles in which no Nexus drives the bus have all zeroes, good parity on the bus, by default.

When the SBIA detects bad parity (or the SBI Fault Wire is being driven), it latches status in the SBISTS Register, stops updating the SBI Silo (which contains a history of the 16 SBI cycles preceding the Fault), drives the SBI Fault Wire to insure that the SBI Fault Status remains latched in either the SBISTS Register or in the CNFGR Registers located in the other Nexuses, and requests an interrupt.

If only one Nexus has its Parity Fault bit set, then the trouble is probably in a parity checker in that Nexus. If all Nexuses show a Parity Fault, then the trouble is probably in the Nexus which has Xmitr During Fault set (SBISTS <26> or CNFGR <26>). Also, the Silo (which will continue to record for a few more cycles after the Fault occurrs) should contain the TR Level of the Nexus which was transmitting on the SBI during the Fault.

ERROR SIGNATURE:

```
SBIA SBISTS <31> = Parity Fault
SBIA SBISTS <23> = P1 PTY ERR (Note 1)
SBIA SBISTS <22> = P0 PTY ERR (Note 1)
SBIA SBISTS <19> = Fault Latch (SBIA is driving the Fault Wire)
                   (Note 2)
SBIA SBISTS <17> = SBI Fault Signal (Live state of Fault Wire)
SBIA SBISTS <16> = Fault Silo Lock (A Fault has locked the Silo)
SBIA SBISTS <26> = Xmitr During Fault (Note 3)

Nexuses CNFGR <31> = Parity Fault
NEXUSES CNFGR <26> = Xmitr During Fault

SBIA SBI SILO <15:1> = Lowest order true bit is active TR Level
                       (Note 4)
"    "   "           = 15 more entries
```

Notes

1. Two different errors

2. Also indicates valid Fault status

3. Only set if the SBIA was transmitting

4. The SBI Silo also records the SBI Tag, ID, and Mask fields. SBI Silo <31> indicates that within 16 cycles after the Silo Fault Lock was cleared another fault occurred.

PROBABLE CAUSE:

Module	Probability
L0202/SBS	High (if transmitter)
SBI Nexus	High (if transmitter)
L0203/SBA	Low
SBI/Terminator	Low

EHM ACTION: The EHM is not entered. Instead, the error is reported by an IPL 1C adapter interrupt.

VMS ACTION: VMS logs the error and continues. Retries are left up to the I/O Driver.

A-12 SBI Write Sequence Fault

OVERVIEW: SBI protocol states that: an SBI Write Masked (Reg Bus <31:28> = 1101) or Interlock Write Mask (Reg Bus <31:28> = 0111) Command with a SBI Tag equal to Command/Address should be followed immediately by cycle that has the Tag equal to Write Data. Likewise, a cycle which indicates an SBI Extended Write Masked Command/Address should be followed immediately by two cycles of Write Data. This is SBI protocol.

When an SBIA accepts a DMA Masked Write Command that is not followed immediately by the right number of Write Data cycles, it latches a Write Sequence Fault in the SBISTS Register, stops updating the SBI Silo (which contains a history of the 16 SBI cycles preceding the Fault), drives the SBI Fault Wire to insure that the SBI Fault Status remains latched in either the SBISTS Register or in the CNFGR Registers located in the other Nexuses, and requests an interrupt. Other SBI Nexuses on the Bus will detect the same Write Sequence Fault on Masked Writes if the command is directed toward their (Nexus) address space.

When an SBIA Nexus (other than the SBIA) detects this Fault it latches Write Sequence Fault in its CNFGR Register and drives the SBI Fault Wire for one cycle. Upon detecting this, the SBIA latches its SBISTS Register, stops updating the Silo, continuously drives the SBI Fault Wire to preserve Fault status in all Nexuses, and requests an interrupt.

The Silo (which will continue to record for a few more cycles after the Fault occurs) should contain the TR Level of the Nexus which was transmitting on the SBI during the Fault.

Either the transmitter (see Xmitr During Fault Status) or the receiver (the Nexus which detected the Fault) should be the problem.

ERROR SIGNATURE:

```
SBIA SBISTS <30>  =  Wr Seq Fault (Note 1)
SBIA SBISTS <26>  =  Xmitr During Fault (Note 2)
SBIA SBISTS <19>  =  Fault Latch (SBIA is driving the Fault Wire)
                     (Note 3)
SBIA SBISTS <17>  =  SBI Fault Signal (Live state of Fault Wire)
SBIA SBISTS <16>  =  Fault Silo Lock (A Fault has locked the Silo)

NEXUSES CNFGR <30>  =  Wr Seq Fault
NEXUSES CNFGR <26>  =  Xmitr During Fault

SBIA SBI Silo <15:1>  =  Lowest order true bit is active TR Level.
                         (Note 4)
"    "   "            =  15 more entries.
```

Notes

1. If the SBIA detected the Fault.

2. If the SBIA caused the Fault in another Nexus.

3. Also indicates valid Fault status.

4. The SBI Silo also records the SBI Tag, ID, and Mask fields. SBI Silo <31> indicates that within 16 cycles after the Silo Fault Lock was cleared another Fault occurred.

PROBABLE CAUSE:

Module	Probability
L0202/SBS	High
SBI Nexus	High
L0203/SBA	Low

EHM ACTION: The EHM is not entered. Instead, the error is reported by adapter IPL 1C interrupt.

VMS ACTION: VMS logs the error and continues. Retries are left to the I/O Driver.

A-13 SBI Unexpected Read Data Fault

OVERVIEW: When a Nexus decodes a SBI Tag indicating Read Data and the SBI ID field matches it own ID (all with good SBI parity) but it is not expecting Read Data (it may be mistaken in that belief), it detects an Unexpected Read Data Fault.

When an SBIA detects this Fault (the ID that it compares for is the CPU's, 16) it latches Unexpected Read Data Fault in the SBISTS Register, stops updating the SBI Silo (which contains a history of the 16 SBI cycles preceding the Fault), drives the SBI Fault Wire to insure that the SBI Fault Status remains latched in either the SBISTS Register or in the CNFGR Registers located in the other Nexuses, and requests an interrupt. Other SBI Nexuses on the Bus will detect the same Unexpected Read Data Fault on SBI Read Data if the command is directed toward their (Nexus) address space.

When an SBI Nexus (other than the SBIA) detects this Fault it latches Unexpected Read Data Fault in its CNFGR Register and drives the SBI Fault Wire for one cycle. Upon detecting this the SBIA latches its SBISTS Register, stops updating the Silo, continuously drives the SBI Fault Wire to preserve Fault status in all Nexuses, and requests an interrupt.

The Silo (which will continue to record for a few more cycles after the Fault occurs) should contain the TR Level of the Nexus which was transmitting on the SBI during the Fault.

The Nexus which has its Xmitr During Fault bit set is probably the problem. However the detecting Nexus may have simply forgotten that it requested this Data. With luck, the Silo will have recorded that it had done so.

Note that if Read Data is returned after a Nexus is no longer expecting it, perhaps due to a Read Data Timeout, the Nexus will detect Unexpected Read Data Fault.

ERROR SIGNATURE:

```
SBIA SBISTS <29> = Unexp Rd Dat Fault (Note 1)
SBIA SBISTS <26> = Xmitr During Fault (Note 2)
SBIA SBISTS <19> = Fault Latch (SBIA is driving the Fault Wire)
                   (Note 3)
SBIA SBISTS <17> = SBI Fault Signal (Live state of Fault Wire)
SBIA SBISTS <16> = Fault Silo Lock (A Fault has locked the Silo)

Nexuses CNFGR <29> = Unexp Rd Dat Fault
Nexuses CNFGR <26> = Xmitr During Fault

SBIA SBI Silo <15:1> = Lowest order true bit is active TR Level.
                       (Note 4)
"    "   "           = 15 more entries
```

Notes

1. If the SBIA detected the Fault.

2. If the SBIA caused the Fault in another Nexus.

3. Also indicates valid Fault status.

4. The SBI Silo also records the SBI Tag, ID, and Mask fields. SBI Silo <31> indicates that within 16 cycles after the Silo Fault Lock was cleared another Fault occurred.

PROBABLE CAUSE:

Module	Probability
L0202/SBS	High (if the transmitter)
SBI Nexus	High (if the transmitter)
L0203/SBA	Low

EHM ACTION: The EHM is not entered. Instead, the error is reported by adapter IPL 1C interrupt.

VMS ACTION: VMS logs the error and continues. Retries are left to the I/O Driver.

A-14 SBI Interlock Sequence Fault

OVERVIEW: When the SBIA detects a valid SBI Interlock Write (ie. one with good parity which is directed toward internal memory) that was not proceeded by a valid SBI Interlock Read, it latches an Interlock Sequence Fault in the SBISTS Register, stops updating the SBI Silo (which contains a history of the 16 SBI cycles preceding the Fault), drives the SBI Fault Wire to insure that the SBI Fault Status remains latched in either the SBISTS Register or in the CNFGR Registers located in the other Nexuses, and requests an interrupt. Other SBI Nexuses which are capable of the Interlock function will detect the same Interlock Sequence Fault on Interlock Writes if the command is directed toward their (Nexus) address space.

When an SBI Nexus (other than an SBIA) detects this Fault it latches Interlock Sequence Fault in its CNFGR Register and drives the SBI Fault Wire for one cycle. Upon detecting this the SBIA latches its SBISTS Register, stops updating the Silo, continuously drives the SBI Fault Wire to preserve Fault status in all Nexuses, and requests an interrupt.

The Silo (which will continue to record for a few more cycles after the Fault occurrs) should contain the TR Level of the Nexus which was transmitting on the SBI during the Fault.

Either the transmitter (see Xmitr During Fault status) or the receiver (the Nexus which detected the Fault) should be the problem. Note, however, that the SBIA can also detect this Fault following a DMA Interlock Timeout. The SBIA starts its Interlock Timer as soon as it ACK's an Interlock Read on the SBI. If the MBox (which must arbitrate the SBIA request--and may "hang up" doing so) and the SBIA (which has its own overhead) return the Read Data to the requesting Nexus just in time to prevent the Nexus from detecting a Read Data Timeout, then the Nexus (after an unknown amount of SBI arbitration time) may get off an Interlock Write after the SBIA has already timed out waiting for it (and is no longer expecting it).

ERROR SIGNATURE:

```
SBIA SBISTS <28> = Intlk Seq Fault (Note 1)
SBIA SBISTS <26> = Xmitr During Fault (Note 2)
SBIA SBISTS <19> = Fault Latch (SBIA is driving the Fault Wire)
                   (Note 3)
SBIA SBISTS <17> = SBI Fault Signal (Live state of Fault Wire)
SBIA SBISTS <16> = Fault Silo Lock (A Fault has locked the Silo)

Nexus  CNFGR <28> = Intlk Seq Fault
Nexus  CNFGR <26> = Xmitr During Fault

SBIA SBI Silo <15:1> = Lowest order true bit is active TR Level.
                       (Note 4)
"    "   "           = 15 more entries
```

Notes

1. If the SBIA detected the Fault.

2. If the SBIA caused the Fault in another Nexus.

3. Also indicates valid Fault status.

4. The SBI Silo also records the SBI Tag, ID, and Mask fields. SBI Silo <31> indicates that within 16 cycles after the Silo Fault Lock was cleared another Fault occurred

PROBABLE CAUSE:

Module	Probability
L0202/SBS	High
SBI Nexus	High
L0203/SBA	Low

EHM ACTION: The EHM is not entered. Instead, the error is reported by adapter IPL 1C interrupt.

VMS ACTION: VMS logs the error and continues. Retries are left to the I/O Driver.

A-15 SBI Multiple Transmitter Fault

OVERVIEW: Each cycle that a Nexus transmits on the SBI, is read back by the Nexus, latched in its SBI ID field, and compared against the ID that it transmitted (its own ID except when it is transmitting Read Data. If the Nexus is transmitting Read Data the ID will be that of the receiving Nexus). A mismatch (which may simply be a mis-compare) is detected as a Multiple Transmitter Fault. This Fault may or may not be accompanied by a Parity Fault.

When an SBIA detects this Fault, it latches a Multiple Transmitter Fault in the SBISTS Register, stops updating the SBI Silo (which contains a history of the 16 SBI cycles preceding the Fault), drives the SBI Fault Wire to insure that the SBI Fault Status remains latched in either the SBISTS Register or in the CNFGR Registers located in the other Nexuses, and requests an interrupt. The other SBI Nexuses can detect the same Multiple Transmitter Fault during their transmit cycles.

When an SBI Nexus (other than the SBIA) detects this Fault it latches Multiple Transmitter Fault in its CNFGR Register and drives the SBI Fault Wire for one cycle. Upon detecting this the SBIA latches its SBISTS Register, stops updating the Silo, continuously drives the SBI Fault Wire to preserve Fault status in all Nexuses, and requests an interrupt.

The Silo (which will continue to record for a few more cycles after the Fault occurs) should contain the TR Level(s) of the Nexus(es) which was transmitting on the SBI during the Fault.

If more than one Nexus has its Xmitr During Fault bit set, then the one with the highest (i.e. lowest priority) TR level is the problem (assuming that the Silo reveals that the higher priority Nexus was entitled to be on the bus at the time of the Fault). If only one Nexus has its Xmitr During Fault bit set, then that Nexus is the problem.

ERROR SIGNATURE:

```
SBIA SBISTS <27> = Multi Xmit Fault (Note 1)
SBIA SBISTS <26> = Xmitr During Fault (Note 2)
SBIA SBISTS <19> = Fault Latch (SBIA is driving the Fault Wire)
                   (Note 3)
SBIA SBISTS <17> = SBI Fault Signal (Live state of Fault Wire)
SBIA SBISTS <16> = Fault Silo Lock (A Fault has locked the Silo)

Nexuses CNFGR <27> = Multi Xmit Fault
Nexuses CNFGR <26> = Xmitr During Fault

SBIA SBI Silo <15:1> = Lowest order true bit is active TR Level
                       (Note 4)
"    "   "           = 15 more entries
```

Notes

1. If the SBIA detected the Fault.

2. If the SBIA was transmitting.

3. Also indicates valid Fault status.

4. The SBI Silo also records the SBI Tag, ID, and Mask fields. SBI Silo <31> indicates that within 16 cycles after the Silo Fault Lock was cleared another Fault occurred.

PROBABLE CAUSE:

Module	Probability
L0202/SBS	High
SBI Nexus	High
L0203/SBA	Low

EHM ACTION: The EHM is not entered. Instead, the error is reported by adapter IPL 1C interrupt.

VMS ACTION: VMS logs the error and continues. Retries are left to the I/O Driver.

CHAPTER 4

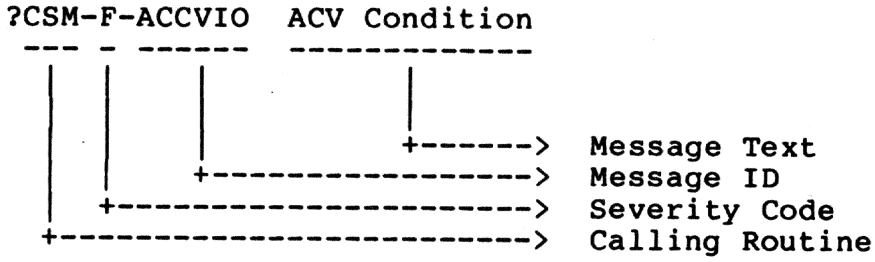
CONSOLE MESSAGES

Console Messages
(Version 9.0)

OVERVIEW

This section describes console messages for Version 9.0 of the console software. (To determine which version of the console software, EDOBA, is running, execute the SHOW VERSION console command.) It begins with a description of the console message format followed by a list of tables. The tables identify the program or routine running at the time the message was printed and the type of message (error, warning, etc). Finally the tables list, in alphabetical order, all known messages and what they mean. In many cases the description will also include a statement explaining how you might respond to the message.

CONSOLE MESSAGE FORMAT - Console Messages consist of four parts.



Part 1: The Calling Routine - begins with a question mark and includes the three character name of the routine that initiated the message. For example, in the above message the Console Support Microcode (CSM) initiated the message print out. The following is a list of Console routines that display messages.

- CSM - Console Support Microcode
- DCN - General Console
- DCP - Diagnostic Console
- ECR - Error Correction Routine
- EMM - Enviornmental Monitor Module
- HEX - Hexadecimal Debugger
- MCP - Macro Control Program

Part 2: Severity Code - consists of a letter (E, F, W, or I) inclosed in hyphens. The letter indicates the general severity of the condition.

E = Error - Indicates that the routine printing the message is responding to a device or hardware error of some sort.

F = Fatal - Indicates that the routine printing the message either detected an internal consistancy error (e.g., a bad parameter passed to a routine) or a totally unexpected or unservicable error (e.g., errors from RT).

W = Warning - Indicates that the routine printing the message was able to complete some, but not all, of the operation. You should check the result before proceeding.

I = Information - Indicates that the routine printing the message completed the operation successfully and that you should be aware of the information that follows the message header.

Part 3: Message ID - is a six letter mnemonic that identifies the message.

Part 4: Message Text - is the text of the message. It consists of a line of text that explains the reason for the message.

CONSOLE MESSAGE TABLES - The Console Messages have been further defined and organized into the following tables.

Table 1 CSM - Console Support Microcode Fatal Messages

Table 2 (DCN) General Console Error Messages
Table 3 (DCN) General Console Fatal Messages
Table 4 (DCN) General Console Information Messages
Table 5 (DCN) General Console Warning Messages

Table 6 (DCP) Diagnostic Console Error Messages
Table 7 (DCP) Diagnostic Console Fatal Messages
Table 8 (DCP) Diagnostic Console Information Messages
Table 9 (DCP) Diagnostic Console Warning Messages

Table 10 (ECR) Error Correction Routine Error Messages

Table 11 (EMM) Enviornmental Monitor Module Error Messages
Table 12 (EMM) Enviornmental Monitor Module Fatal Messages

Table 13 (HEX) Hexadecimal Debugger Warning Messages

Table 14 (MCP) Macro Control Program Error Messages
Table 15 (MCP) Macro Control Program Fatal Messages
Table 16 (MCP) Macro Control Program Information Messages
Table 17 (MCP) Macro Control Program Warning Messages

Table 18 MCPSNP.LST (KAF Snap Shot Routine) Messages

NOTE

Refer to the Console Software Specification for further information and specific details on console operation.

Also keep in mind that these message are subject to change with Console Revision changes.

Table 1 Console Support Microcode (CSM) Fatal Messages

Header	Message
?CSM-F-ACCVIO	ACV Condition - The Console does not have access to part or all or all of the data requested. By raising the PSL CUR_MODE Field Encoding and issuing the command again it may be possible to get a successful response. Also try using the physical address with memory management turned off. If the problem persists there may be a fault in the Access RAM.
?CSM-F-BADIPR	Bad IPR Number - The IPR "Number" argument is currently unassigned.
?CSM-F-BDCKSM	Bad Packet Checksum - The computed checksum of the RBUF Packets does not match the Console checksum. This could be a Dual Port RAM, CBus or EBox Microcode problem. Try re-initializing the CPU. If that doesn't work run the MHC diagnostic.
?CSM-F-BDVADR	Bad Virtual Address - The virtual address either has no translation or a bad translation.
?CSM-F-CF64KB	Can't Find 64KB - CSM was unable to find a good 64KB section of memory. There is a good chance that the MBox/Array either has a serious problem or it has not been initialized properly.
?CSM-F-CFDRPB	Can't Find RPB - CSM was unable to find the Restart Parameter Block (RPB) in memory. Either that section of memory was over written or the BBU power to support the Array Refresh is switched off or the 10 minute (BBU) Array Refresh limit expired. In any case a cold re-start is necessary. Note: This is a normal response if, after a power up, the Restart Control Switch is in the Restart Boot or Restart Halt position.
?CSM-F-HERABT	Hardware Error Abort - Something went wrong during the execution of a CSM command. The Machine Check Stack Frame built by the EHM (ESC: <17:2F> should provide information about the error). If the error persists run the Micro-Diagnostics.

Table 2 General Console (DCN) Error Messages

Header	Message
?DCN-E-BA11PF	BA11 Power Failure - The Console detected a BA11 Power failure when it read the status of the RL02. Check power at the BA11.
?DCN-E-CSPERR	"RAM ID" Control Store Parity Error - While in Console I/O Mode a CS/DRAM Parity Error was detected in the RAM specified. Use the "VERIFY" Command (in Debug context) to determine the the "good" "bad" and take the appropriate corrective action. If the Console is in Program I/O Mode the Control Store Parity Error is Uncorrectable. Following this the Console will print the reason that the Error is Uncorrectable.
?DCN-E-DVCERR	Read/Write Disk Error - A Read/Write error was detected while reading or writing a file on the RL02. Examine the RLV12 Control and Status Registers for specific details. If error is unique to a specific file re-install the file from tape or, if necessary, replace the pack.
?DCN-E-FULERR	"device name" Device Full - The RL02 is full. This may be due to fragmented files. Some file may need to be deleted or the Pack may need to be rebuilt.

Table 3 General Console (DCN) Fatal Messages

Header	Message
?DCN-F-ACCERR	"filename" Illegal Access - The user does not have the privileges necessary to access the file specified. (e.g., the file may be Read only protected.)
?DCN-F-CHNERR	Read/Write I/O Channel Invalid - (Software Problem)
?DCN-F-EIARG	Invalid Argument on "aaaaaa" Call - (Software Problem)
?DCN-F-EOFERR	Read/Write End_Of_File - the Console software encountered an unexpected EOF while reading a file. There is something wrong with the file format. Restore the file from tape.
?DCN-F-INVPD	Invalid Parameter Detected - (Software Problem)
?DCN-F-TBLERR	Table Build Fails - This is a fatal Console software error. Re-boot the system.
?DCN-F-TIMEXP	Timer Either Expired or Not Set - (Software Problem)
?DCN-F-TIMMAX	Maximum Number of Timers Set - (Software Problem)
?DCN-F-TRPERR	Trap at PC xxxxxxx - This is a fatal Console software error. Re-boot the system.

Table 4 General Console (DCN) Information Messages

Header	Message
?DCN-I-CCABRT	^C abort - The user typed Control C (^C) which interrupted the command in progress.
?DCN-I-CLKNLK	CPU Clock Frequency Unstable (Not Locked) - The System Clock is more than 60 nanoseconds out phase with the 1MHZ Reference. This message will only occur for Rev C5 Clock Modules. Either change to a more stable clock frequency or use the command "SET CLOCK FREQ QUIET" to suppress the message.
?DCN-I-CSLFAL	"ram file" Already Loaded - The file specified has already been loaded in the Control Store or Dispatch RAM.
?DCN-I-RTYCON	Remote Terminal Connected - Indicates that the Remote Terminal is connected to the Console. This message will be echoed on both the Local and Remote terminals.
?DCN-I-RTYDIS	Remote Terminal Disconnected - Indicates that the Remote Teminal has connected from the Console.
?DCN-I-VENRNG	KA860.REV/KA865.REV not found or invalid - Either the KA860.REV/KA865.REV file does not exist or it was over written. In either case the Console will not be able to check for proper microcode versions when it loads the CS/DRAMs. As this is not a Fatal Error it will not inhibit the Console from booting the system.
?DCN-I-VERINC	"ram file" Ver. Incorrect, Should Have Ver. - The CS/DRAM file just loaded is not compatable with the revision level of the system (as specified in the RL02 file: KA860.REV/KA865.REV). This message will also be be displayed if you specify an incorrect file name. (E.g., typing the Console command "LOAD/ECS CSM040" will result in this message because CSM040 is a CSM Overlay and not considered part of the Main EBox Microcode load file. The Console will still, however, load the file but you should beware of erroneous responses.
?DCN-I-VFYERC	"ram file" Verify Found "n" Microwords Bad - During CS/DRAM verification (using the "VERIFY" command) "n" microwords were found to have errors. Note: the maximum error count displayed will be 256. If the errors persist then run the MHC Diagnostic to isolate the fault and then take the appropriate corrective action. See Note 1.

Table 4 General Console (DCN) Information Messages (cont.)

Header	Message
?DCN-I-VFYERR	"ram file" Verification Fails - If The Console is in "Degbug Mode" during CS/DRAM verification this message will be displayed followed by the Good/Bad Data for the first 256 errors. See Note 1.

Note

1. The "VERIFY" command must be executed in Degbug context in order to display the "Good/Bad" Data.

   >>>> VERIFY<cr> or
   DC>> VERIFY<cr>

   Otherwise only the total number of errors will be displayed.

Table 5 General Console (DCN) Warning Messages

Header	Message
?DCN-W-CLKRER	Command Invalid with CPU Clock Running - The CPU Clock must be stopped (i.e., STOP CPU) in order for the command to execute properly.
?DCN-W-COMABT	Command Procedure Aborted - The Abort Flag was set and an error was detected while executing a command file. For troubleshooting purposes you can bypass this condition (and force the console to execute the command file anyway) by using the "SET ABORT OFF" command. See ?DCN-W-COMCWE Message below.
?DCN-W-COMCWE	Command Procedure Completed with Errors - The Abort on Error switch was not set when an error was detected while executing command file. The Console continued to execute the file, but the results are unpredictable.
?DCN-W-COMERR	Nesting Depth Exceeded - The maximum number of nested command files (4) has been exceeded.
?DCN-W-CSLERR	"ram file" CS Load/Verify Failure (file bad) - The contents of the "ram file" specified does not match the format of the CS/DRAM being loaded. Entering a command such as the following would cause this message "LOAD/ECS MCF.BPN<cr> because the MCF RAM data has a different format than the EBox CS.
?DCN-W-EIRID	Reg_ID Undefined - The Register ID that the user supplied as a command argument does not exist. Use the "SHOW REGISTERS" command to display all defined registers and register IDs.
?DCN-W-EISID	SDB_ID Not Found in CAD Tables - The SDB_ID that the user supplied as a command argument does not exist in the CAD Tables. See Note 1.
?DCN-W-EUREG	Register Name Undefined - The Register Name that the user supplied as a command argument has not been defined. Use the "SHOW REGISTERS" command to display all defined registers and register IDs.
?DCN-W-EUSIG	Signal Name Not Found in CAD Tables - The SDB Signal Name that the user supplied as a command argument does not exist in the CAD Tables. See Note 1.
?DCN-W-EUSYM	Symbol Name Not Found in CAD Tables - The V$ Symbol that the user supplied as a command argument does not exist in the CAD Tables. See Note 1.

Table 5 General Console (DCN) Warning Messages (cont.)

Header	Message
?DCN-W-FILNAM	"aaaaaa" File Not Found - The file specified does not exist on the RL02.
?DCN-W-INVCLK	Command Invalid for this Version Clock Module'
?DCN-W-INVRMT	Command Invalid from Remote Terminal - The last command entered cannot be executed from a remote terminal. (e.g., changing the Baud Rate.)
?DCN-W-INVXTL	Unassigned Crystal Mnemonic'
?DCN-W-OPNERR	"device name" Too Many Files Opened - The maximum number of files that can be opened at the same time (4) has been exceeded.
?DCN-W-PARSER	Ambiguous Command - Two or more commands match the command abbreviation. Be more specific.
?DCN-W-PARSER	Invalid Command - Either the command was entered in the wrong context or the command does not exist. Check for proper context and spelling.
?DCN-W-USESTP	Use INIT/POWER Command to Initialize EMM -

Note

1. Check the CDF860.DAT/CDF865.DAT file to be sure it contains the right CAD File names for the revision of your machine. Use the command:

   SHOW CONFIG/ASCII<cr>

Table 6 Diagnostic Console (DCP) Error Messages

Header	Message
?DCP-E-ALIVEE	DSM Alive Failure - This message is similar to the one below, however, it will only occur if PASSES=0. Generally it indicates that you are running the micro diagnostics out of sequence. Execute @TSTCPU.
?DCP-E-ALIVEE	DSM Alive Failure in Test xx - You should only get this message after you have issued a "START" command to a diagnostic. It means that the diagnostic should have finished running its current test, but has not. The microcode may be hung, or the test may have gotten into an infinite loop. All occurances of this failure should be reported to Diagnostic Engineering if they occur during the execution of @TSTCPU.
?DCP-E-BADCHK	Bad Chksum - After 6 retries, the computed checksum still did not match the checksum sent with the DSM message packet. This could be a Dual Port RAM, CBus or EBox Microcode problem. Try re-initializing the CPU. If that doesn't work run the MHC diagnostic.
?DCP-E-CONDER	Invalid Conditional Statement failed to isolate - See Note 1
?DCP-E-DSMVRS	Wrong Version of DSM Loaded -
?DCP-E-EOTBLE	End of Set Data Table Space - The user has issued more than 16 (Max) "Set Data" commands.
?DCP-E-ILLDSM	Invalid DSM Function Code - See Note 1.
?DCP-E-INVDAT	Invalid SDB data, failed to isolate - See Note 1.
?DCP-E-INVDCB	Invalid .DCB isolation file - See Note 1.
?DCP-E-INVDCI	Invalid .DCI isolation file - See Note 1.
?DCP-E-INVDID	Invalid ID failed to isolate - See Note 1.
?DCP-E-INVIND	Invalid Set Data Index - Indicates that either an Isolation Routine or the associated .COM file was mis-read or over written. Re-boot the system and try again. If that fails then try a different pack. The pack you are using may have to be rebuilt.

Table 6 Diagnostic Console (DCP) Error Messages (cont.)

Header	Message
?DCP-E-NOANSD	DSM-DC communication failure - When you see this message, the EBox microsequencer has stopped listening to the Console. This could happen because of a programming fault, because the hardware is not initialized properly or because the hardware is broken. Follow the procedure outlined under ?DCP-E-ALIVEE.
?DCP-F-STSHIN	Stash Data Type Invalid - This message should never occur. If it does it indicates that the DCI File is bad.
?DCP-E-UFLTDP	Unexpected Fault Detected at End of Pass - See Note 2
?DCP-E-UFLTDT	Unexpected Fault Detected Fatal CPU Error - See Note 2
?DCP-E-UMICTP	Unexpected Micro Trap Fatal, CPU Error - See Note 2
?DCP-E-UMICTP	Unexpected Micro Trap in Test xx at Vector xx - You should get this message only after you have started a microdiagnostic. It means that there is something wrong in the hardware that is causing Microtraps in the EBOX that the current test has not requested or tried to force. If you see this message it means that a fault is in the machine that should have been caught by a previous diagnostic, or that the machine has not been initialized properly. What you should do: 1. Enable HARDCOPY if you have a hardcopy terminal available 2. Type "SHOW Switches" 3. Type "SHOW Data" 4. Type "Examine/WBUS 6" 5. Type "Examine/WBUS 7" 6. Type "Examine/WBUS 9" 7. Type "Examine/WBUS 11" 8. Type "Examine/WBUS 12" 9. Type "Examine/WBUS 13"

Table 6 Diagnostic Console (DCP) Error Messages (cont.)

Header	Message
	10. Type "START" - This will cause the tests to run again. It will indicate whether the the problem was a spurious one time event, or a initialization or setup problem in the test microcode.
?DCP-E-UNPEOP	Unexpected Pause at End of Pass See Note 2
?DCP-E-UNPEOT	Unexpected Pause at End of Test See Note 2
?DCP-E-UNXEOD	Unexpected End of Dispatch Table See Note 2
?DCP-E-VERNDM	Version Numbers Do Not Match -

Note

1. This is most likely a software problem (DSM, DC, or the isolation file). Check to make sure that the CPU revision level matches the revision level in KA860.REV/KA865.REV on the RL02.

2. These messages should never occur unless you are micro stepping a diagnostic. If they occur during micro stepping they should be considered as informational Messages. That is, the Console Software is keeping you informed about the actions of DSM.

Table 7 Diagnostic Console (DCP) Fatal Messages

Header	Message
?DCP-F-LDFAIL	DSM Load Failure - DC was either unable to load DSM or unable to start DSM. Re-initalize the CPU and DC. If the problem persists run the Micro-Hard-Core Diagnostic.

Table 8 Diagnostic Console (DCP) Information Messages

Header	Message
?DCP-I-BADMWD	Stop on uMark Bit - A micromark bit was detected while running diagnostics.
?DCP-I-FAPAUS	Fault Detected, Pausing... The user set the /Fault:pause switch and the diagnostic detected a fault.
?DCP-I-PAUSEI	Pausing... Either the user typed Control P (^P) or the Fault Switch is set to "NOABORT" or "PAUSE".
?DCP-I-SWDATA	Set Data Command Not Invoked - The user must enter a "Set Data" command prior to entering a "Show Data" command.

Table 9 Diagnostic Console (DCP) Warning Messages

Header	Message
?DCP-W-CLKSTP	Clock Not Running - The CPU Clock is stopped and must be started for the command to execute properly.
?DCP-W-DIASTA	Diags Not Started - The user issued a "Continue" or "Step" command before starting the diagnostic.
?DCP-W-LMTRUN	Limit of Set Data ASCII Text, Truncate - The user has exceeded 30 (Max) ASCII characters in a string.
?DCP-W-FAILIS	Isolation Algorithm Failed to Isolate - There is a problem in the isolation file such that DC is unable execute the isolation algorithm.
?DCP-W-ISOEOF	End of Isolation File Encountered - DC unexpectedly encountered an EOF in an Isolation File. There is something wrong with the Isolation File.

Table 10 Error Correction Routine (ECR) Error Messages

Header	Message
?ECR-E-INTERR	CSPE Interrupt, Code Invalid (0) - The Console was interrupted to correct a CS/DRAM Parity Error, but when the interrupt code was read it was zero which is unassigned. This will result in a KAF. If the problem persists the Console module or the EBE module is the most likely cause. See CL07-CL09 and EBE3)
?ECR-E-MBTERR	"ram id" Multi-Bit-Error, Uncorrectable - The syndrome that was calculated for RAM parity error did not identify a single bit in error. Therefore, the Console software assumes that the RAM had a multiple bit error and prints this message. This will result in a KAF. See Note 1.
?ECR-E-MUNREC	MCS MBox Not Recoverable - MBox single bit CS Parity Errors are correctable but not recoverable (see Chapter 2 - MBox Control Store Correction). This will result in a KAF.
?ECR-E-NOECCD	"ram id" No ECC Data in Table - The ECC data needed to correct the CS/DRAM Parity Error specified was not in the corresponding ECC Table. This will occur if: o ^C was typed during the INIT routine before the ECC Tables were loaded. o The ECC Tables are not refreshed on a reboot. o The CS/DRAM address that the Console read (via the SDB) was nonexistent. In all cases this will result in a KAF.
?ECR-E-PCFAIL	"ram id" Correction Attempted and Failed - The Console was unable to correct the CS/DRAM Parity Error specified after 5 attempts. This will result in a KAF. See Note 1.
?ECR-E-SYNGTR	"ram id" Syndrome > RAM_Size, Uncorrectable - The syndrome generated as a result of the CS/DRAM Parity Error specified indicates that a bit beyond the size of the RAM was at fault. (e.g., bit 22 in the IBox DRAM which is only 20 bits wide.) This will result in a KAF. See Note 1.
?ECR-E-SYNZRO	"ram id" Syndrome = 0, Transient, Continuing - Indicates that the VAX8600/8650 detected a transient error associated with the CS/DRAM specified. That is, a parity error was detected and latched in the appropriate CS/DRAM logic, but when the data latch was read (or the CS RAM re-read) the data was OK. There is,

Table 10 Error Correction Routine (ECR) Error Messages (cont.)

Header	Message
	however, a very low possibility that multiple bit error occurred such that it produced a real zero syndrome. The console will report this as a transient error and no further action will be taken by the console.
?ECR-E-UPCERR	"ram id" Can't Read Box Address - The CS/DRAM correction routine was unable to read the micro-address associated with the error. This will result in a KAF. Most likely this is either an SDB failure or a failure in the addressing logic associated with the CS/DRAM specified.

Note

1. The Console Software reports "Good Data/Bad Data, Syndrome, and Address" for uncorrectable CS/DRAM Parity Errors.

Table 11 Environmental Monitor Module (EMM) Error Messages

Header	Message
?EMM-E-EMMACK	No TRANSPORT_ACK from EMM
?EMM-E-EMMACL	EMM_LAT_AC_LO Failed to Deassert
?EMM-E-EMMAFA	AIR FLOW FAULT PENDING <SHUTDOWN IMMINENT>
?EMM-E-EMMAFA	AIR FLOW FAULT PENDING <CAN'T POWER UP>
?EMM-E-EMMAFD	REGULATOR_A_OK Failed to Deassert
?EMM-E-EMMAFF	AIR_FLOW Fault Failed to Deassert
?EMM-E-EMMANO	MODULE A Not OK on EMM Power-up
?EMM-E-EMMAOK	REGULATOR_A is Not OK
?EMM-E-EMMBFD	REGULATOR_B_OK Failed to Deassert
?EMM-E-EMMBOK	REGULATOR_B is Not OK
?EMM-E-EMMBUF	EMM has No Protocol Buffer
?EMM-E-EMMCAC	CONCOLE_CPU AC LOW FAILED TO DEASSERT
?EMM-E-EMMCDC	CONCOLE_DC LOW FAILED TO DEASSERT
?EMM-E-EMMCFD	REGULATOR_C_OK Failed to Deassert
?EMM-E-EMMCOK	REGULATOR_C is Not OK
?EMM-E-EMMCOL	Data Errors (Collisions) on EMM Bus
?EMM-E-EMMDCL	EMM_LAT_DC_LO Failed to Deassert
?EMM-E-EMMDED	Console/EMM Communication Temporarily Suspended - Communication with the EMM has been suspended for 30 seconds due to excessive errors with the EMM or the EMM communication link (these errors are reported prior to the message stating that communications is suspended). While EMM communication is suspended the ALERT led will flash double-time (1/4 sec. on, 1/4 sec. off). After 30 seconds the console tries to re-establish communication with the EMM, however, failure to do so will not again be reported (but the led will continue to flash until the link is re-established).
?EMM-E-EMMDFD	REGULATOR_D_OK Failed to Deassert
?EMM-E-EMMDOK	REGULATOR_D is Not OK
?EMM-E-EMMEFD	REGULATOR_E_OK Failed to Deassert

Table 11 Environmental Monitor Module (EMM) Error Messages (cont.)

Header	Message
?EMM-E-EMMEOK	REGULATOR_E is Not OK
?EMM-E-EMMFFD	REGULATOR_F_OK Failed to Deassert
?EMM-E-EMMFOK	REGULATOR_F is Not OK
?EMM-E-EMMHFD	REGULATOR_H_OK Failed to Deassert
?EMM-E-EMMHOK	REGULATOR_H is Not OK
?EMM-E-EMMI55	EMM 5.5 Interrupt Broken -- replace EMM
?EMM-E-EMMI65	EMM Encountered Unexpected 6.5 Interrupt
?EMM-E-EMMINV	Invalid Exception Code from EMM
?EMM-E-EMMJFD	REGULATOR_J_OK Failed to Deassert
?EMM-E-EMMJOK	REGULATOR_J is Not OK
?EMM-E-EMMKAC	MOD_K_AC_LO is Asserted
?EMM-E-EMMKOK	REGULATOR_K is Not OK
?EMM-E-EMMLAC	MOD_L_AC_LO is Asserted
?EMM-E-EMMLOK	REGULATOR_L is Not OK
?EMM-E-EMMNEG	EMM Rejected Command Request Due to Present Conditions
?EMM-E-EMMPER	EMM Encountered RAM Parity Error
?EMM-E-EMMRES	No Response from EMM
?EMM-E-EMMRST	EMM Encountered Restart 1 Instruction
?EMM-E-EMMRZA	RED ZONE FAULT PENDING <SHUTDOWN IMMINENT>
?EMM-E-EMMRZA	RED ZONE FAULT PENDING <CAN'T POWER UP>
?EMM-E-EMMTRP	EMM Encountered Unexpected Trap Interrupt
?EMM-E-EMMUNK	EMM Encountered Unexpected Trap to PC 0
?EMM-E-EMMURC	Unknown Restart Code in RTDREG

Table 12 Environmental Monitor Module (EMM) Fatal Messages

Header	Message
?EMM-F-EMMMTL	EMM Protocol Message Too Long (Software Problem)
?EMM-F-EMMTIP	EMM Transmission Already in Progress (Software Problem)
?EMM-F-EMMXTO	Console-to-EMM Protocol Message Transmit Timeout - Either the switch controlling BBU power to the TOY is off or the is a problem with the TOY chip on the Console module.

Table 13 Hexadecimal Debugger (HEX) Warning Messages

Header	Message
?HEX-W-ADRFOR	Address Field Out of Range - The micro-address specified exceeds the size of the CS or DRAM. See Note 1.
?HEX-W-ANFERR	Microaddress Not Found in File - The micro-address specified does not exist in the RAM File. (e.g., CSM is not included as part of the EBox.BPN file.) See Note 1.
?HEX-W-CLKRUN	Command Invalid with CPU Clock Running - The CPU clock must be stopped ("STOP CPU") before the command can be executed properly. See Note 1.
?HEX-W-DATFOR	Data Field Out of Range - The data specified exceeds the size of the CS or DRAM.
?HEX-W-INCHNO	Invalid Channel Number - The SDB Control Channel specified does not exist. See Note 1.
?HEX-W-INVRPT	Invalid Repeat Function - The command cannot be executed with the "Repeat" function. See Note 1.
?HEX-W-INVSID	Invalid ID or Name - The last SDB ID or SDB Name entered does not exist. Make sure the correct CDF files are loaded for the CPU revision level ("SHOW CDF860.DAT/ASCII" or "SHOW CDF865.DAT/ASCII). See Note 1.

Note

1. Use the Console Software "HELP" command or refer to the Console Software Spec for more details on command usage.

Table 14 Macro Control Program (MCP) Error Messages

Header	Message
?MCP-E-C40ICE	CSM040 Hung During CPU Initialization - CSM was unable to properly initialize itself. This indicates that there is an interaction problem between the EBox and one of the other boxes. If the problem persists run MHC and the Microdiagnostics to isolate the fault. If all else fails try single stepping through the CSM Overlay.
?MCP-E-CBADCK	CSM Sent Bad Checksum - The computed checksum did not match the checksum sent with the last CSM message. This could be a Dual Port RAM, CBus or EBox Microcode problem. Try re-initializing the CPU. If that doesn't work run the MHC diagnostic.
?MCP-E-CSMLOP	CSMs Console Loop Not Running" - The EBox is either hung or not started. To clear this condition either start the CPU (if ^P was typed) or use the INIT/CPU if the EBox is hung. See Note 1.
?MCP-E-NOCSMR	No Acknowledgment from CSM - CSM failed to respond to a request. If this occurred during a INIT/CPU command chances are CSM is hung trying to execute a command. See Note 1.
?MCP-E-NODATA	CSM Sent No Data Packet - The command protocol includes a data packet but CSM failed to send a data packet. See Note 1.
?MCP-E-NODATW	CSM Sent Unexpected Data Packet - CSM either sent an extra or unexpected Data Packet when a Data Packet was not part of the command protocol. See Note 2.
?MCP-E-NORPKT	No Response Packet Received from CSM - CSM is most likely hung trying to perform some kind of an initialization. See Note 1.
?MCP-E-PK2CNS	CSM Pkt2, Cntl Not Set See Note 2.
?MCP-E-UNEXPD	CSM Sent Data for Non-data Response See Note 2.
?MCP-E-UNKCSM	Unknown CSM Packet Code See Note 2.

Table 14 Macro Control Program (MCP) Error Messages (cont.)

Header	Message
?MCP-E-XCCHKS	X Command Cmd_Check_Sum Failure - The Checksum calculated by MCP did not match the Checksum associated with the command portion of the X Command. See Note 3.
?MCP-E-XDCHKS	X Command Data_Check_Sum Failure - The Checksum calculated by MCP did not match the Checksum associated with the data portion of the X Command. See Note 3.
?MCP-E-XRTIMO	X Command Receiver Time_out - Once the X Command has established a link the console expects a byte no less than once a second. This message indicates that the X Command has not send a byte during the last second. See Note 3.

Note

1. To determine where CSM is Hung:

   a. Type "STOP CPU"

   b. Type "MIC" - This will cause the current Microsequencer PCs to be typed on the terminal.

   c. Type "space bar" 10 more times - This causes a whole sequence of Microsequencer PCs to be typed out. This helps us to find out what the CPU thinks its doing.

   d. Type "return" - This gets you out of MIC mode.

   e. Type "UNHANG"

   f. Type "@STKFRM" - The EHM may have built a Machine Check Stack Frame for this error condition. This command will dump ESC:17 through 2F.

   g. Refer to the Console Software Spec to determine which CSM Overlay was loaded.

2. These are CSM protocol problems. First try re-loading the EBox Microcode. If that doesn't work run the Micro Diagnostics

3. The X Command is used to down line load files from a host system to the T11. It was intended for use during the engineering debug phase of the project. It is not intended to be used by the FIELD.

Table 15 Macro Control Program (MCP) Fatal Messages

Header	Message
?MCP-F-ABSDED	ABus Dead - Indicates that ABus Dead was detected while in Program I/O Mode. An ABus Dead condition is treated just like a Power Fail condition.
?MCP-F-INVCSM	Invalid CSM Overlay Number (Software Problem)
?MCP-F-PWRFAI	AC Power Failure - Indicates that an AC Input Unit (H7170) detected a loss of AC Power and sent the signal AC LO to the EMM. The EMM in turn sends EMM3 CPU AC LO to the I/O Adapters (SBIAs) and the Console. The I/O Adapters in turn generate SBAQ SBI FAIL which notifies the Nexus of the loss of AC power. This allows the Nexus to execute an orderly power fail shut down. The Console, after receiving EMM3 CPU AC LO, sends CL09 CPU PF INTER to the VAX8600/8650. Then both the Console and the VAX8600/8650 begin executing the power down sequence. After loosing AC power the system has approximately 10 milliseconds before the DC power becomes unstable. This allows the system to sweep cache and save the state of the CPU (i,e., GPRs etc.). If battery back up is enabled the system will be able to preserve the state of the arrays for approximately 10 minutes. Thus the system will be able to recover from short term losses of AC power.
?MCP-F-PWRFAL	Power Fail - Indicates that a Power Fail condition was detected while in Program I/O Mode.

Table 16 Macro Control Program (MCP) Information Messages

Header	Message
?MCP-I-BBUINV	Battery Backup Unit Invalid -
?MCP-I-CPSRUN	CPU is Still Running - The early versions of the Console software would automatically halt the VAX8600/8650 when ^P was typed. On later versions this was changed. The Console software no longer halts the VAX8600/8650 when ^P is typed. Instead it prints this informational message. You must type "HALT CPU" to stop the VAX8600/VAX8650.
?MCP-I-HDECOL	Hardware Not Up to Proper ECO Level - According to the System ID Register (SID) and KA860.REV/KA865.REV the files on the RL02 are not compatable with the system. Check KA860.REV/KA865.REV against the system ID.
?MCP-I-MCLDST	Aborting Redundant Cold-Start Attempt - A Cold Restart has been attempted and failed. Second and subsequent automatic Cold Restarts are aborted.
?MCP-I-MWRMST	Aborting Redundant Warm-Start Attempt - A Warm Restart has been attempted and failed. A second Warm Restart attempt will be aborted and a Cold Restart will be attempted if enabled by the via the System Control Panel).
?MCP-I-LARPWR	Lost Array Refresh Power (warm start not possible) - Either the BBU is switched off or the 10 minute time limit for the Battery Back Up Unit to supply power to the array refresh circuit expired. A warm re-start is no longer possible. A Cold Restart will be attempted. If enabled via the System Control Panel.
?MCP-I-NOPAMM	Command Invalid, PAMM Not Init'd, Do INIT/PAMM
?MCP-I-RPBBSY	RPB Restart-in-Progress Flag Set - The warm restart attempt failed. If the Restart Control Switch is in the correct position (RESTART BOOT) a cold restart will be attempted.
?MCP-I-RPBINV	RPB Invalid/Not Found - The RPB in memory is not valid. The warm restart must be aborted. If the Restart Control Switch is in the correct position (RESTART BOOT) a cold restart will be attempted.

Table 17 Macro Control Program (MCP) Warning Messages

Header	Message
?MCP-W-ADROOR	Address Out of Range - The address specified is outside the the range for Examine/Deposit Commands. See Note 1.
?MCP-W-CPHUNG	CPU is Hung - Indicates that the VAX8600/8650 has stopped running for some reason. Usually this message is either proceeded by or followed by a message that explains the reason that the CPU is Hung. For example this message will almost always precede a Keep Alive Fail condition when the Snap File mechanism is enabled. The only exception is when the CPU directs the Console (via a CSM Code of 1B) to Halt the system. There are a number of cases, however, when a second message explaining the reason for the HUNG CPU will not be printed. For example, if the Snap File mechanism is disabled when a KAF condition occurs no additional messages will be printed. The same kind of situation would occur if a WBus Parity Error was detected while the Console was in Console I/O mode. One way to approach this problem is to stop the CPU and single step (MIC) the EBox. Look for: 1. One of the following EBox UPCs: a. UPC 20 - Indicates a parity error in both the A and B RAMs b. UPC 21 - Indicates a double error condition c. UPC 24 - Indicates a WBus Parity Error 2. A tight microcode loop. Generally the microcode is looping waiting for some event to occur. Look in the listings to determine what the event is. Then try to determine why the event did not occur. 3. An EBox or IBox Stall condition. Examine the SDB ESTALL and ISTALL Registers. If a stall condition exists try to determine if it is the cause of the CPU HUNG condition.
?MCP-W-DATOOR	Data Out of Range - The Data specified exceeds 32 bits. See note 1.
?MCP-W-INVIPR	Invalid IPR Number - The IPR "Number" argument is currently unassigned. See Note 1.

Table 17 Macro Control Program (MCP) Warning Messages (cont.)

Header	Message
?MCP-W-INVRRW	Read or Write Not Allowed - The command specified is not appropriate because the IPR is either Read or Write only. See Note 1.
?MCP-W-XINCOM	X Command in .COM File Invalid - X Commands cannot be executed in a command file. See Note 1.

Note

1. Use the Console Software "HELP" command or refer to the Console Software Spec for more details on command usage.

Table 18 MCPSNP Messages (KAF Snap Shot Routine)

The KAF Routine prints the following banner followed by a reason message.

"Attempting to save machine state due to:"

DOUBLE ESCRATCH PARITY ERROR - (KAF Reason Code: 18)

The Error Handling Microcode (EHM) determined that there was a parity error in both copies of the EBox Scratch Pad RAMs. Most likely this condition occurred when the EHM was attempting to correct a GPR parity error by copying the good GPR to the bad GPR. This is a non-recoverable error condition. The EHM responded to this error condition by looping at EBox UPC 20 which in turn resulted in this KAF.

This is a sticky problem. There is a good chance that the GPR parity error will once again be detected when the KAF routine uses CSM to read the EBox Scratch Pad RAMs. If that is the case then the EScratch Record of the SNAP file will contain the contents of the EScratch up to the point where the parity error occurred. The remainder of that section will contain all ones (FFF...). In addition to being unable to copy some or all of the EScratch, the KAF routine will be unable to copy the CPU IPRs, the PAMM, the top 64 longwords on the Interrupt Stack, and the SBIA/Nexus Registers.

Probable Cause:

Module	Probability	RAMs (GPRs)
L0209/EDP	High	E500, E501, E502, E503 E613, E614, E615, E617 E712, E713, E714, E715 E812, E813, E814, E815 E903, E907
L0219/EBE	Low	

MACHINE DOUBLE ERROR - (KAF Reason Code: 19)

The Error Handling Microcode (EHM) was in the process of handling an error when a second EBox related error was detected. This is a non-recoverable error condition. The EHM responded to this error condition by looping at EBox UPC 21 which, in turn resulted in this KAF.

Approach: Contact the RDC and request that they analyze the SNAP file. Meanwhile, if possible use VSRBLD to translate the SNAP file. Otherwise type "SHOW SNAP1.DAT and translate the SNAP file manually. The ESC should contain a partial Machine Check Stack Frame for the first error. However, beware that the EHM over wrote the following EScratch locations with status from the second error.

ESC: 12 (Trap Vector)
ESC: 15 (EBCS)
ESC: 19 (VMQ)
ESC: 2F (PSL)

Table 18 MCPSNP Messages (KAF Snap Shot Routine) (cont.)

WBUS PARITY ERROR - (KAF Reason Code: 1A)

The EBox updated all copies of the GPRs with bad parity. This is a non-recoverable error condition. The EHM responded to this error condition by looping at EBox UPC 24 which in turn resulted in this KAF.

The symptoms indicate that the parity calculated on the WBus data did not match the parity calculated at the output of the WReg. Most likely the IBox or FBox is contaminating the WBus or one of the parity generators is bad.

Probable Cause:

Module	Probability
L0209/EDP	High
L0219/EBE	High
L0206/IDP	Low
L0212/FBA	Low
L0223/FTM	Low (FBox Terminator)

CPU ERROR HALT - (KAF Reason Code: 1B)

The KAF Condition was initiated by the Console Support Microcode (CSM). The specific reason for the KAF is contained in the Master Header Record Byte 13 (CSM Status Word Entry Code.) See list below.

0 = CSM could not be forced to run by the console program after the KAF. The EBox microcode may have been corrupted, re-initialize the system. If that doesn't work run the Micro-diagnostics.

4 = Interrupt Stack not valid. The CPU was processing an interrupt or an exception, but when it attempted to push "state" on the Interrupt Stack it discovered that the Interrupt Stack was mapped "NO ACCESS" or "NOT VALID". This generally indicates that the CPU got into a loop handling an interrupt or exception and as a result the interrupt stack overflowed to a page mapped "NO ACCESS" or "NOT VALID".

Translate the SNAP file. Look at the last 64 longwords on the Interrupt Stack. That may provide a clue to the loop that the CPU was in. Also look at the contents of the CSLINT register. If the CPU was in an interrupt loop you may get some idea of the source. Finally, look at EHSR and the Machine Check section of the EScratch Record to determine if the CPU was handling an error.

Table 18 MCPSNP Messages (KAF Snap Shot Routine) (cont.)

5 = Non-EBox or "VMS ENTERED" double error. There are two ways this condition can occur:

a. If the EHM was processing an error and a second (non-EBox) error was detected, the EHM will call CSM with a code of 5 in CSM.STATUS. In this case you will find a partially build Stack Frame (for the first error) in the EScratch. The vector address in EHSR will identify the port that reported the second error.

b. If the VMS Machine Check Handler was handling an error and a second error was detected, the EHM will build a Stack Frame for the second error and then call CSM with a code of 5 in CSM.STATUS. In this case you will find the first Machine Check Stack Frame on the interrupt stack,, and the second Machine Check Stack Frame in the EScratch.

6 = Kernel Mode HALT. The processor executed a HALT instruction while in Kernel Mode. Examine the SDB EBOX OPCODE Register to determine if the processor actually executed a halt. If so examine EBox Scratch Pad location 2E to determine the PC of the Halt Instruction. Then use the VMS listings to look up the reason that VMS Halted.

7 = SCB vector with <1:0> = 3. The Vector Code Field in the System Control Block <1:0> was equal to 3 which is a reserved code. Either the SCB was overwritten or a wrong vector address was generated.

8 = SCB vector with <1:0> = 2. The Vector Code Field in the System Control Block <1:0> was equal to 2 which means: service this event in Writeable Control Store (WCS). However the WCS either does not exist or was not loaded. The result of the operation is a HALT. Again, either the SCB was overwritten or a wrong vector address was generated.

9 = Pending error on HALT. The CPU was in the process of handling an error condition when P was typed on the console. The Interrupt Stack, the Machine Check section of the EScratch and the contents of EHSR may provide some idea of the type of error the CPU was processing when P was typed.

A = CHMx with IS = 1. The CPU executed a Change Mode instruction when PSL <26> (Interrupt Stack) was set.

B = CHMx vector <1:0> not 0. The CPU executed a Change Mode instruction and the SCB Vector Code <1:0) was not equal to zero.

Table 18 MCPSNP Messages (KAF Snap Shot Routine) (cont.)

UNCORRECTABLE CS PARITY ERROR - (KAF Reason Code: 1C)

The Console was unable to correct a Control Store or Dispatch RAM for one of the following reasons:

INTERR CSPE Interrupt, Code Invalid (0)

MBTERR "ram id" Multi-bit-error, Uncorrectable

MUNREC MCS MBox Not Recoverable

NOECCD "ram id" No ECC Data in Table

PCFAIL "ram id" Correction Attempted and Failed

SYNGTR "ram id" Syndrome > RAMSize, Uncorrectable

SYNZRO "ram id" Syndrome = 0, Uncorrectable

UPCERR "ram id" Can't Read Box Address

Refer to Table 10 for a description of uncorrectable Control Store and Dispatch parity errors. Examine the contents of CSES to determine the RAM, Address, and Syndrome. Then refer to the RAM Callout Tables in the section on Manual Stack Frame Analysis to identify the failing RAM.

POWER SYSTEM FAILURE (DC LOW) - (KAF Reason Code: 1D)

During a normal power failure AC Low will proceed DC Low by approximately 10 milliseconds. This allows VMS enough time to sweep the cache, save the state of the GPRs and CPU Registers, etc. If, however, a DC regulator fails (resulting in DC Low only) then this message is printed and a KAF results. Review the EMM Record of the Snap file to determine the exact cause of the failure.

UNKNOWN MACHINE HANG - (KAF Reason Code: 1E)

The KAF timer expired and the KAF Routine read the EBox UPC and the CSM Status word but was unable to determine the cause of the KAF. Most likely the system is stalled (ESTALL or ISTALL) or hung in a micro loop waiting for some event to occur.

Approach: Contact the RDC and request that they analyze the SNAP file. Meanwhile, if possible use VSRBLD to translate the SNAP file. Otherwise translate the SNAP file manually. The ESC may contain a partial Machine Check Stack Frame.

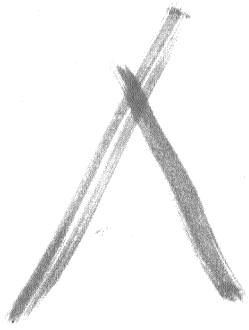
Additional MCPSNP.LST (KAF Routine) Messages

Both SNAP Files Still Valid

The KAF Routine was called to capture (Snapshot) the state of the system but both SNAP Files were still valid. That is, they still contain status captured during the previous two KAF Conditions. As a result the Console will not capture the state of the machine. Use the Console command "SET SNAP INVALID" to invalidate both SNAP Files.

SNAPx.DAT Created

Currently there two Snap file names used by the console; SNAP1.DAT and SNAP2.DAT. This message tell you the name of the Snap File that the KAF Routine created as a result of the KAF Condition.



CHAPTER 5

THE VMS SYSTEM EVENT FILE

THE VMS SYSTEM EVENT FILE

OVERVIEW

The VMS Operating System maintains a System Event File called ERRLOG.SYS. The file is located in the SYS$SYSROOT:[SYSERR] directory and is used to record certain events that occur during system operation. The types of events that are recorded in the event file are listed in Table 1.

Table 1 System Event File Entry Type Definitions

Entry Code	Description
01	Device Error
02	Machine Check
04	Bus Error
05	SBI Alert
06	Soft ECC Error
07	Asynchronous Write Error
08	Hard ECC Error
09	11/780 Unibus Adapter error
10	11/750 Fault Through SBI Vector
11	11/730 Unibus Error
12	11/780 Massbus Adapter Error
13	KA86 SBIA Error
14	KA86 CRD Log
15	KA86 Environmental Monitor
16	KA86 Processor Error Halt
17	KA86 Console Reboot
32	Cold Start (ie: System Boot)
35	New File Created
36	Warm Start (ie: System Power Recovery)
37	Crash Re-start
38	Time Stamp Entry
39	System Service Message
40	System Bugcheck
41	Operator Message
42	Network Message
64	Volume Mount
65	Volume Dismount
96	Device Timeout
97	Undefined Interrupt
98	Asynchronous Device Attention
99	Software Parameters
100	Logged Message
101	Logged MSCP Message
112	User Bugcheck
273	Unknown Entry

Each time one of these events occur, the normal operation of VMS is interrupted and a special routine is called to handle the event. The routine requests a System Event Buffer and then gathers predefined

information about the event (e.g., system status, hardware and software registers, etc.) and puts it in the buffer. Once the buffer is built the routine queues a request to append the buffer to the System Event File. When the queue is processed the buffer is appended to SYS$SYSROOT:[SYSERR]ERRLOG.SYS. This process takes place anytime one of the events listed in Table 1 occur.

Two programs (ANALYZE/ERRORLOG and RETRIEVE - a Spear Library function) are available to translate the contents of the System Event File into ASCII reports. Both of these programs use the Error Log Formatter (ERF) to translate the entries in the Event File. Therefore, regardless of which program you use the format of the translated entries will be the same. The main difference between the two programs is the command syntax, the selection criteria, and the format of the Summary reports they produce. In addition to translating system event file entries Spear is capable of analyzing the contents of the event file and calculating system availability.

ANALYZE/ERRORLOG

ANALYZE/ERRORLOG uses a non-interactive command syntax. That is, the Command, Qualifiers and Arguements, are entered in a single string. The Qualifers allow you to select specific entries from a binary System Event File and either produce a seperate binary event file that contains only those entries, or translate the entries and produce an ASCII Report. For a complete description of the this utility, including more information about the ANALYZE/ERRORLOG command and its qualifiers , see the VAX/VMS Utilities Reference Volume.

COMMAND SYNTAX:

ANALYZE/ERROR_LOG [/qualifier=arguement][,...]] [file-spec[,...]]<cr>

- The base command can be abbreviated to ANA/ERR.
- All Qualifiers are preceeded by a slash.
- Multiple Arguements to a Qualifier are seperated by a comma.
- In some case special characters such as the equal sign, parenthese, and colon are required. If the qualifier requires special characters they will appear in the syntax examples shown in Table 2.

Table 2 summarizes the qualifiers and defaults associated with ANALYZE/ERROR_LOG. The full qualifier is spelled out in the left column. In the syntax example to the right of the column the qualifier is abbreviated to its most common form. The effect of the qualifier is described below the syntax example.

Table 2 ANALYZE/ERROR_LOG Command Qualifiers and Defaults

ANALYZE/ERROR_LOG ANA/ERR<cr>

Translate the entire system event file SYS$SYSROOT:[SYSERR]ERRLOG.SYS and output a full ASCII report of each entry on the terminal.

This is the default case. The defaults are as follows:

INPUT FILE - The default input file spec is SYS$SYSROOT:[SYSERR]ERRLOG.SYS.

OUTPUT - The default output is an ASCII report. which is sent to SYS$OUTPUT. The system default for SYS$OUTPUT is your terminal.

QUALIFIERS - The default qualifiers are: /FULL /ENTRY=(START:1,END:EOF)

ANALYZE/ERROR_LOG ANA/ERR ERRLOG.OLD<cr>

Translate the entire system event file specified (ERRLOG.OLD;5) and output a full ASCII report of each entry on SYS$OUTPUT.

With the exception of the input file specification the defaults for this case are the same as above. Any binary (untranslated) system event file may be specified as input.

/BEFORE ANA/ERR/BEF=16-AUG-85-10:35 ERRLOG.OLD;5<cr>
ANA/ERR/BEF=-3-:12:30 ERRLOG.OLD;5<cr>

Select only those entries dated earlier than the "date-time" specified.

The qualifier accepts absolute time (begining August 16,1985 at 10:30), delta time (begining 2 days, 11 hours, and 30 minutes ago), or a combination of both. For further details on specifying times refer to Section 2.5 in the VAX/VMS DCL Dictionary.

/BINARY ANA/ERR/INCLUDE=(DISKS)/BIN=FS:DISK.ERRORS<cr>

Do not translate the selected entries. Instead write them in the directory and file specified. If no directory is specified use the users default directory. If no file type is specified, use .DAT as the file type.

You must supply a file name. If you omit the directory it will default to the directory you are using. If you omit the file type it will default

to: DAT.

The following qualifiers should not be used in conjunction with the /Binary qualifier:

/BRIEF /OUTPUT /SUMMARY
/FULL /REGISTER_DUMP

/BRIEF

ANA/ERR/BRI ERRLOG.OLD;5<cr>
Do not generate a full report for each selected entry. Instead generate an abbreviated report containing key only information about each entry.

/ENTRY

ANA/ERR/ENT=(START:12,END:29)<cr>
Select only the Entry Numbers specified. If either the START or END arguement is omitted default to START:1,END:EOF.

/EXCLUDE

ANA/ERR/EXC=(MTA0,DRA5) ERRLOG.OLD;5<cr>
Do not select any entries generated for the Device Class, Device Name, or Entry Type specified.

The acceptable Device and Entry keywords are listed under the /INCLUDE qualifier.

/FULL

ANA/ERR/INCLUDE=(DISKS)/FULL ERRLOG.OLD;5<CR>
Generate a full ASCII report for the entries specified.

This is the default report format and normally does not need to be specifed as part of the ANALYZE/ERROR_LOG command string.

See Examples: 1 through 15

/NOFULL

ANA/ERR/STATISTICS/NOFULL ERRLOG.OLD;5<CR>
Do not generate a full ASCII report for the entries specified.

This Qualifier is normally used when you only want a special ASCII report such as a Summary or Statistical report. If you don't specify NOFULL, a full translation of the selected entries will preceede the Summary or Statistical Report.

/INCLUDE

ANA/ERR/INC=(MACHINE_CHECKS,BUGCHECKS)<cr>
Select only those entries generated for the Device Class, Device Name, or Entry Type specified.

The acceptable Device and Entry keywords are listed below.

Device Class Keywords

BUSES - All Bus related Entries
DISKS - All Disk Related Entries
REALTIME - All Realtime Related Entries
SYNC_COMMUNICATIONS - All Synchrounious Line Entrie

TAPES - All Tape related Entries

Device Physical Name Constructs

DB - An entire group of devices
DB,DR,XF - A list of device groups
DBA1 - A specific device/unit number
DBA1,HSC1$DUA1,DYA0 - A list of devices

Entry Types

ATTENTIONS - device attention entries
BUGCHECKS - bugcheck entries
CONTROL_ENTRIES - Control Entries
CPU_ENTRIES - CPU Related Entries
DEVICE_ERRORS - Device Error Entries
MACHINE_CHECKS - Machine Check Entries
MEMORY - Memory Error Entries
TIMEOUTS - Device Timeout Entries
UNKNOWN_ENTRIES - All Entries that had either an unknown entry type or an unknown device type/class.
UNSOLICITED_MSCP - Unsolicited MSCP Entries
VOLUME_CHANGES - Volume Mount and Dismount Entries

/LOG

ANA/ERR/LOG ERRLOG.OLD;5<cr>
Send a message to the SYS$OUTPUT stating the number of entries that were selected and rejected for each input file.

Refer to the /REJECT qualifier for an explaination of rejected entries. See Example: 16

/OUTPUT

ANA/ERR/OUT=ERROR_LOG.LST ERRLOG.OLD;5<cr>
Do not print the ASCII Report. Instead save the report in the file specified. If no file is specified write the report into xxxx.LST (where xxxx is the name of the input file).

/REGISTER_DUMP

ANA/ERR/INCLUDE=(CPU)/REG ERRLOG.OLD;5<CR>
Do not use the specified (Brief/Full) format for translating Memory, Device Error, and Device Timeout entries. Instead select only the register information from those entries and translate that information into hexadecimal longword (cryptic) format. Use the specified format for translating all other types of selected entries.

This qualifier requires that the INCLUDE qualifier be part of the command string. Also, regardless of whether or not they were specified as by INCLUDE Qualifier, all Memory, Device Error, Device Timeout entries will be selected and translated in cryptic format.

See Example: 17

/REJECT

```
ANA/ERR/INCLUDE=(MTA0)/REJ=ERRORS.BIN
ERRLOG.OLD;5<CR>
```

Put all rejected entries in the file specified. Do not translate the entries, write them in binary format. If no file is specified write the entries into xxxx.REJ (where xxxx is the name of the input file).

Rejected entries consist of all entries that were not specifically selected in the command string. That is, those entries that were outside the time window specified by either the /SINCE, /BEFORE arguements; those entries that were not with in the range specified by the /ENTRY(START: ,END:) arguements; those entries that did not match the /INCLUDE arguements; and those entries that were specifically rejected by the /EXCLUDE agruements.

/SID_REGISTER

```
ANA/ERR/SID=%X0405F09E ERRLOG.OLD;5<CR>
```

Select only those entries that were reported by the CPU associated with the System ID specified.

/SINCE

```
ANA/ERR/SIN=16-AUG-85-10:35 ERRLOG.OLD;5<cr>
ANA/ERR/SIN=-3-:12:30 ERRLOG.OLD;5<cr>
```

Select only those entries that occurred on or after the date and time specified.

You can specify an absolute time (begining August 16,1985 at 10:30), a delta time (begining 2 days, 11 hours, and 30 minutes ago), or a combination of absolute and delta times. For further details on specifying times refer to Section 2.5 in the VAX/VMS DCL Dictionary.

/STATISTICS

```
ANA/ERR/NOFULL/STAT ERRLOG.OLD;5<cr>
```

Generate and append a statistical report to the end of the ASCII report that states CPU Time used and the number of page faults, buffered I/O, and direct I/O, that occurred during the execution of the ANALYZE/ERROR_LOG command.

See Example: 18

/SUMMARY

```
ANA/ERR/NOFULL/SUM=(DEV,MEM) ERRLOG.OLD;5<cr>
```

Generate a summary report for each of the report types specified by the keyword and append the report(s) to the end of the ASCII report. If no keywords are supplied, generate a full set of summary reports.

The following is a list of the Summary Keywords and the type of report they will generate.

Keyword	Meaning
DEVICE	Include the Device Rollup section in the report.

ENTRY	Include the Summary of Entries Logged section in the report.
HISTOGRAM	Include the Processed Entries Hour of Day Histogram in the report.
MEMORY	Include the Summary of Memory Errors section in the report.
VOLUME	Include the Volume Label section in the report.

See Examples: 19 through 22

SPEAR

In contrast to ANALYZE/ERRORLOG, Spear uses an interactive command syntax. The user is prompted for arguements and qualifiers. In addition to interactive prompting Spear supports an extensive help facility as well as a built in tutorial function called INSTRUCT. Because of this online documentation it is not necessary to document the Spear dialogue in this manual. Instead you are directed to run Spear, review the Instruct package and then use the help facility as necessary.

EXAMPLES

The following examples represent sample reports produced by the Error Record Formatter (ERF). These reports have been included so that you will have some idea of the type of information that can be extracted from System Event files using either ANALYZE/ERROR_LOG or SPEAR.

The following is a list of the examples and the corresponding Entry Types:

Example 1: (Entry Type 002) Machine Check
Example 2: (Entry Type 006) Soft ECC Error
Example 3: (Entry Type 013) KA86 SBIA Error
Example 4: (Entry Type 015) 11/780 Environmental Monitor
Example 5: (Entry Type 016) KA86 Processor Halt

Example 6: (Entry Type 017) KA86 Console Reboot
Example 7: (Entry Type 032) Cold Start
Example 8: (Entry Type 037) Crash Re-start
Example 9: (Entry Type 040) System Bugcheck
Example 10: (Entry Type 096) Device Timeout

Example 11: (Entry Type 098) Asynchronous Device Attention
Example 12: (Entry Type 273) Unknown Entry
Example 13: ANALYZE/ERROR_LOG/LOG Report Format
Example 14: ANALYZE/ERROR_LOG/REGISTER_DUMP Report Format
Example 15: ANALYZE/ERROR_LOG/STATISTICS Report Format

Example 16: ANALYZE/ERROR_LOG/SUMMARIZE=(DEVICE) Report Format
Example 17: ANALYZE/ERROR_LOG/SUMMARIZE=(VOLUME) Report Format
Example 18: ANALYZE/ERROR_LOG/SUMMARIZE=(ENTRY) Report Format
Example 19: ANALYZE/ERROR_LOG/SUMMARIZE=(HISTOGRAM) Report Format

Example 1: Machine Check (Entry Type 002)

```
 V A X / V M S        SYSTEM ERROR REPORT      COMPILED  6-SEP-1985 16:46
                                                            PAGE   1.

 ***************************** ENTRY       1. *****************************
 ERROR SEQUENCE 43.                                LOGGED ON SID 0405F270

 MACHINE CHECK    2-JUL-1985 17:34:44.00
                  KA86  REV# 5.  SERIAL# 624.  MFG PLANT 15.

       EHMSTS           41001803
                                        VMS ERROR CODE = IBOX
                                        MICRO TRAP VECTOR = 18 (X)
                                        IBOX SP CORR
                                        EHM ENTERED
       EVMQSAV          0008073D
                                        VIRTUAL ADDRESS FOR EBOX PORT
                                        _ REQUESTS
       EBCS             00002000
                                        IBOX ERR
       EDPSR            00000000

       CSLINT           00606E1F
                                        C BUS ADDRESS = 1F (X)
                                        C BUS DATA = 6E (X)
                                        INTERRUPT PRIORITY REQUEST = 0.
                                        I/O ADAPTER = 3.
       IBESR            00806000
                                        UOP SEL = IBOX REGISTER SELECT
                                        UTPR <2:0> = FORK(IB PORT, IBOX ERR)
                                        ENABLE ETRAP
                                        IAMUX PARITY ERROR
       EBXWD1           00000051
                                        TOP OF "SP STACK"
                                        _ CONTENT IS ONE OF THE LAST
                                        _ LONGWORDS WRITTEN TO MBOX
       EBXWD2           00A00040
                                        TOP OF "SP STACK" MINUS ONE
                                        _ CONTENT IS ONE OF THE LAST
                                        _ LONGWORDS WRITTEN TO MBOX
       VASAV            00011B04
                                        VIRTUAL ADDRESS FOR OP FETCH
                                        _ PORT REQUEST ADDRESS
                                        _ CALCULATION FOR OPERAND
                                        _ PRE-FETCH AND RESULT DELIVERY
       VIBASAV          0008074E
                                        VIRTUAL ADDRESS OF NEXT IBUF
                                        _ PORT REQUEST TO FILL IBUFFER
       ESASAV           0008073E
                                        PC OF INSTRUCTION DURING EBOX
                                        _ EXECUTION AND RESULT STORAGE
       ISASAV           00080742
                                        PC OF INSTRUCTION WHICH VA
                                        _ CALCULATION UNIT IS DOING ADDRESS
                                        _ CALCULATION OR OPERAND PRE-FETCH
                                        _ OR IS PASSING OPERAND DATA
```

```
CPC             00080742
                        PC OF INSTRUCTION IN
                        _ DECODE UNIT
MSTAT1          84004000
                        BLOCK HIT
                        ABUS ADAPTER = 0.
                        WORD COUNT = 0.
                        CYCLE TYPE = READ REGISTER
                        DEST CP = EBOX
MSTAT2          00000F00
                        DIAGNOSTIC STATUS FROM SBIA
                        _     RD COM/MSK <3:0> = F (X)
                        _     RD DAT L/S <1:0> = 0 (X)
                        PAMM DATA = ARRAY #0.,SLOT #1.
MDECC           00060400
                        (* DATA NOT VALID *)
MERG            00000100
                        MEMORY MANAGEMENT ENABLE
CSHCTL          00001003
                        CACHE 0 ENABLE
                        CACHE 1 ENABLE
MEAR            0000007C
                        PHYSICAL ADDRESS IN PA LATCH
                        AT TIME OF ERROR = 0000007C
MEDR            0000001F
                        DATA WORD USED DURING ERROR
FBXERR          FFFFFFFF
                        (* DATA NOT VALID *)
CSES            FFFFFFFF
                        (* DATA NOT VALID *)
ERROR PC        00080742
ERROR PSL       03C00028
                        N-BIT
                        INTEGER OVERFLOW TRAP ENABLE
                        INTERRUPT PRIORITY LEVEL = 00.
                        PREVIOUS MODE = USER
                        CURRENT MODE  = USER
IOA ES          00000000
                        (* DATA NOT VALID *)
```

Example 2: Soft ECC Error (Entry Type 006)

```
V A X / V M S        SYSTEM ERROR REPORT      COMPILED  1-OCT-1985 08:57
                                                        PAGE   1.

****************************** ENTRY       1. ******************************
ERROR SEQUENCE 209.                                LOGGED ON SID 04FFFFFF

CORRECTED MEMORY ERROR  10-SEP-1985 17:15:19.03
                        KA86  REV# 255.  SERIAL# 4095.  MFG PLANT 15.

HIGH CRD ERROR RATE - CRD LOGGING DISABLED

TOTAL CORRECTED DATA ERRORS LOGGED FOR THIS ENTRY 9.

CORRECTED ERROR 1.

      MDECC             00260000
                                      SYNDROME = CORRECTED CHECK BIT C0.
                                      DATA SINGLE BIT ERROR

      MEAR              3FFFFFFC
                                      PHYSICAL ADDRESS IN PA LATCH
                                      AT TIME OF ERROR = 3FFFFFFC

      MSTAT1            C3000000
                                      ABUS ADAPTER = 0.
                                      WORD COUNT = 3.
                                      CYCLE TYPE = NOP
                                      DEST CP = IBF (LOAD IBTP FROM MCC)

      MSTAT2            09000001
                                      MBOX LOCK
                                      PAMM DATA = ARRAY #0.,SLOT #1.
                                      SMU8.

CORRECTED ERROR 2.

      MDECC             00261400
                                      SYNDROME = CORRECTED DATA BIT #1.
                                      DATA SINGLE BIT ERROR

      MEAR              012AFC00
                                      PHYSICAL ADDRESS IN PA LATCH
                                      AT TIME OF ERROR = 012AFC00

      MSTAT1            64006006
                                      ANY REFILL
                                      C0 TAG MISS
                                      BLOCK HIT
                                      ABUS ADAPTER = 0.
                                      WORD COUNT = 0.
                                      CYCLE TYPE = CP REFILL
                                      DEST CP = OP FETCH
```

```
        MSTAT2          00040F00

                                    DIAGNOSTIC STATUS FROM SBIA
                                    _     RD COM/MSK <3:0> = F (X)
                                    _     RD DAT L/S <1:0> = 0 (X)
                                    PAMM DATA = ARRAY #4.,SLOT #5.

CORRECTED ERROR 3.

        MDECC           00261400

                                    SYNDROME = CORRECTED DATA BIT #1.
                                    DATA SINGLE BIT ERROR

        MEAR            01297400

                                    PHYSICAL ADDRESS IN PA LATCH
                                    AT TIME OF ERROR = 01297400

        MSTAT1          64006002

                                    ANY REFILL
                                    C0 TAG MISS
                                    BLOCK HIT
                                    ABUS ADAPTER = 0.
                                    WORD COUNT = 0.
                                    CYCLE TYPE = CP REFILL
                                    DEST CP = OP FETCH

        MSTAT2          00044F00

                                    DIAGNOSTIC STATUS FROM SBIA
                                    _     RD COM/MSK <3:0> = F (X)
                                    _     RD DAT L/S <1:0> = 0 (X)
                                    _     ABUS BAD DATA CODE
                                    PAMM DATA = ARRAY #4.,SLOT #5.

CORRECTED ERROR 4.

        MDECC           00261400

                                    SYNDROME = CORRECTED DATA BIT #1.
                                    DATA SINGLE BIT ERROR

        MEAR            01297400

                                    PHYSICAL ADDRESS IN PA LATCH
                                    AT TIME OF ERROR = 01297400

        MSTAT1          64006002

                                    ANY REFILL
                                    C0 TAG MISS
                                    BLOCK HIT
                                    ABUS ADAPTER = 0.
                                    WORD COUNT = 0.
                                    CYCLE TYPE = CP REFILL
                                    DEST CP = OP FETCH

        MSTAT2          00040F00

                                    DIAGNOSTIC STATUS FROM SBIA
                                    _     RD COM/MSK <3:0> = F (X)
                                    _     RD DAT L/S <1:0> = 0 (X)
                                    PAMM DATA = ARRAY #4.,SLOT #5.
```

Example 3: KA86 SBIA Error (Entry Type 013)

```
 V A X / V M S        SYSTEM ERROR REPORT      COMPILED  6-SEP-1985 12:55
                                                              PAGE   1.

 ***************************** ENTRY       1. *****************************
 ERROR SEQUENCE 66.                                LOGGED ON SID 0405F270

 SBIA ERROR   2-JUL-1985 19:17:12.92
              KA86  REV# 5.  SERIAL# 624.  MFG PLANT 15.

       ERROR PC          80008B1F
       ERROR PSL         00000000
                                        INTERRUPT PRIORITY LEVEL = 00.
                                        PREVIOUS MODE = KERNEL
                                        CURRENT MODE  = KERNEL

       IOA ADDRESS       80029200
       DMAI CMD/ADDRS    403C747E
                                        (* DATA NOT VALID *)
       DMAI ID           0000000E
                                        (* DATA NOT VALID *)
       DMAA CMD/ADDRS    18001800
                                        (* DATA NOT VALID *)
       DMAA ID           00000010
                                        (* DATA NOT VALID *)
       DMAB CMD/ADDRS    103C747F
                                        (* DATA NOT VALID *)
       DMAB ID           0000000E
                                        (* DATA NOT VALID *)
       DMAC CMD/ADDRS    B03E09FC
                                        (* DATA NOT VALID *)
       DMAC ID           0000000E
                                        (* DATA NOT VALID *)
       IOA DC            00000000
       IOA ES            1C000000
       IOA CS            EE000000
                                        CPU TR SELECT = 2.
                                        ENABLE SBI CYCLES IN
                                        ENABLE SBI CYCLES OUT
                                        MASTER INTERRUPT ENABLE
       IOA CF            01000010
                                        SOFTWARE REQUIRED SBI REV = 0.
                                        SBI
                                        16M OF MEMORY ADDRESSABLE (ABUS)
       SBIA FS           040F0000
                                        FAULT SILO LOCK
                                        SBI FAULT
                                        FAULT INTERRUPT ENABLE
                                        FAULT LATCH
                                        TRANSMITTER DURING FAULT
       SBIA SC           00000000
                                        COUNT FIELD = 0.
                                        COMPARE TAG = 0.
                                        COMPARE CMD/MSK = 0.
       SBIA MT           00000000
       SBIA ER           00000000
       SBIA TA           0802000E
                                        (* DATA NOT VALID *)
```

SBI SILO LOCKED, DETAILED SUMMARY

```
                        00000000
                        1C000000
                                        VALID READ DATA
                                        ID = 0.
                        00000002
                                        TR 1. ACTIVE
                        00000000
                        00000000
                        00000000
                        00000000
                        00000000
                        00000000
                        00000000
                        00000000
                        00000000
                        00000000
                        00000000
                        00000000
                        00000000
ADAPTER TR# 3.

      "DW" CSR          00000028
                                        ADAPTER IS UBA 0.
ADAPTER TR# 14.

      CNFGR             20180038
                                        ADAPTER IS "CI"
                                        READ DATA TIMEOUT
                                        COMMAND TRANSMIT TIMEOUT
                                        UNEXPECTED READ DATA FAULT
```

Example 4: KA86 Environmental Monitor (Entry Type 015)

```
V A X / V M S        SYSTEM ERROR REPORT      COMPILED  9-SEP-1985 08:30
                                                              PAGE    1.

******************************* ENTRY       1. *******************************
ERROR SEQUENCE 9413.                                  LOGGED ON SID 0405F09E

EMM EXCEPTION    9-JUL-1985 14:37:41.01
                 KA86  REV# 5.  SERIAL# 158.  MFG PLANT 15.

STATUS CHANGE IN T1 TEMPERATURE, THE TEMPERATURE IS NOW IN YELLOW ZONE


******************************* ENTRY       2. *******************************
ERROR SEQUENCE 9414.                                  LOGGED ON SID 0405F09E

EMM EXCEPTION    9-JUL-1985 14:37:44.43
                 KA86  REV# 5.  SERIAL# 158.  MFG PLANT 15.

STATUS CHANGE IN T1 TEMPERATURE, THE TEMPERATURE IS NOW NORMAL
```

Example 5: KA86 Processor Halt (Entry Type 016)

```
V A X / V M S        SYSTEM ERROR REPORT      COMPILED  9-SEP-1985 08:52
                                                              PAGE   1.

****************************** ENTRY       1. ******************************
ERROR SEQUENCE 1053.                                LOGGED ON SID 0405F270

KAF SNAPSHOT  12-JUL-1985 09:48:27.50
              KA86  REV# 5.  SERIAL# 624.  MFG PLANT 15.

SYS$SYSROOT:[SYSERR]ERRSNAP.LOG;25    12-JUL-1985 09:48:22.53


****************************** ENTRY       2. ******************************
ERROR SEQUENCE 1058.                                LOGGED ON SID 0405F270

KAF SNAPSHOT  12-JUL-1985 09:48:38.06
              KA86  REV# 5.  SERIAL# 624.  MFG PLANT 15.

SYS$SYSROOT:[SYSERR]ERRSNAP.LOG;26    12-JUL-1985 09:48:33.68


****************************** ENTRY       3. ******************************
ERROR SEQUENCE 1116.                                LOGGED ON SID 0405F270

KAF SNAPSHOT  12-JUL-1985 15:47:01.59
              KA86  REV# 5.  SERIAL# 624.  MFG PLANT 15.

SYS$SYSROOT:[SYSERR]ERRSNAP.LOG;27    12-JUL-1985 15:46:55.83
```

Example 6: KA86 Console Reboot (Entry Type 017)

```
 V A X / V M S        SYSTEM ERROR REPORT      COMPILED  6-SEP-1985 12:58
                                                          PAGE   1.

 ****************************** ENTRY       1. ******************************
 ERROR SEQUENCE 82.                                LOGGED ON SID 04FFFFFF

 CONSOLE REBOOT SUCCESS   4-FEB-1985 16:30:46.62
                          KA86  REV# 255.  SERIAL# 4095.  MFG PLANT 15.
```

Example 7: Cold Start (Entry Type 032)

```
 V A X / V M S        SYSTEM ERROR REPORT      COMPILED  9-SEP-1985 09:00
                                                          PAGE   1.

 ****************************** ENTRY       1. ******************************
 ERROR SEQUENCE 0.                                 LOGGED ON SID 0405F24F

 SYSTEM START-UP  10-JUL-1985 15:56:51.62
                  KA86  REV# 5.  SERIAL# 591.  MFG PLANT 15.

 TIME OF DAY CLOCK     72306E69
```

Example 8: Crash Re-start (Entry Type 037)

```
 V A X / V M S        SYSTEM ERROR REPORT      COMPILED  9-SEP-1985 09:08
                                                          PAGE    1.

 ***************************** ENTRY       1. *****************************
 ERROR SEQUENCE 6906.                                LOGGED ON SID 0405F09E

 FATAL BUGCHECK  24-JUN-1985 21:15:03.51
                 KA86  REV# 5.  SERIAL# 158.  MFG PLANT 15.

 OPERATOR, Operator requested system shutdown



 ***************************** ENTRY       2. *****************************
 ERROR SEQUENCE 8075.                                LOGGED ON SID 0405F09E

 FATAL BUGCHECK   1-JUL-1985 12:34:20.33
                  KA86  REV# 5.  SERIAL# 158.  MFG PLANT 15.

 MACHINECHK, Machine check while in kernel mode

       PROCESS NAME     .A473373:......

       PROCESS ID       00070122

       ERROR PC         80245D62

       ERROR PSL        045F0008
                                        N-BIT
                                        INTERRUPT PRIORITY LEVEL = 31.
                                        PREVIOUS MODE = EXECUTIVE
                                        CURRENT MODE  = KERNEL
                                        INTERRUPT STACK

 STACK POINTERS

 KSP 7FFE7E00  ESP 7FFE9D80  SSP 7FFED04E  USP 7FF6D91C  ISP 806B0F50

 GENERAL REGISTERS

 R0  00000000  R1  00001F73  R2  000000AA  R3  7FFBA207  R4  7FFBA207
 R5  7FFBA21C  R6  7FFBA668  R7  7FFB9D40  R8  7FFBA223  R9  00000006
 R10 7FFBA205  R11 7FFBA2CD  AP  00000003  FP  7FFE9DE4  SP  806B0F94


 SYSTEM REGISTERS

       P0BR             80A5C200
                                        P0 PTE BASE (VIRT ADDR)
       P0LR             000002DB
                                        TOTAL P0 PAGES
       P1BR             8027F200
                                        P1 PTE BASE (VIRT ADDR)
       P1LR             001FFB5B
                                        TOTAL NON-EXISTENT P1 PAGES
```

```
SBR             00FC3400        SYSTEM PTE BASE (PHY ADDR)

SLR             0000F300        TOTAL PAGES 'SYSTEM' VIRT MEM

PCBB            00423878        PCB BASE (PHY ADDR)

SCBB            00FBF600        SCB BASE (PHY ADDR)

ASTLVL          00000004        NO AST'S PENDING

SISR            00000000        INTERRUPT REQUEST ACTIVE =  0.

ICCS            800000C1        RUN
                                INTERRUPT ENABLE
                                INTERRUPT
                                ERROR

ICR             FFFFDA62        INTERVAL COUNT REGISTER

TODR            6D7B5185
```

Example 9: System Bugcheck (Entry Type 040)

```
 V A X / V M S        SYSTEM ERROR REPORT      COMPILED  9-SEP-1985 09:11
                                                              PAGE   1.

 ***************************** ENTRY       1. *****************************
 ERROR SEQUENCE 0.                                  LOGGED ON SID 0405F09E

 NON-FATAL BUGCHECK   4-JAN-1978 09:54:51.52
                      KA86  REV# 5.  SERIAL# 158.  MFG PLANT 15.

 UNXINTEXC, Unexpected interrupt or exception

       PROCESS NAME    .NULL.........

       PROCESS ID      00010000

       ERROR PC        80004680

       ERROR PSL       04170000
                                    INTERRUPT PRIORITY LEVEL = 23.
                                    PREVIOUS MODE = KERNEL
                                    CURRENT MODE  = KERNEL
                                    INTERRUPT STACK

 STACK POINTERS

 KSP 00000100  ESP 00000100  SSP 00000100  USP 00000100  ISP 806B0FAC

 GENERAL REGISTERS

 R0  00006E2A  R1  00006E29  R2  00000000  R3  801BEE1D  R4  00000149
 R5  801BEE11  R6  801BEDD8  R7  805A86C0  R8  805A8A80  R9  00000000
 R10 00000000  R11 800036B0  AP  FFFFFFFF  FP  A0000000  SP  806B0FF0
```

Example 10: Device Timeout (Entry Type 096)

```
V A X / V M S        SYSTEM ERROR REPORT      COMPILED  9-SEP-1985 09:16
                                                              PAGE   1.

****************************** ENTRY      1. ******************************
ERROR SEQUENCE 334.                                  LOGGED ON SID 04FFFFFF

"UNKNOWN DEVICE" ENTRY  25-JAN-1985 20:06:38.09
                        KA86  REV# 255.  SERIAL# 4095.  MFG PLANT 15.

ERROR LOG RECORD

      ERF$L_SID         04FFFFFF
                                      SYSTEM ID REGISTER
      ERL$W_ENTRY           0060
                                      ERROR ENTRY TYPE
      EXE$GQ_SYSTIME    93C156A0
                        008D7A56      64 BIT TIME WHEN ERROR LOGGED
      ERL$GL_SEQUENCE       014E
                                      UNIQUE ERROR SEQUENCE = 334.
      UCB$B_ERTCNT            00
                                      REMAINING RETRIES = 0.
      UCB$B_ERTMAX            00
                                      MAXIMUM RETRIES = 0.
      IRP$Q_IOSB        0000022C
                        00000000      FINAL IOSB
      UCB$W_STS             0150
                                      DEVICE STATUS
      UCB$B_DEVCLASS          20
                                      DEVICE CLASS = 32.
      UCB$B_DEVTYPE           00
                                      DEVICE TYPE = 0.
      IRP$L_PID         000100F7
                                      REQUESTING PROCESS ID
      IRP$W_BOFF            0000
                                      TRANSFER BYTE OFFSET = 0.
      IRP$W_BCNT            0000
                                      TRANSFER BYTE COUNT = 0.
      UCB$L_MEDIA       800029C0
                                      DEVICE DEPENDANT PHYSICAL ADDRESS
      UCB$W_UNIT            0000
                                      PHYSICAL UNIT NUMBER = 0.
      UCB$W_ERRCNT          0001
                                      UNIT ERROR COUNT = 1.
      UCB$L_OPCNT       00001B22
                                      UNIT OPERATION COUNT = 6946.
      ORB$L_OWNER       00000000
                                      OWNER UIC = [000,000]
      UCB$L_DEVCHAR     0C402000
                                      DEVICE CHARACTERISTICS
      UCB$B_SLAVE             00
                                      DEVICE SLAVE CONTROLLER = 0.
      IRP$W_FUNC            0020
                                      QIO FUNCTION CODE
```

```
DDB$T_NAME      3031300A
                24325035
                00415358
                00000000
                                        /.0105P2$XSA...../

LONGWORD 1.     00000009

LONGWORD 2.     00004091

LONGWORD 3.     00000000

LONGWORD 4.     00000001

LONGWORD 5.     00000100

LONGWORD 6.     00000000

LONGWORD 7.     00000005

LONGWORD 8.     00000C18

LONGWORD 9.     0000401F

LONGWORD 10.    0000002F
```

Example 11: Asynchronous Device Attention (Entry Type 098)

```
V A X / V M S        SYSTEM ERROR REPORT      COMPILED  9-SEP-1985 09:22
                                                          PAGE   1.

******************************* ENTRY      1. *******************************
ERROR SEQUENCE 37.                                 LOGGED ON SID 0405F270

DEVICE ATTENTION  24-JUN-1985 09:10:59.99
                  KA86  REV# 5.  SERIAL# 624.  MFG PLANT 15.

CI SUB-SYSTEM, _F$PAA0: - PORT ERROR BIT(S) SET

      PORT WILL BE RESTARTED, 50. OF 50. RETRIES REMAINING

      CNFGR            00100038
                                        ADAPTER IS "CI"
                                        COMMAND TRANSMIT TIMEOUT
      PMCSR            0000004C
                                        MAINTENANCE INTERRUPT ENABLE
                                        MAINTENANCE INTERRUPT FLAG
                                        PROGRAMMABLE STARTING ADDRESS
      PSR              00000001
                                        RESPONSE QUEUE AVAILABLE
      PFAR             80F89DBC
      PESR             00000000
      PPR              03F80007

      UCB$B_ERTCNT          32
                                        50. RETRIES REMAINING
      UCB$B_ERTMAX          32
                                        50. RETRIES ALLOWABLE
      UCB$L_CHAR       0C450000
                                        SHARABLE
                                        AVAILABLE
                                        ERROR LOGGING
                                        CAPABLE OF INPUT
                                        CAPABLE OF OUTPUT
      UCB$W_STS             0810
                                        ONLINE
                                        SOFTWARE VALID
      UCB$W_ERRCNT          0001
                                        1. ERRORS THIS UNIT
```

Example 12: Unknown Entry (Entry Type 273)

```
V A X / V M S        SYSTEM ERROR REPORT      COMPILED  6-SEP-1985 13:44
                                                              PAGE   1.

****************************** ENTRY       1. ******************************
ERROR SEQUENCE 83.                                  LOGGED ON SID 04FFFFFF

"UNKNOWN ENTRY"    4-FEB-1985 16:32:11.75
                   KA86  REV# 255.  SERIAL# 4095.  MFG PLANT 15.

ERROR LOG RECORD

      ERF$L_SID         04FFFFFF
                                        SYSTEM ID REGISTER

      ERL$W_ENTRY           0111
                                        ERROR ENTRY TYPE

      EXE$GQ_SYSTIME    46F32860
                        008D8214        64 BIT TIME WHEN ERROR LOGGED
      ERL$GL_SEQUENCE       0053
                                        UNIQUE ERROR SEQUENCE = 83.
      LONGWORD 1.       0000005A
                                        /Z.../
```

Example 13: The following printout is the product of the /LOG qualifier. Refer to the /REJECT qualifier for an explaination of rejected entries.

```
%ERF-I-INPUT, SYS$SYSTEM:ERRLOG.SYS, 5 selected, 12 rejected
```

Example 14: The following printout is the product of the /REGISTER_DUMP qualifier. The cryptic format (shown below) can be used to identify control and status bits common to multiple entries.

```
 V A X / V M S          SYSTEM ERROR REPORT      COMPILED 11-SEP-1985 11:33
                                                                 PAGE   1.

      CSR       CR        SR        DCR       FMER      FUBAR

      00000028 0000007C 00000001 08000028 00000000 0000F86D
      00000028 0000007C 00000001 08000028 00000000 0000F86D
      00000028 0000007C 00000001 08000028 00000000 0000F86D
      00000028 0000007C 00000001 08000028 00000000 0000F86D
      00000028 0000007C 00000001 08000028 00000000 0000F86D
```

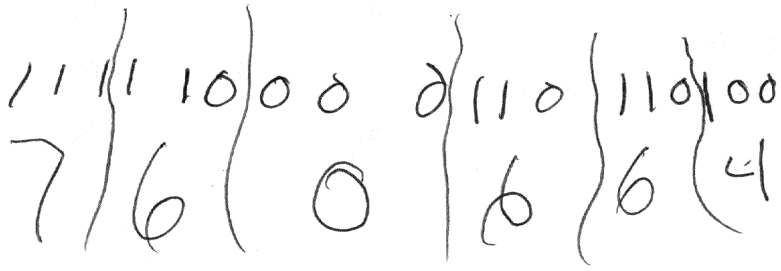
Example 15: The following printout is the product of the /STATISTICS qualifier.

```
 V A X / V M S          SYSTEM ERROR REPORT      COMPILED 11-SEP-1985 09:13
                                                                 PAGE   1.

 PROGRAM RUNTIME STATISTICS

      TIMES IN SECONDS          PAGE     DIRECT     BUFFERED
          CPU  ELAPSED          FAULTS      I/O          I/O

          1.3      4.4             150       17            7
```



Example 16: The following printout is the product of the /SUMMARIZE qualifier. Specifically this is a sample of the Device Summary Report.

V A X / V M S SYSTEM ERROR REPORT COMPILED 13-SEP-1985 09:42
PAGE 1.

DEVICE ROLLUP LOGGED BY SID 0405F270

DEVICE	ERROR BITS SET		QIO TIMEOUT		ERRORS THIS SESSION	QIOS THIS SESSION
	[HARD]	[SOFT]	[HARD]	[SOFT]		
_HSC003$DUA0:	0.	3.	0.	0.	0.	0.
_HSC003$DUA1:	0.	7.	0.	0.	0.	0.
_HSC003$DUA2:	0.	1.	0.	0.	0.	0.
_HSC003$DUA4:	0.	4.	0.	0.	0.	0.
_HSC003$DUA5:	0.	22.	0.	0.	31.	23152.
_HSC002$DUA1:	0.	8.	0.	0.	4.	10321.
_HSC002$DUA2:	0.	2.	0.	0.	0.	0.
_HSC002$DUA3:	0.	1.	0.	0.	0.	0.
_HSC002$DUA4:	0.	1.	0.	0.	0.	0.
_HSC002$DUA5:	0.	2.	0.	0.	1.	2563.
_HSC002$DUA8:	0.	7.	0.	0.	0.	0.
_HSC002$DUA9:	0.	11.	0.	0.	0.	0.
_HSC002$DUA10:	0.	3.	0.	0.	0.	0.
_HSC002$DUA11:	0.	2.	0.	0.	0.	0.
_HSC002$DUA12:	0.	5.	0.	0.	0.	0.
_HSC002$DUA13:	0.	7.	0.	0.	0.	0.
_F$PAA0:	8.	6.	0.	0.	1.	0.
_F$LCA0:	4.	0.	0.	0.	1.	0.

Example 17: The following printout is the product of the /SUMMARIZE qualifier. Specifically this is a sample of the Volume Summary Report.

```
V A X / V M S        SYSTEM ERROR REPORT      COMPILED 13-SEP-1985 09:42
                                                          PAGE   2.


VOLUME LABEL(S) LOGGED BY SID 0405F270

                                    QIO(S)     ERROR(S)  MOUNT(S)

      LABEL --

      _CSA1:                         273.          0.        19.

      LABEL -- Exchange

      _CSA1:                          28.          0.         2.

      LABEL -- SCRATCH

      _DUA6:                       17547.          0.         1.

      LABEL -- VAX console

      _CSA1:                          96.          0.        24.
```

Example 18: The following printout is the product of the /SUMMARIZE qualifier. Specifically this is a sample of the Entry Summary Report.

```
V A X / V M S          SYSTEM ERROR REPORT      COMPILED 13-SEP-1985 09:42
                                                              PAGE   3.


 SUMMARY OF ALL ENTRIES LOGGED BY SID 0405F270

       MACHINE CHECK                          5.
       UBA INTERRUPT                          3.
       SBIA ERROR                             1.
       CPU ERROR HALT                        24.
       SYSTEM START-UP                       34.
       ERRLOG.SYS CREATED                     1.
       FATAL BUGCHECK                        21.
       TIME-STAMP                           106.
       VOLUME MOUNT                         608.
       VOLUME DISMOUNT                      215.
       DEVICE ATTENTION                       8.
       ERL$LOGSTATUS                         20.
       ERL$LOGMESSAGE                        92.
       ERL$LOGMSCP                            1.
       UNKNOWN ENTRY TYPE                     4.

       DATE OF EARLIEST ENTRY        16-JUN-1985 23:12:17.38
       DATE OF LATEST ENTRY           9-JUL-1985 22:28:29.81
```

Example 19: The following printout is the product of the /SUMMARIZE qualifier. Specifically this is a sample of the Histogram Summary Report.

```
V A X / V M S        SYSTEM ERROR REPORT      COMPILED 13-SEP-1985 09:42
                                                          PAGE   4.


PROCESSED ENTRIES HOUR-OF-DAY HISTOGRAM LOGGED BY SID 0405F270

      00:00      9. *********
      01:00     34. **********************************
      02:00     23. ***********************
      03:00     51. ***************************************************
      04:00     46. ***********************************************
      05:00      4. ****
      06:00     10. **********
      07:00     32. ********************************
      08:00     34. **********************************
      09:00    123. ************************************************************
      10:00     36. ************************************
      11:00     28. ****************************
      12:00     55. ******************************************************
      13:00     54. *****************************************************
      14:00     65. ***************************************************************
      15:00     63. *************************************************************
      16:00     71. ***************************************************************
      17:00     60. ************************************************************
      18:00     41. ****************************************
      19:00    110. ********************************************************************
      20:00     98. ***************************************************************
      21:00     23. ***********************
      22:00     67. ******************************************************************
      23:00      6. ******
```

APPENDIX A

ERROR HANDLING MICROCODE (EHM) FLOW CHART

This Appendix contains a set of flow charts that describe how the Error Handling Microcode (EHM) captures the state of the CPU, clears the error condition, rolls back the RLOG and PCs, and passes control to the VMS Machine Check Handler (MCHK). The notes following the flow charts explain what is happening at each block in the flow.

The flow charts and notes represent the first revision of the EHM, first shipped with VAX 8600/8650 Console RL02 pack Rev. 2.1.

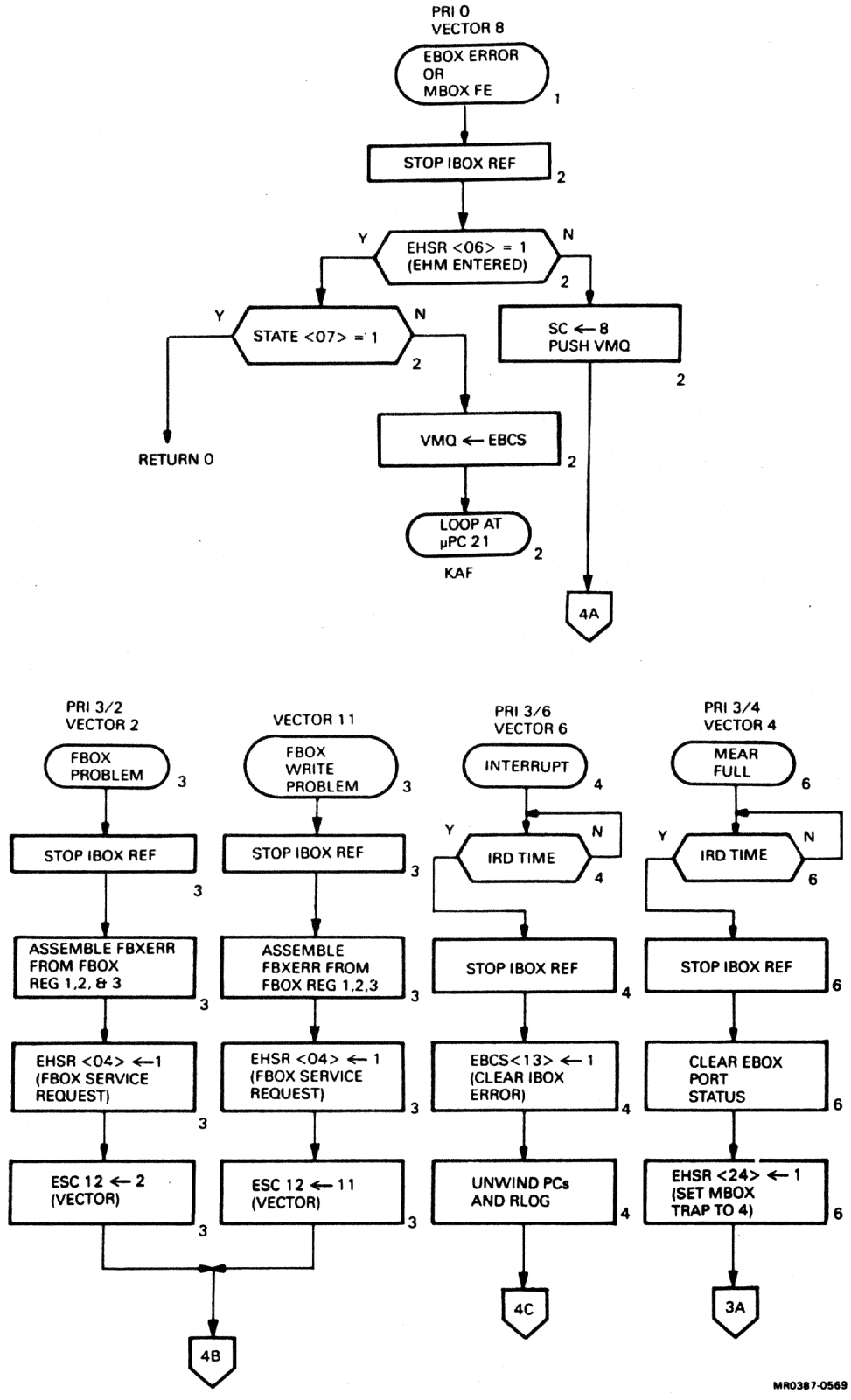


Figure A-1 Error Handling Microcode Flow Chart (Part 1 of 20)

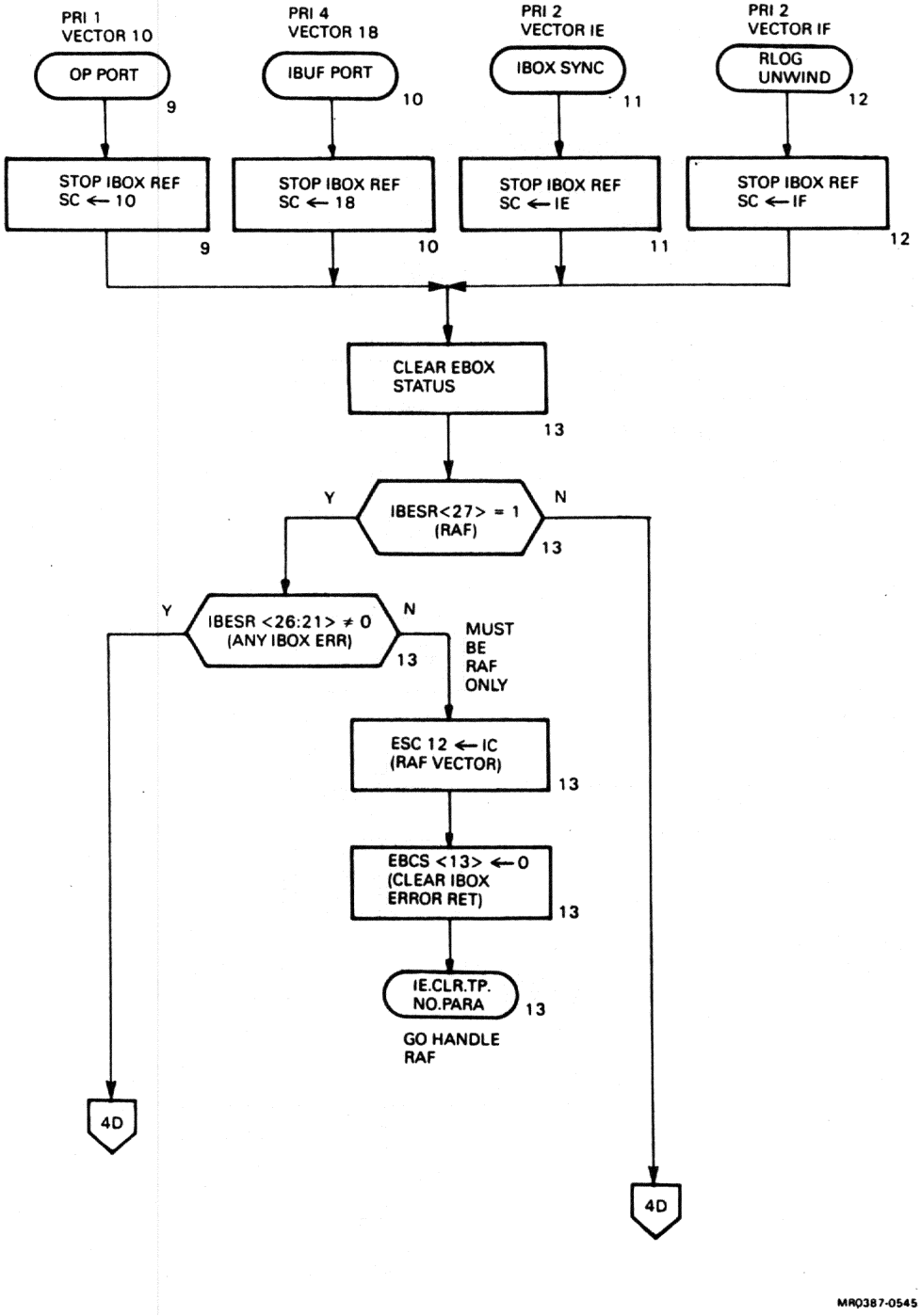


Figure A-1 Error Handling Microcode Flow Chart (Part 2 of 20)

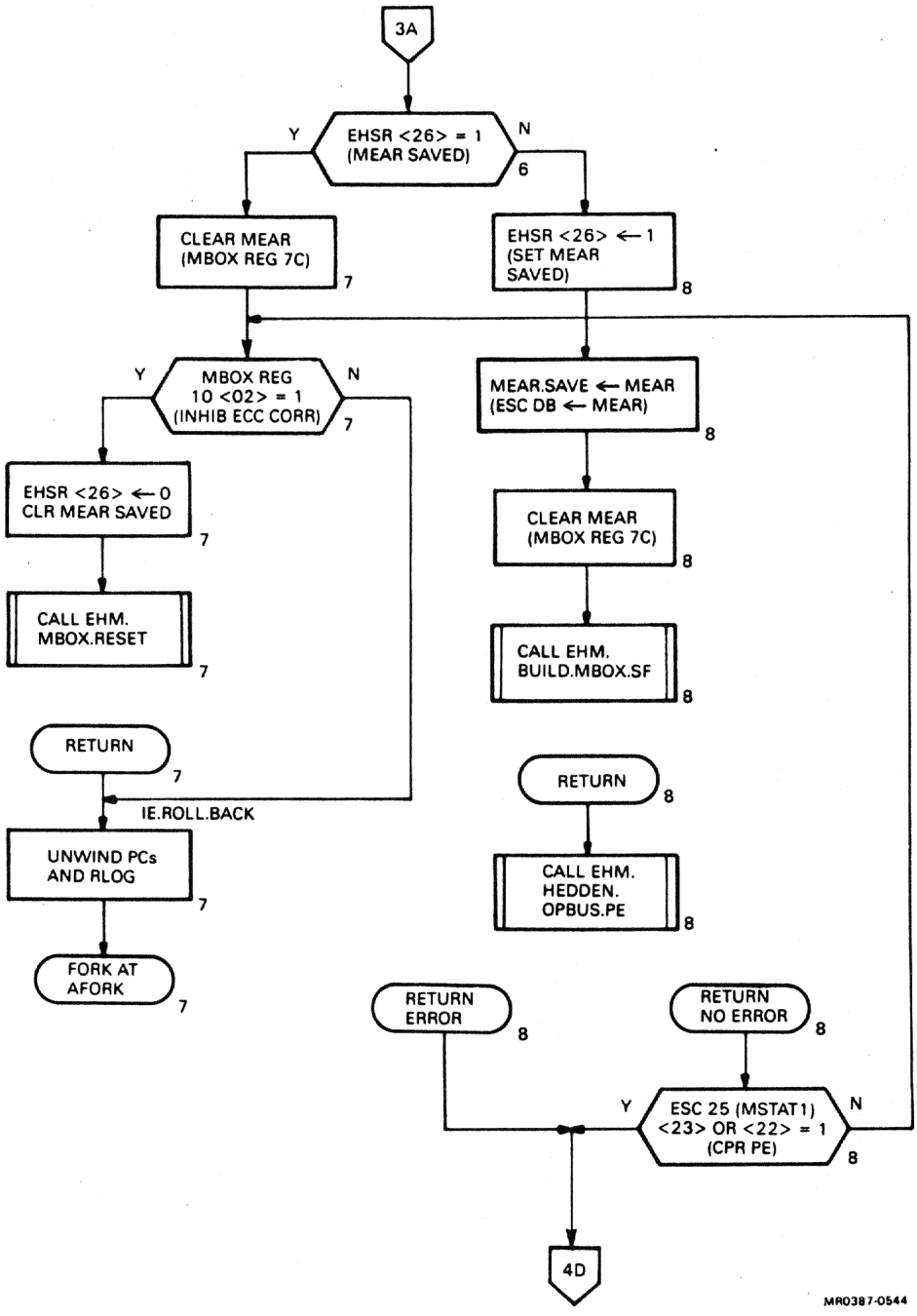


Figure A-1 Error Handling Microcode Flow Chart (Part 3 of 20)

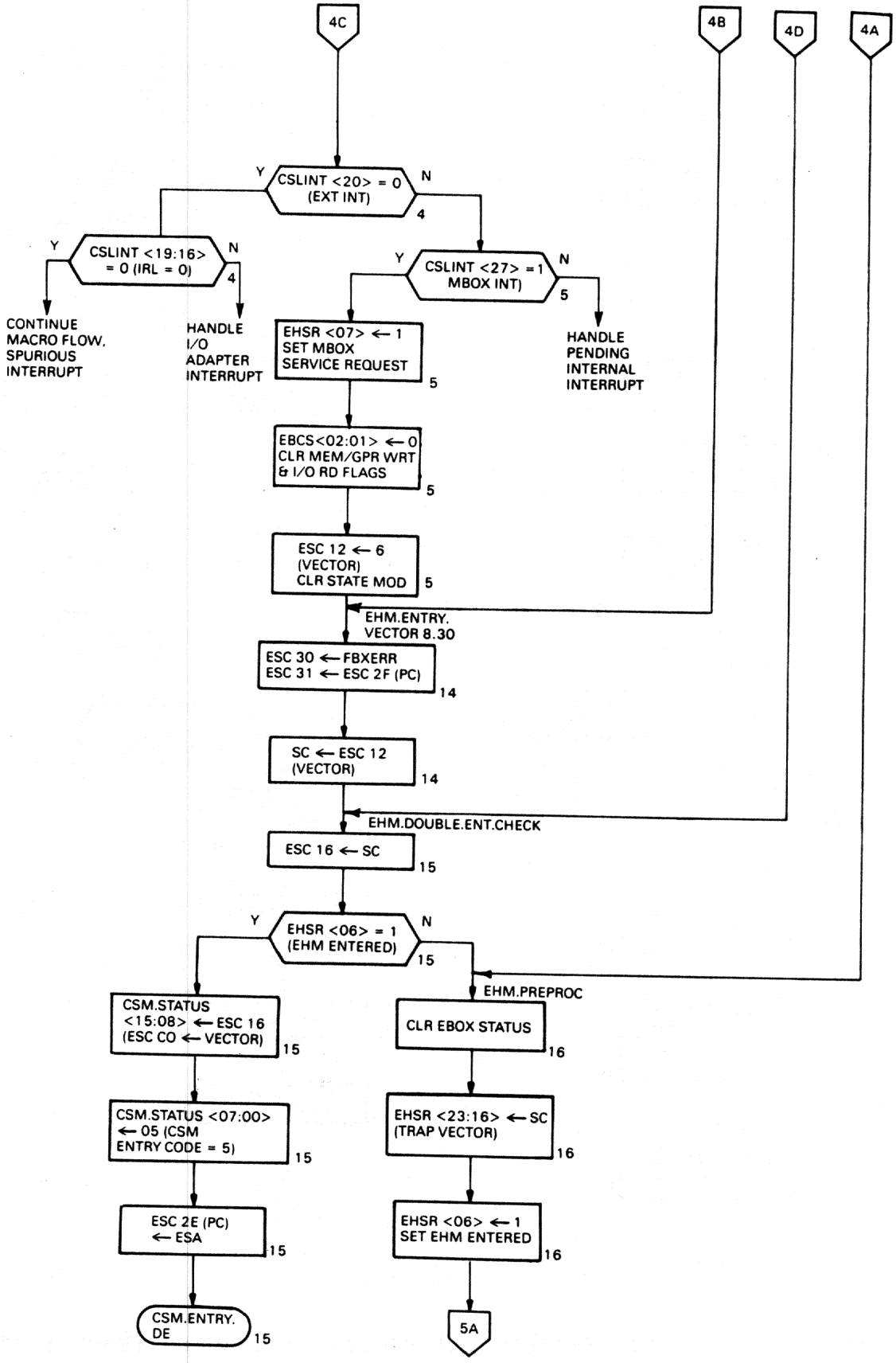


Figure A-1 Error Handling Microcode Flow Chart (Part 4 of 20)

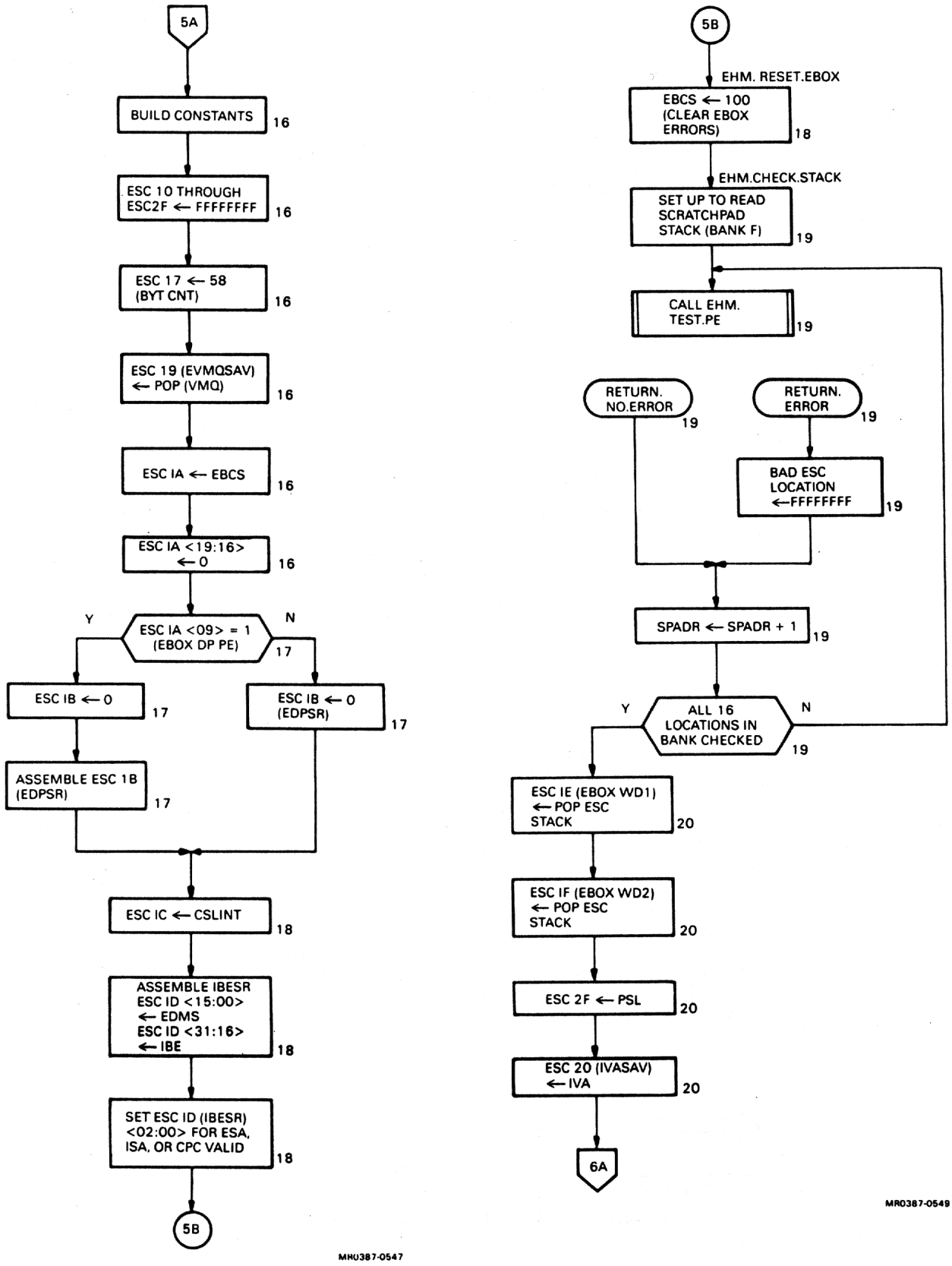


Figure A-1 Error Handling Microcode Flow Chart (Part 5 of 20)

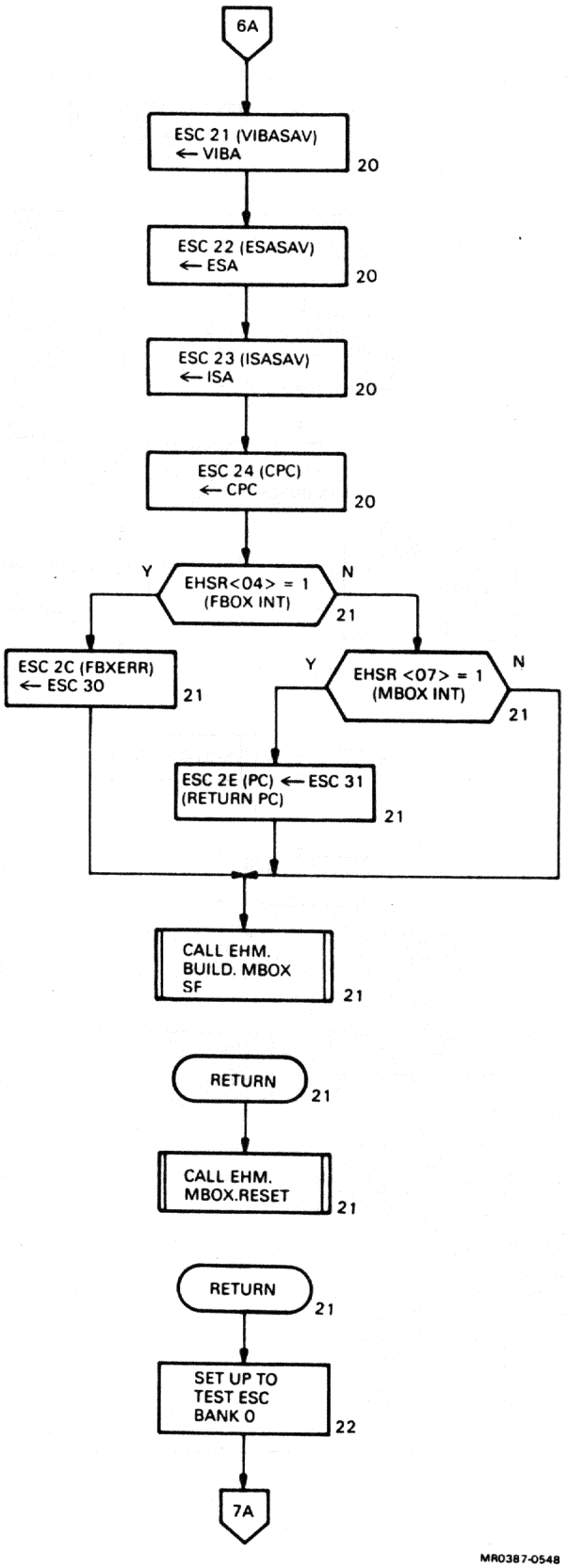


Figure A-1 Error Handling Microcode Flow Chart (Part 6 of 20)

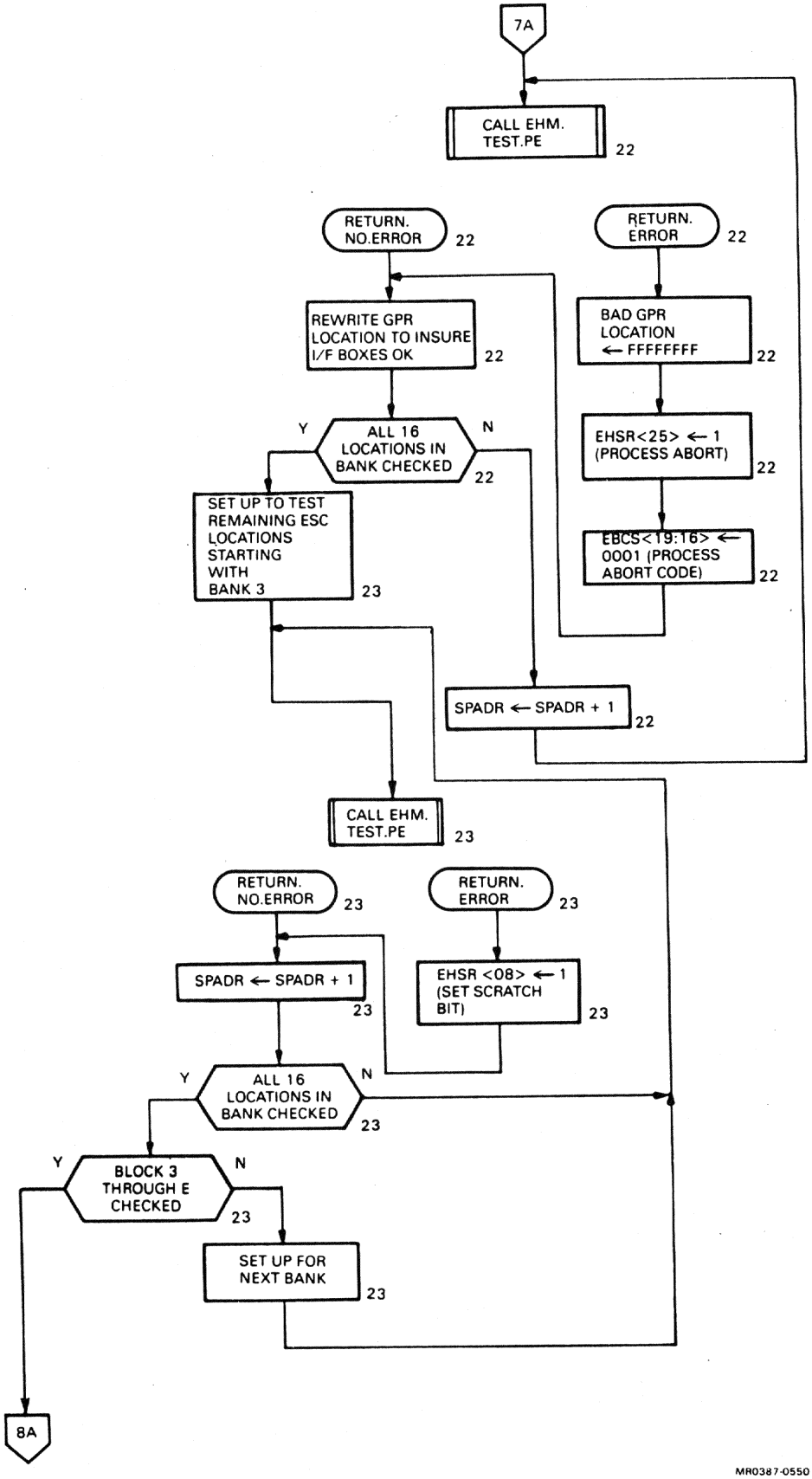


Figure A-1 Error Handling Microcode Flow Chart (Part 7 of 20)

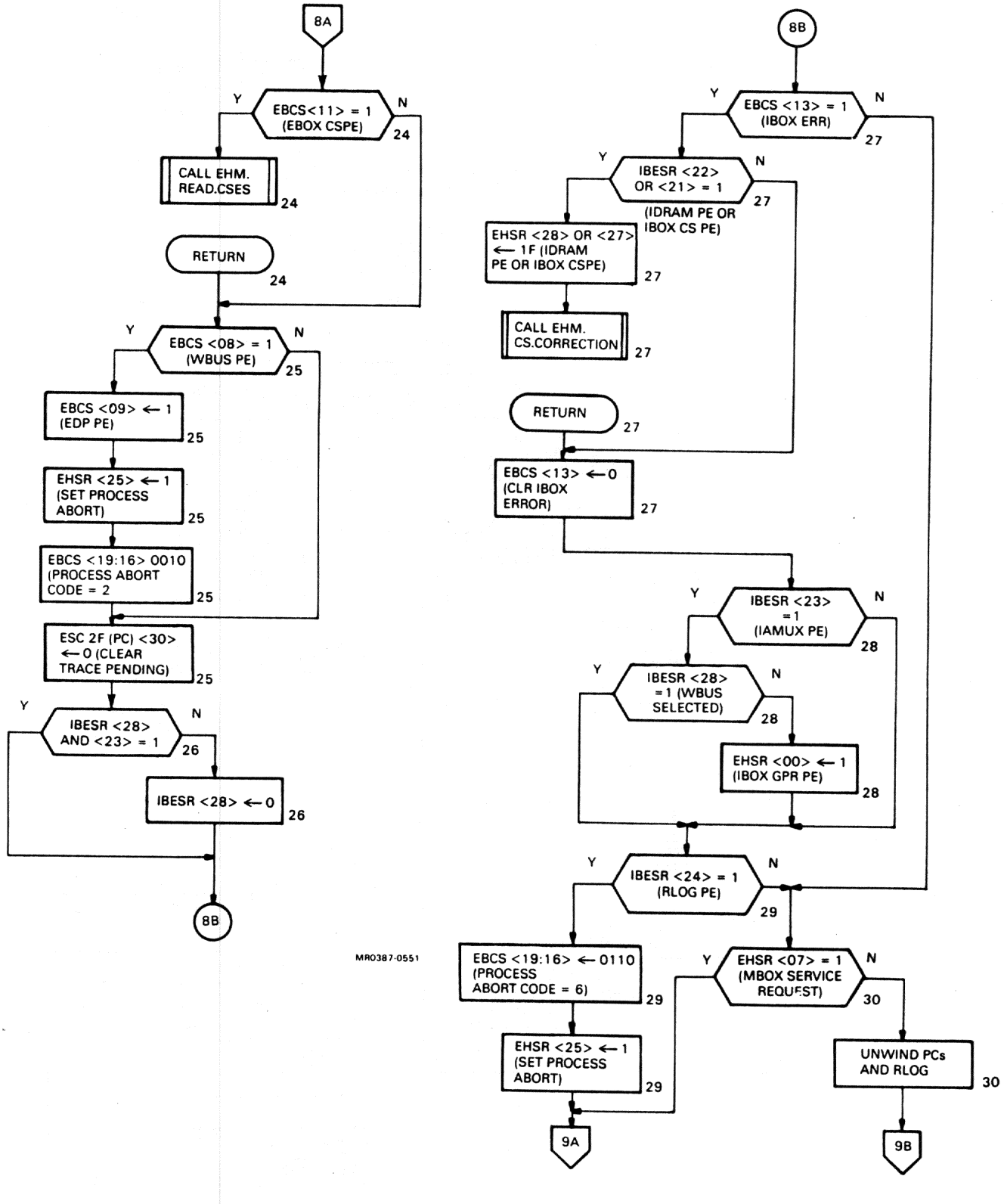


Figure A-1 Error Handling Microcode Flow Chart (Part 8 of 20)

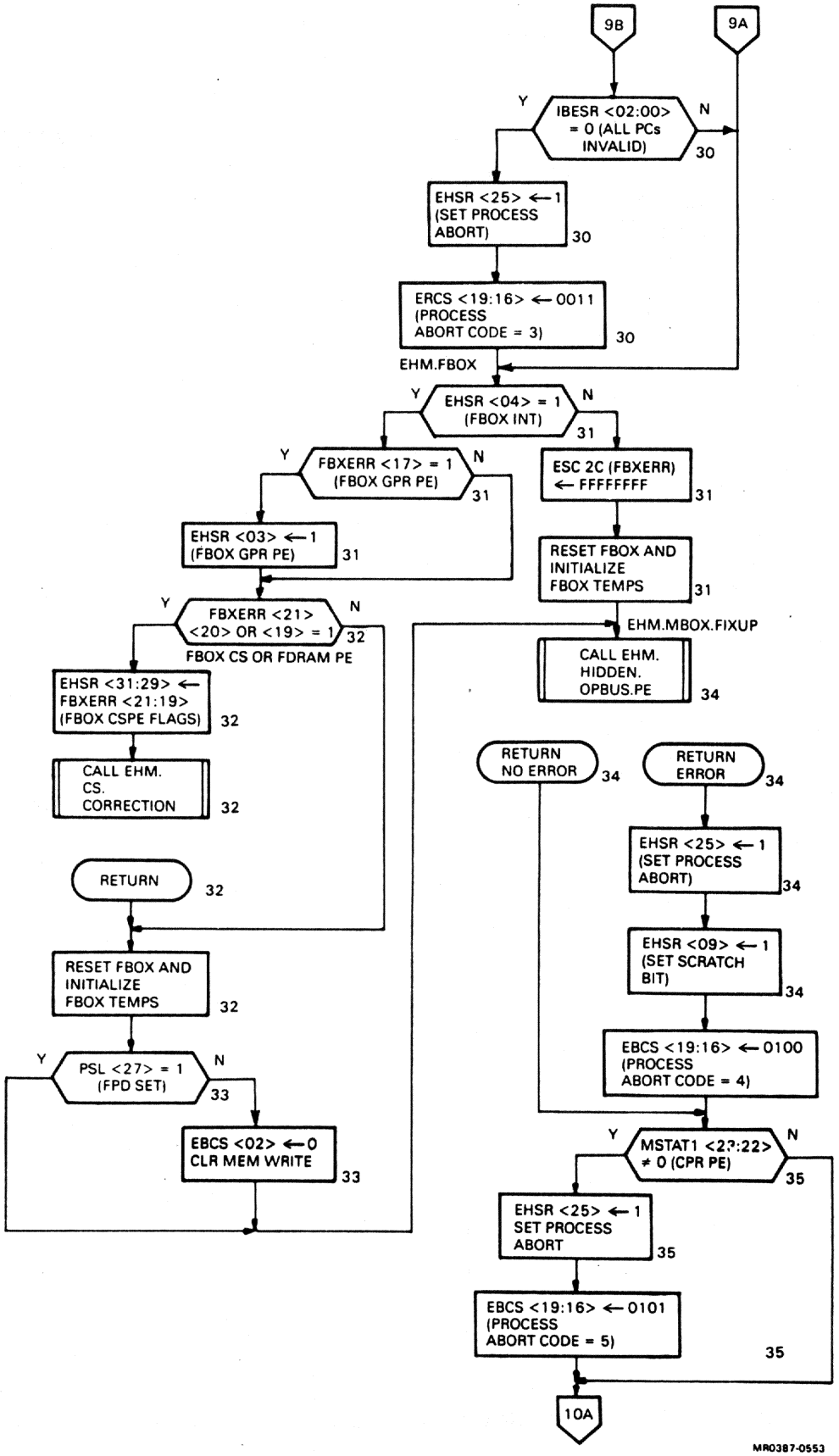


Figure A-1 Error Handling Microcode Flow Chart (Part 9 of 20)

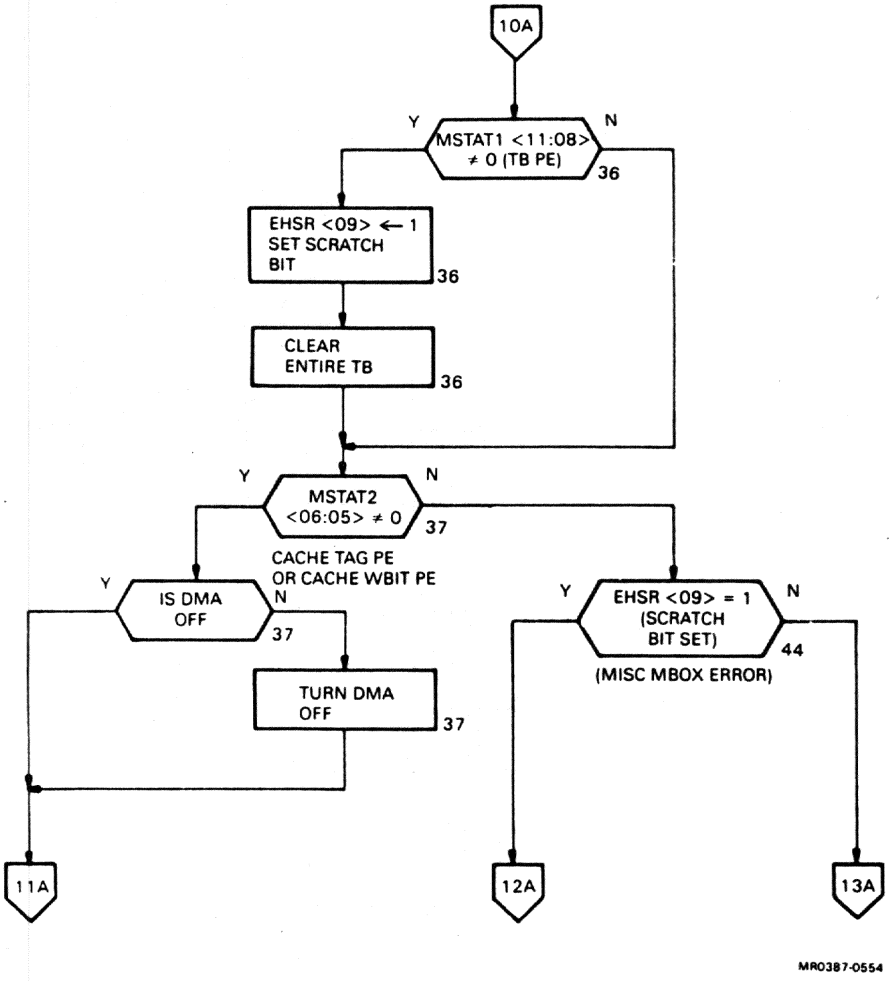


Figure A-1 Error Handling Microcode Flow Chart (Part 10 of 20)

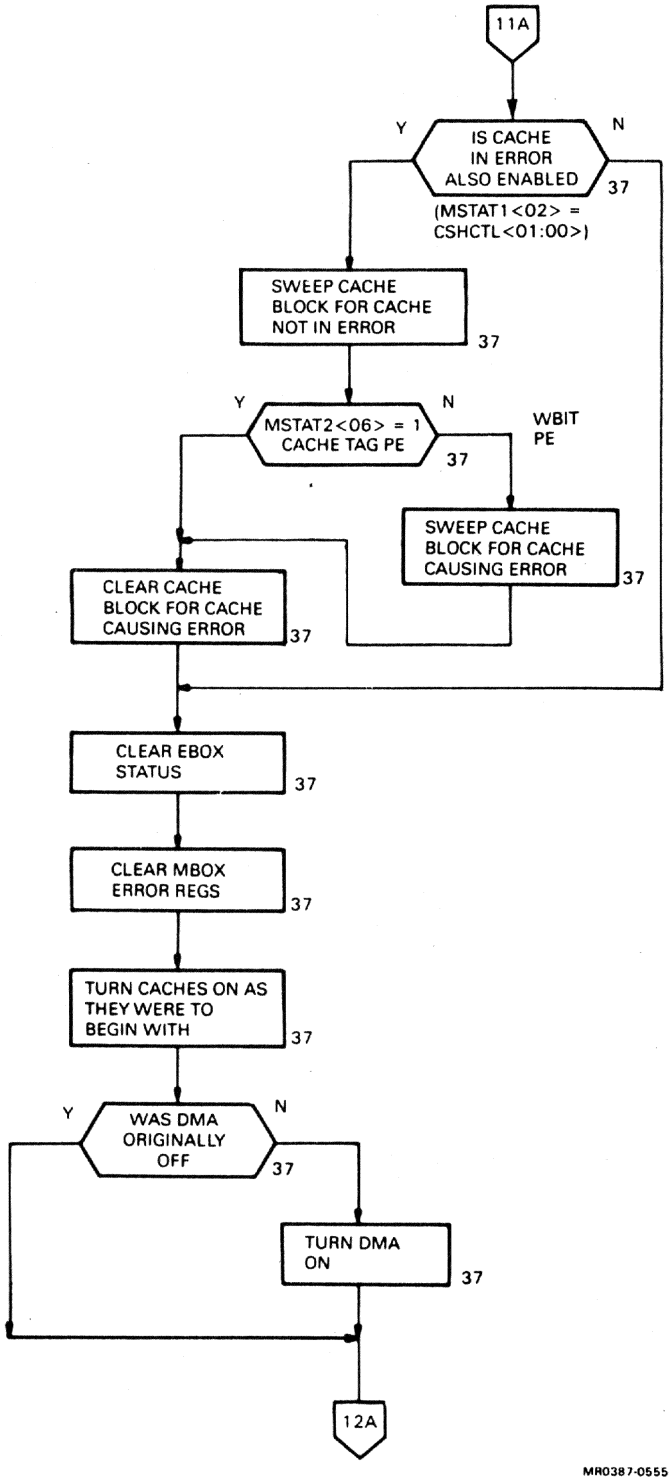


Figure A-1 Error Handling Microcode Flow Chart (Part 11 of 20)

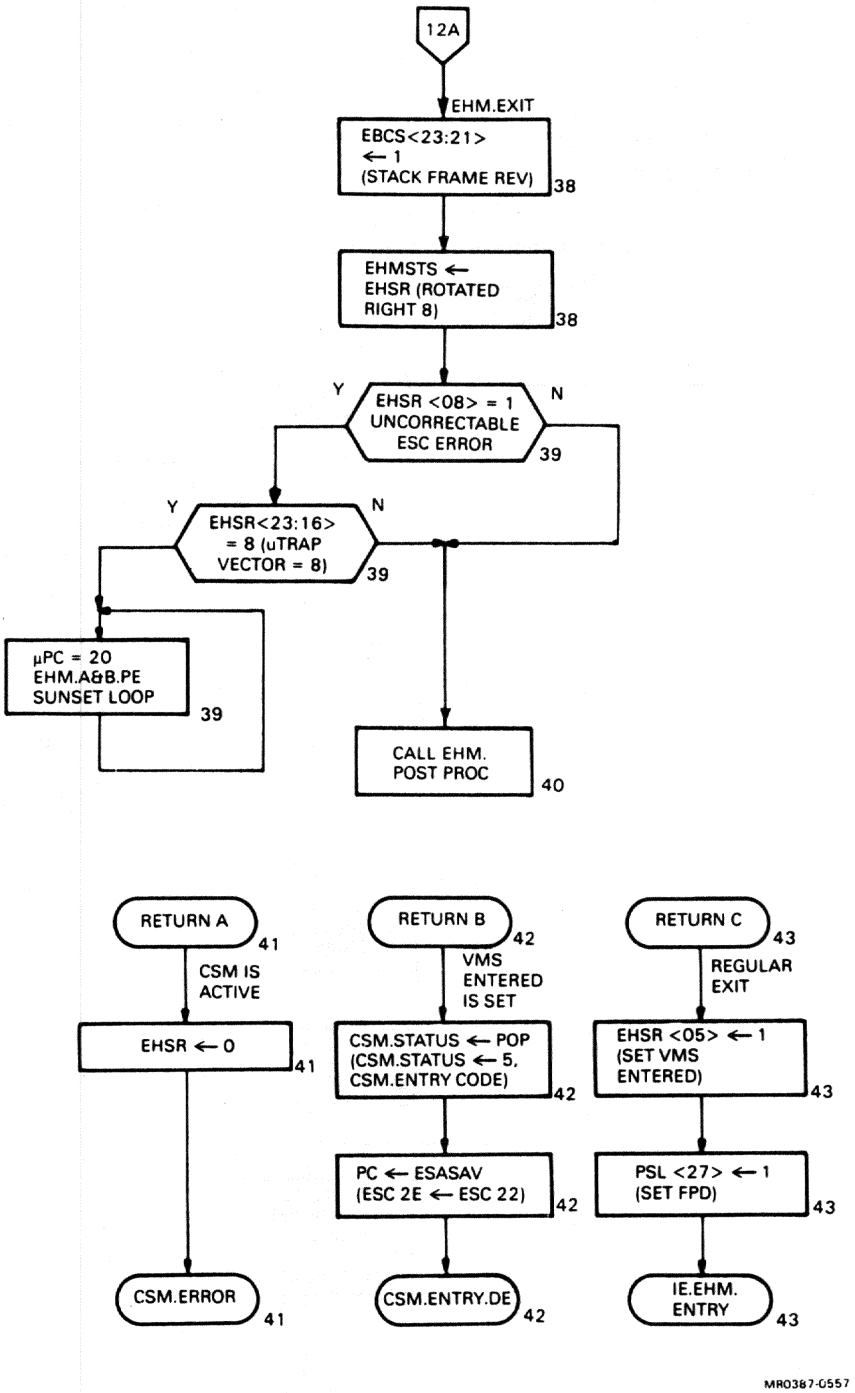


Figure A-1 Error Handling Microcode Flow Chart (Part 12 of 20)

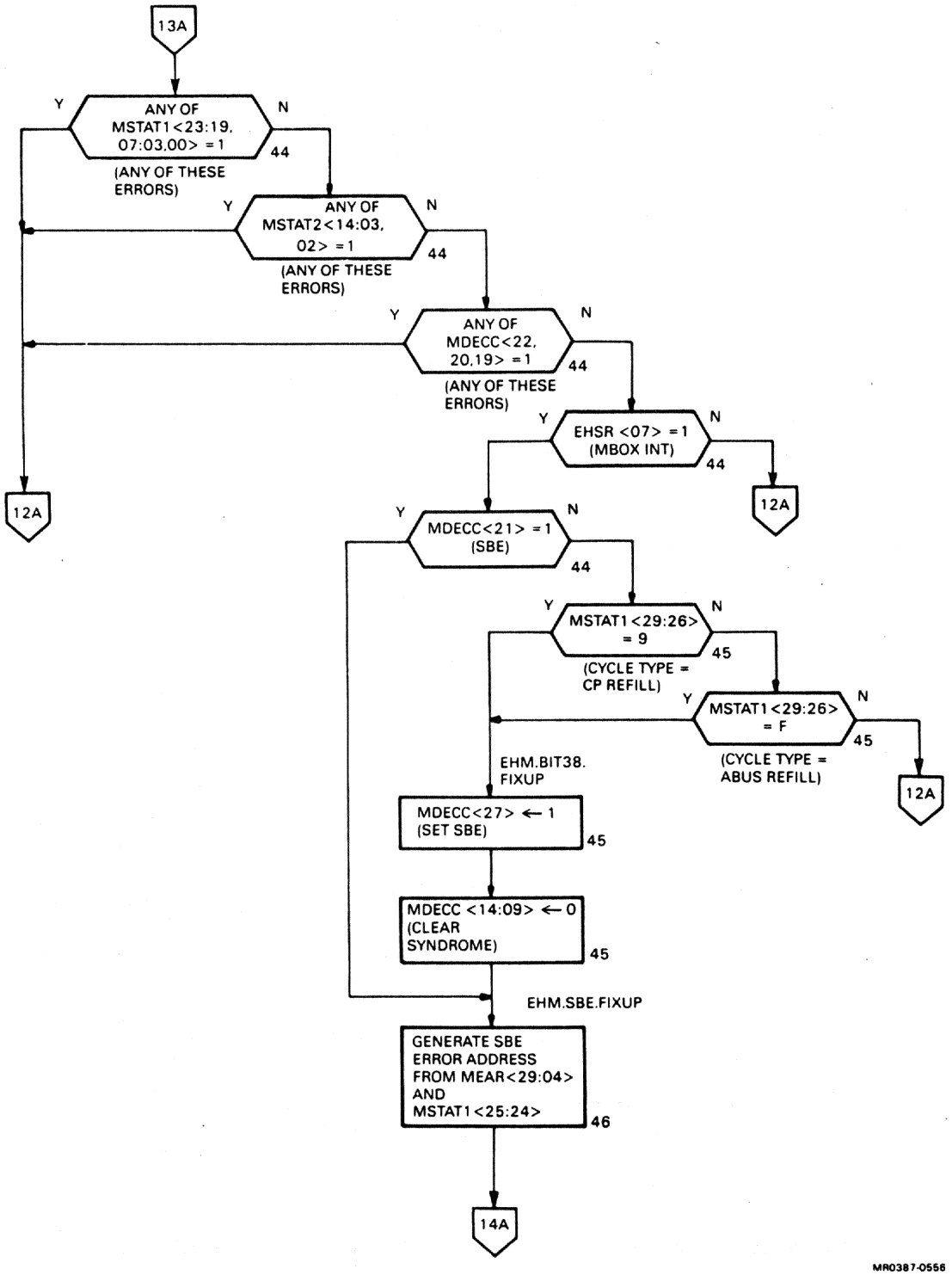


Figure A-1 Error Handling Microcode Flow Chart (Part 13 of 20)

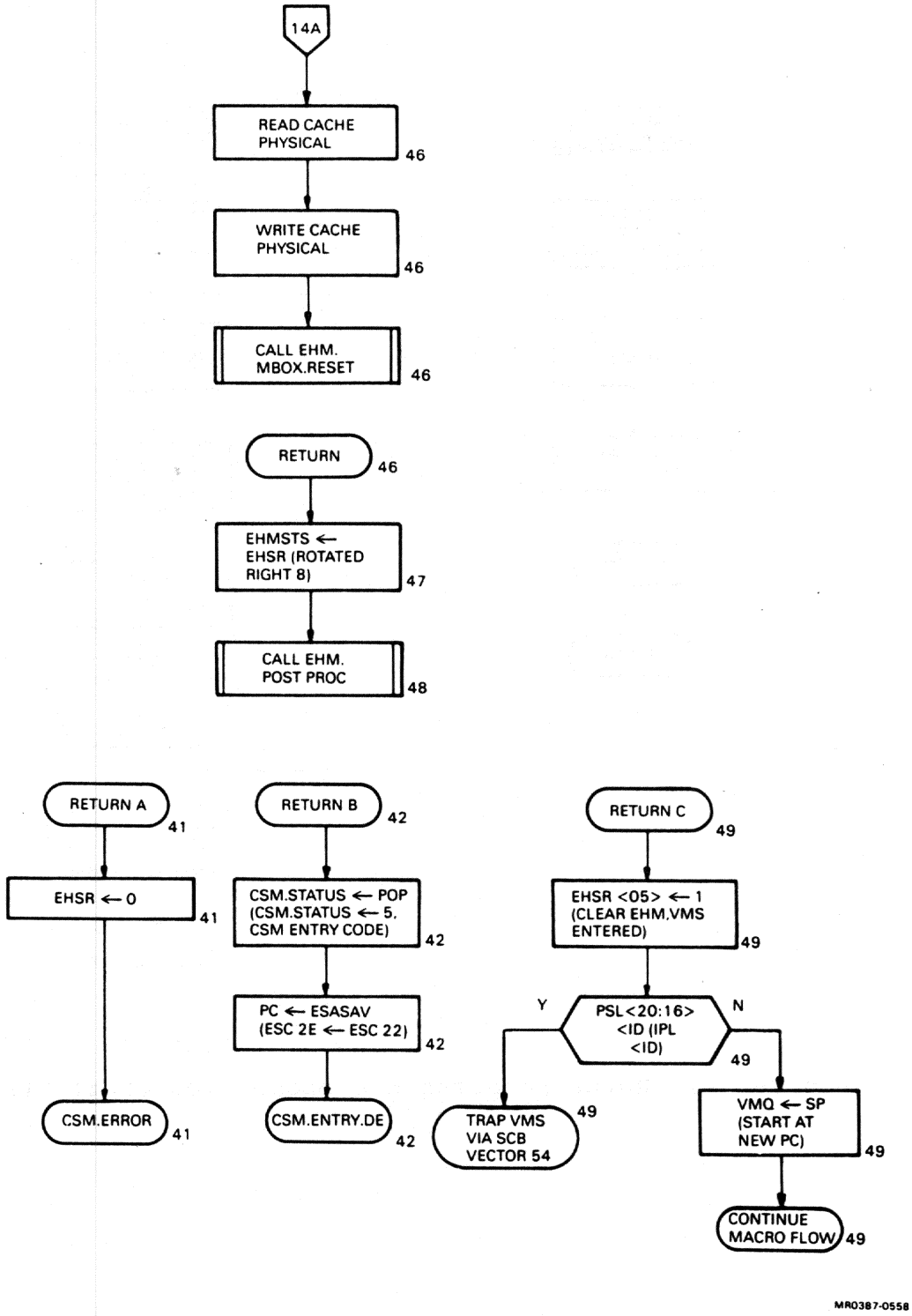


Figure A-1 Error Handling Microcode Flow Chart (Part 14 of 20)

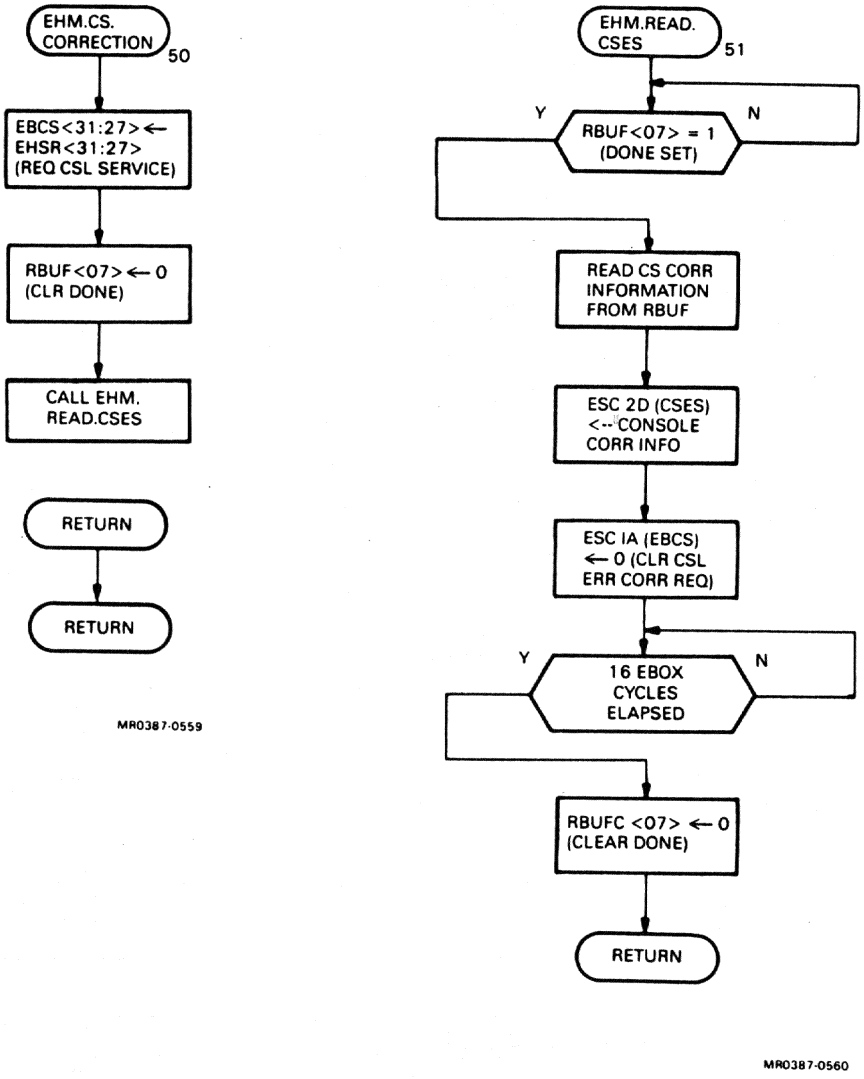


Figure A-1 Error Handling Microcode Flow Chart (Part 15 of 20)

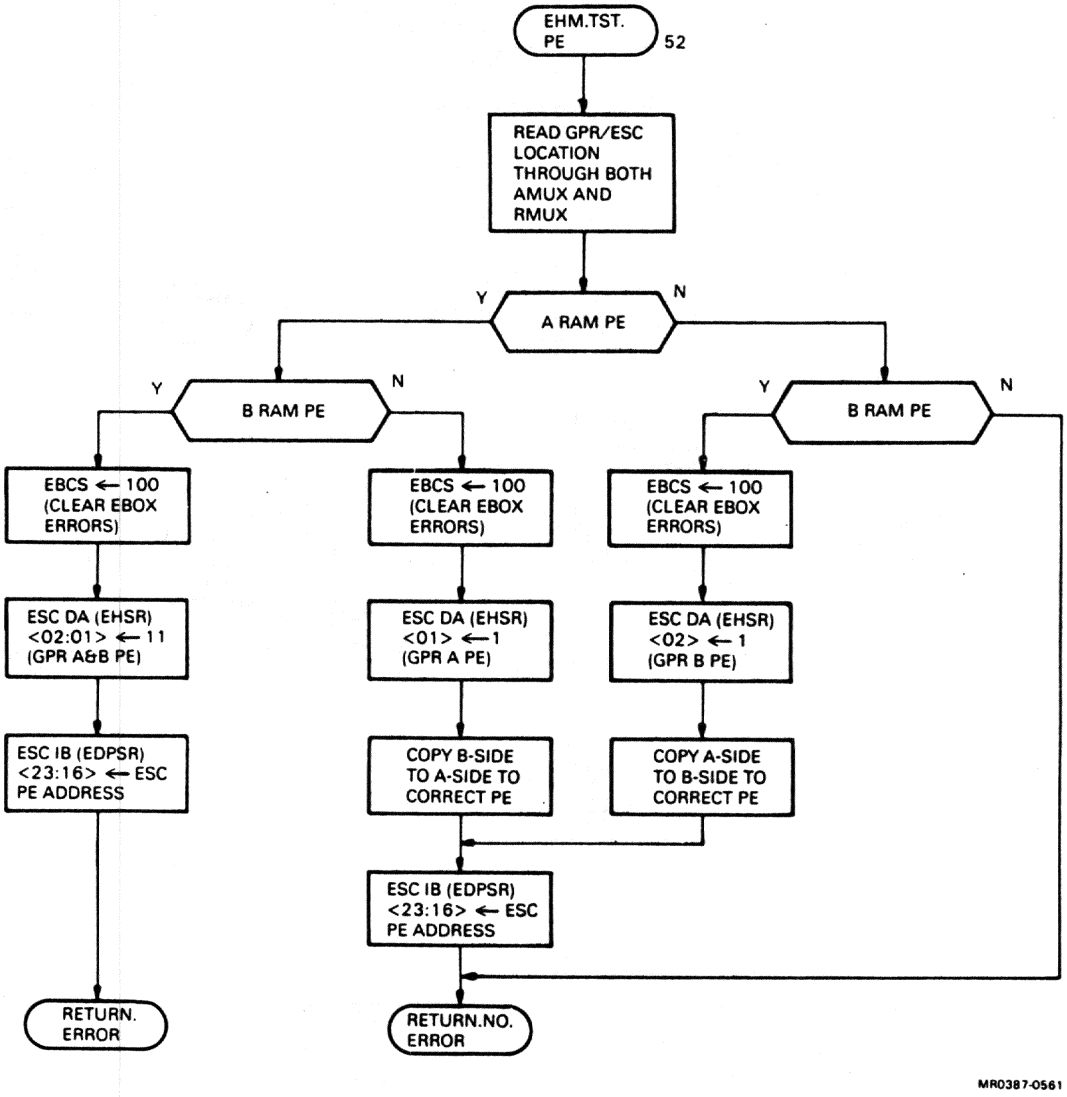


Figure A-1 Error Handling Microcode Flow Chart (Part 16 of 20)

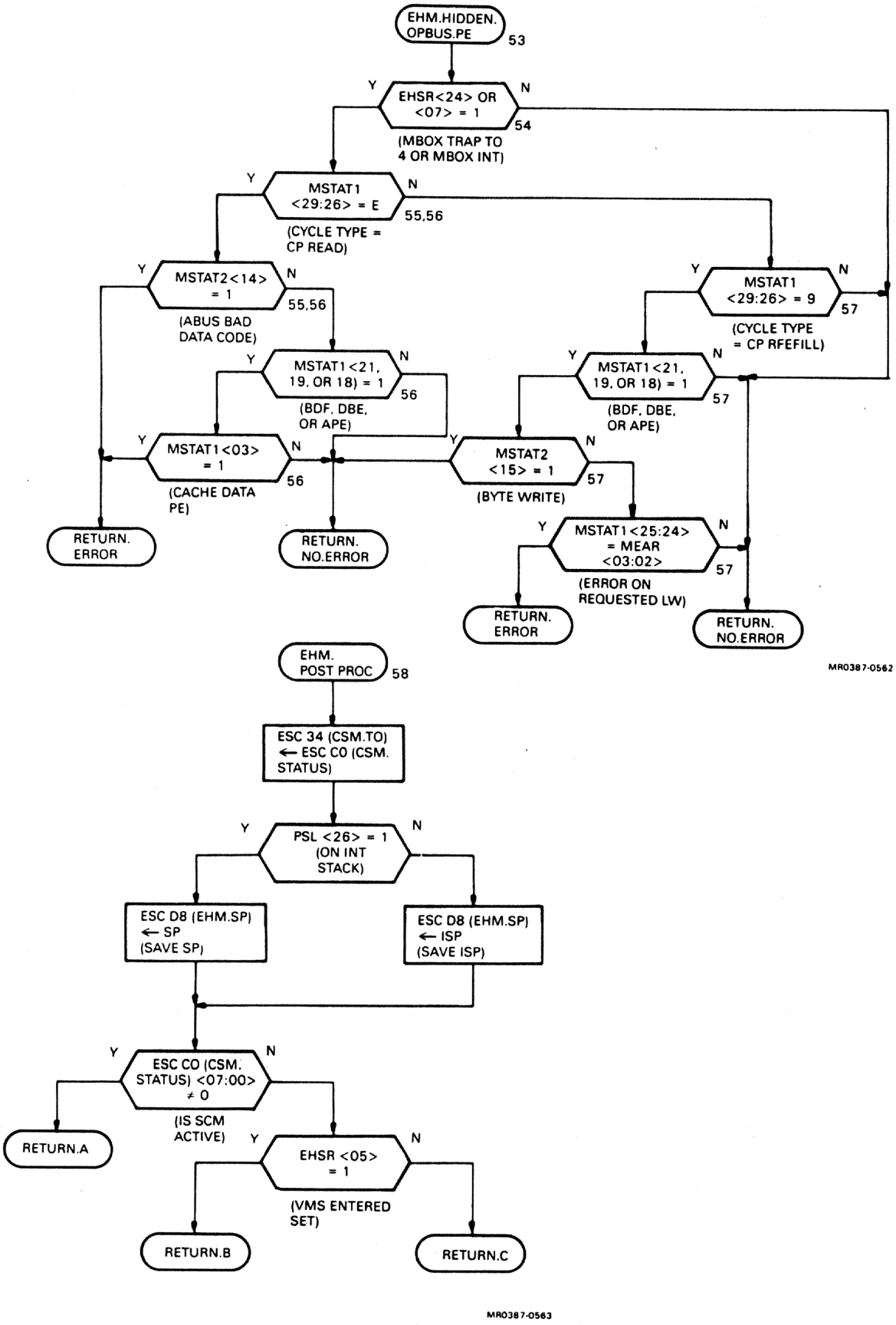
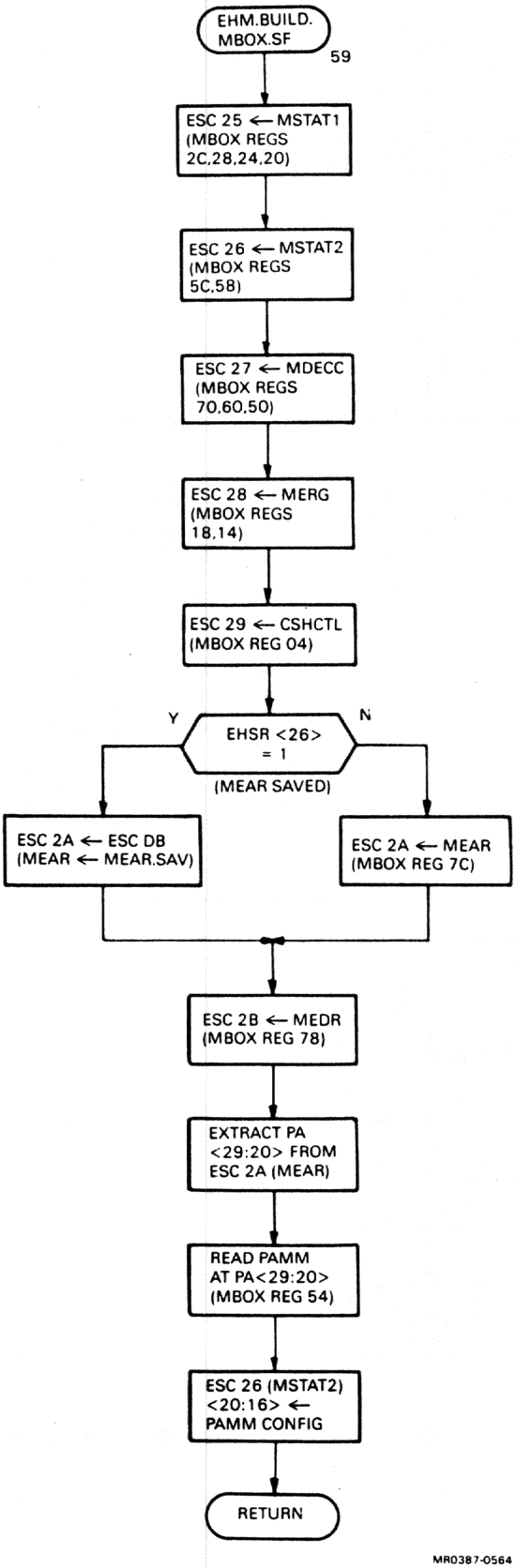


Figure A-1 Error Handling Microcode Flow Chart (Part 17 of 20)



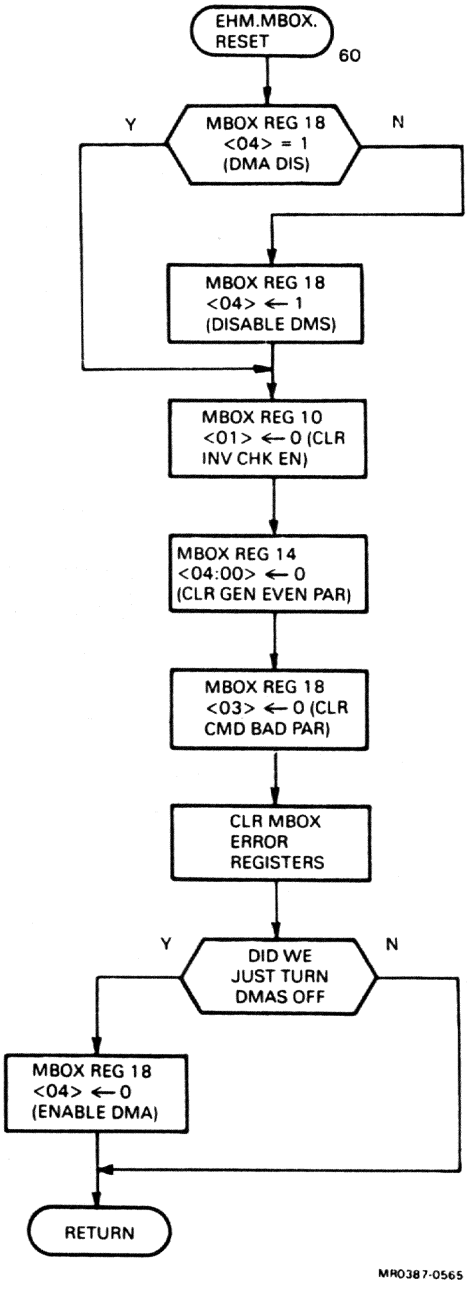


Figure A-1 Error Handling Microcode Flow Chart (Part 18 of 20)

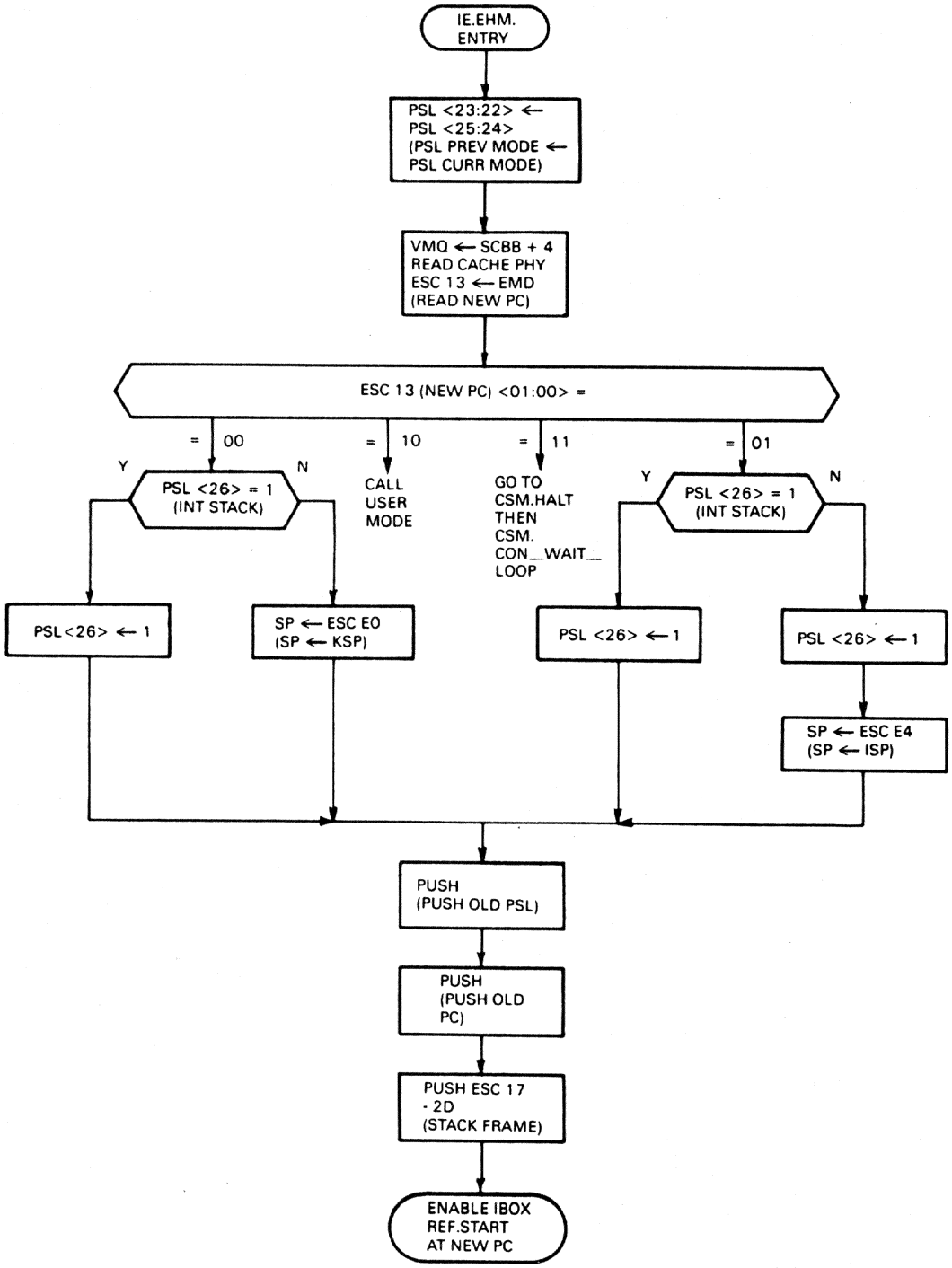


Figure A-1 Error Handling Microcode Flow Chart (Part 19 of 20)

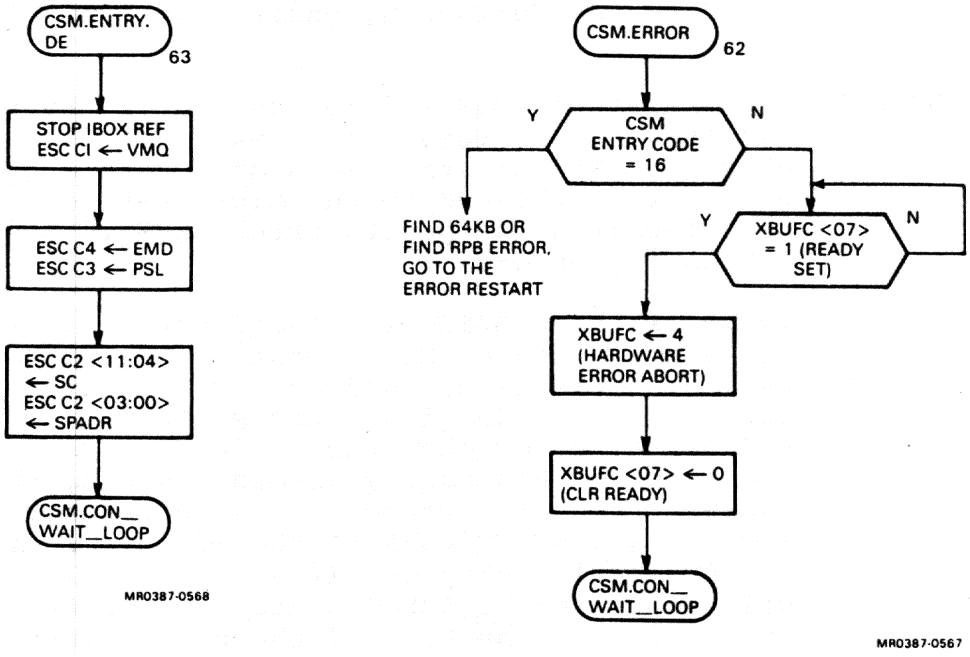


Figure A-1 Error Handling Microcode Flow Chart (Part 20 of 20)

NOTE 1 The entry at trap vector 8 is assigned to EBox errors and MBox fatal errors, both at main priority 0, sub-priority 0, and TB parity errors associated with EBox requests at main priority 0, sub-priority 1. These errors have the highest priority because they could affect the instruction the EBox is executing, which in turn may affect the ability of the system to retry the operation.

This is also the entry point for ESC parity errors that are detected during ESC parity checking.

NOTE 2 The EHM stops IBox references and checks to see if EHM ENTERED (EHSR <06>) is set. If it is, STATE <07> is checked. If set, the EHM simply does a RETURN [0], which will return the EHM to the microinstruction after the one that incurred the parity error (EHM is checking the ESC for parity errors).

To check the GPR/ESC for parity errors, it is necessary to examine each ESC location on both the A-side and B-side. This is carried out by EHM, after EHM ENTERED is set. This code also sets STATE <07>, to indicate that ESC parity error checking is in progress. In the event that there are parity errors, reading the location will cause a trap to vector 8. Therefore, we check EHM ENTERED and STATE <07> to see if we were in fact checking for ESC parity errors. If so, the parity checking code will use the latched status in the PDP MCA to indicate which side of the ESC had the parity error.

One pitfall is, that to check for EHM ENTERED, it requires reading ESC DA (EHSR). If a parity error is encountered while reading the EHSR, the EHM will be in a tight loop of 008 - 02F - 008 - 02F, etc. (for EHM revision 1).

If we are not testing for ESC parity errors, STATE <07> is not set and it is an EBox double entry error (the second time through a trap to 8). EHM will transfer the EBCS to the VMQ (visible on the EVA) and go into a sunset loop at uPC 21.

If EHM entered is not set, the shift counter will be loaded with the vector, 8, and EHM will join the common flow.

NOTE 3 The entry at trap vector 2 (main priority 3, sub-priority 2) does not go directly to EHM, but to the main flow of EBox microcode. This code will stop IBox references, assemble FBXERR from FBox registers 1, 2, and 3, then set EHSR <04> to indicate that there is an FBox service request. The vector, 2, is loaded into ESC 12, then control is passed to EHM at the entry point for trap 2 and trap 6 (EHM.ENTRY.VECTOR8.30).

During an FBox operation, a problem may arise which the FBox cannot process on its own. The FBox asserts FBOX PROBLEM, FBOX ABORT, and FBOX WRITE PROBLEM when it drives the WBus. When the EBox forks to get the destination, it asserts CP SYNC, which allows the EBox trap logic to process FBOX PROBLEM. If the destination address is for a GPR, the trap is allowed (trap vector 2), the WBus write will be aborted to prevent writing bad data into a GPR, and the IBox is prevented from writing the ESA (allowing for the retrieval of the starting address of the instruction with the problem).

If the destination address is a cache address, the EBox trap logic is prevented from generating a trap to 2 because an operand write is in progress. Because the MBox would be writing the result to cache, it recognizes that FBOX WRITE PROBLEM is asserted. In response to the error, the the MBox will assert a port status code of 9, which will cause an operand port trap to 11.

The trap to 11, like a trap to 2, is initially handled by the main body of EBox microcode. It will do the same as it does for a trap to 2, stop IBox references, assemble FBXERR from FBox registers 1, 2, and 3, set EHSR <04>, FBox service request, load ESC 12 with the vector, 11, then pass control to EHM at the entry vector for trap 2 and trap 6.

NOTE 4 The EBox trap logic will generate a trap to 6 (main priority 3, sub-priority 6) at IRD time if there is an interrupt pending. The EBox exception and interrupt microcode will initially process the interrupt.

It will first stop IBox references, set EBCS <13> (WBus register) to clear IBox error, then unwind the PCs and RLOG. If CSLINT <20> is set it is an external interrupt. CSLINT <19:16>, the interrupting IPL, is checked for 0 (zero), and if 0, it must have been a spurious interrupt and the next macro instruction is executed. If the IPL is not 0, then the I/O adapter interrupt is attended to.

NOTE 5 If CSLINT <20> is not set, it is an internal interrupt. The microcode will branch on CSLINT <29:23> to determine which internal interrupt is the pending interrupt. An MBox interrupt is indicated by CSLINT <27> being set.

For the MBox interrupt, the exception and interrupt microcode will set EHSR <07> to indicate that there is an MBox service request, clear EBCS <02:01> (mem/gpr write and I/O read flags), load ESC 12 with the vector, 6, then pass control to EHM microcode at EHM.ENTRY.VECTOR8.30.

NOTE 6 This is the entry point for vector 4, MBox Error Address Register Full (MEAR full), at main priority 3, sub-priority 4. This trap is recognized at IRD time.

MEAR, the MBox error address register, holds the offending address for the last MBox detected error. Whenever the MBox detects an error condition, it latches the error in the various error registers and tries to tell the EBox about the error via trap 8 for fatal MBox errors, or trap 6 (interrupt) for non-fatal errors.

If the error is not a fatal error, and the CPU is operating at an IPL above 1D, the IPL for MBox interrupts, then the EBox won't be notified of the MBox error condition. In this case, the MBox would not be able to record the address and details of a second error, because the first error has not been logged. To remedy this situation, the trap to 4 will save MEAR if it hasn't already been saved.

The EHM microcode will stop IBox references, clear the EBox port status, and set EHSR <24> to indicate that there has been a trap to 4, then check EHSR <26> to see if MEAR has already been saved.

NOTE 7 If EHSR <26> is already set, MEAR.SAV is already full, we have nowhere to save MEAR. When we finally get to handle the error, the MBox multiple error bit will be set. So, MEAR will be cleared, the PCs and RLOG will be unwound, and the next macro instruction will be executed.

In addition, if ECC correction is inhibited by MBox register 10 <02>, EHSR <26> is cleared (MEAR saved) and the MBox reset (see note 60).

NOTE 8 If MEAR has not been saved, MEAR saved will be set (EHSR <26>), and MEAR (MBox register 7C) will be copied to MEAR.SAV, ESC DB. Then MEAR will be cleared and the MBox stack frame will be built (see note 59).

Call a routine to check for hidden OPBus parity error (see note 53).

On the return, if process abort condition 4 (CPU failed to detect an OPBus parity error) or 5 (CPR parity error) exist, then join the regular EHM flow (note 15). Otherwise the next macro instruction will be executed (see note 7).

NOTE 9 Trap vector 10 is assigned to IBox errors detected during an OP-Port write (main priority 1, sub-priority 0), or an EBox read IMD operation (main priority 5, sub-priority 0).

This trap vector is also assigned to TB parity errors that are associated with an OP port request during an OP port write (main priority 1, sub-priority 1) and TB parity errors that are associated with an OP port request during an EBox read IMD operation (main priority 5,

sub-priority 1).

EHM will stop IBox references and load the SC with the vector, 10, then join the common flow for IBox errors.

NOTE 10 Trap vector 18 is assigned to IBox errors detected during an EBox fork (main priority 4, sub-priority 0), or EBox read ID operation (main priority 6, sub-priority 0), or an EBox string read operation (main priority 7, sub-priority 0).

This trap vector is also assigned to TB parity errors associated with IBuffer port requests during EBox fork (main priority 4, sub-priority 1) and TB parity errors associated with OP port requests during EBox string reads (main priority 7, sub-priority 1).

EHM will stop IBox references and load the SC with the vector, 18, then join the common flow for IBox errors.

NOTE 11 Trap vector 1E is assigned to IBox errors detected during an IBox flush and load (CPC SYNC) operation (main priority 2, sub-priority 0).

EHM will stop IBox references and load the SC with the vector, 1E, then join the common flow for IBox errors.

NOTE 12 Trap vector 1F is assigned to IBox errors detected during an IBox RLOG unwind operation (main priority 2, sub-priority 0).

EHM will stop IBox references and load the SC with the vector, 1F, then join the common flow for IBox errors.

NOTE 13 Any IBox error will clear EBox status and check for an operand specifier that attempted to use an addressing mode that is not allowed (reserved addressing mode fault, RAF). IBESR <27> will be set if there was a RAF.

If there is a RAF, IBESR <26:21> is checked for any non-zero bits, in other words, any other IBox errors. If there are any other IBox errors, along with the RAF, the error will be attended to (hardware error). This will insure that a hardware problem will have higher precedence than a RAF.

If there are no other IBox errors, the RAF vector, 1C, will be loaded into ESC 12, EBCS <13> will be cleared (IBox error bit), and control will be transferred to the EBox main microcode to handle the RAF fault.

NOTE 14 If there was a trap to 4 (FBox problem), 6 (MBox interrupt), or 11 (FBox write problem), the code will end up at EHM.ENTRY.VECTOR8.30. For an FBox error, FBXERR contains the contents of FBox registers 1, 2, and 3. It will be transferred to ESC 30 for safekeeping. The return PC in ESC 2F will be transferred to ESC 31. ESC 12, which contains the vector will be loaded into the shift counter.

NOTE 15 For all errors except EBox errors, the vector (in SC) will be loaded into ESC 16 to prepare for a possible EHM double entry. If a second error occurred after EHM ENTERED had been set (EHSR <06>), we will prepare for the CSM wait loop.

For the double error, ESC 16 (vector) is loaded into CSM.STATUS (ESC C0) <15:08> and a CSM entry code of 5 (non-EBox double error was detected) is loaded into CSM.STATUS <07:00>. ESA is read, then written into ESC 2E, and control is transferred to CSM.ENTRY.DE in the main EBox microcode (see note 63).

NOTE 16 If EHM ENTERED is not set, we join the first common point for all trap vectors. The EHM follows a common path for the remainder of the code.

EHM will clear EBox status, load the trap vector, from SC, into EHSR <23:16>, then set EHSR <06>, EHM ENTERED. All constants used by EHM will be rewritten into the scratch pad registers to prevent an inadvertent parity error, and ESC locations 10 through 2F (stack frame) will be written with FFFFFFFF. ESC 17 will be loaded with 58 (Hex), the number of bytes in the stack frame, the contents of the VMQ at the time of the error will be popped off the ESC stack and written into ESC 19, the EBCS will be transferred to ESC 1A, and ECS 1A <19:16> (process abort code) is cleared.

NOTE 17 ESC 1A <09> is checked to see if there is an EBox data path parity error. If so, ESC 1B is first cleared, then EDPSR is assembled from the six 4-bit registers in the PDP MCAs and written into ESC 1B. If there is no EBox data path parity error, ESC 1B is cleared.

NOTE 18 The CSLINT register is loaded into ESC 1C and the IBESR is assembled in ESC 1D; ESC 1D <31:16> from IBE and ESC 1D <15:00> from EDMS. ESC 1D <02:00> will then be set according to the valid bits for ESA, ISA, or CPC. This has to be done at this time because it is the EBox error signal (if there was an EBox error) that is holding the valid bits, and the EBox errors will be cleared shortly.

A 100 (Hex) is then written to WBus register EBCS. Writing a logic "1" to bit 08 will clear EBCS <15>,

<12:09>, and <04>, clearing EBox errors.

NOTE 19 The data stack, bank F of ESC, will be checked for parity errors (see note 52). Part of the set up is to set STATE <07> to allow for the trap to 8 which will occur for an ESC parity error. During ESC parity error checking, an ESC parity error looks like a second error, but the EHM, upon seeing STATE <07> set will cause the next microinstruction after the microinstruction causing the parity error, to be executed (see note 2) to allow the completion of ESC parity error checking.

An error return will occur only if there is a parity error in both sides of the GPR/scratch pad registers, and FFFFFFFF is written to the failing ESC location. If there is a parity error in both sides of the scratch pad stack, the only thing lost is data used by the microcode during the execution of an instruction, and the instruction will be retried.

The non-error return will occur if there was no parity error, or a parity error on one side only, in which case the good side is copied to the bad side.

All 16 locations in the block will be checked, so if there is more than one parity error, EDPSR will contain the information for the last parity error.

NOTE 20 EHM pops the last two words that the EBox wrote to cache off the scratch pad stack and stores them in ESC 1E and 1F. The PSL is stored in ESC 2F, the IVA in ESC 20, VIBA in ESC 21, ESA in ESC 22, ISA in ESC 23, and the CPC in ESC 24.

NOTE 21 If we are here because of an FBox problem that caused a trap to 2 (EHSR <04> = 1), then ESC 30 (FBXERR) will be loaded into ESC 2C in the stack frame. If it was an MBox interrupt, then we will load the return PC, from ESC 31, into ESC 2E. If either 2C or 2E are not written at this time, they will remain all F's.

The remainder of the MBox stack frame will be built at this time (see note 59) followed by resetting the MBox (see note 60).

NOTE 22 EHM sets up to test ESC bank 0 (GPRs) for parity errors (see note 52). Part of the set up is to set STATE <07> to allow for the trap to 8 which will occur for an ESC parity error. If there are no errors, or an error in one side only, the return is RETURN.NO.ERROR.

An error return is made only when there is a parity error on both sides. For this case, the bad GPR location is written with FFFFFFFF, the process abort bit is set (EHSR

<25>), and a process abort code of 1 (unrecoverable GPR parity error) is written to EBCS <19:16>.

For an error return, or non-error return, the GPR is rewritten to insure that the FBox and IBox contain the same data as the EBox.

NOTE 23 Once the GPRs have been tested, EHM sets up and checks the remainder of the scratch pad registers, the constants and architectural registers, in banks 3 through E (see note 52). Part of the set up is to set STATE <07> to allow for the trap to 8 which will occur for an ESC parity error.

For these banks of scratch pad locations, an unused bit, bit 08, in EHSR will be set if there is a parity error in both sides of the ESC. EHSR <08> will be checked at a later time, and if it is set, and if EHM was entered at vector 8 (EBox error or MBox fatal error), will force EHM to a sunset loop at uPC 20 (see note 39).

NOTE 24 If an EBox control store parity error caused the microtrap, the parity error has already been corrected by the console. EHM will read the control store error correction information from the CBus interface (CSES) and write it into the stack frame (see note 51).

When an EBox control store parity error (CSPE) occurs, the EBox stalls because the parity error prevents clocking the control store data register. The parity error will generate an interrupt to the console, which will correct the parity error.

After correcting the control store parity error, the console will use the SDB control channel to generate a "CSPE RESET" in the EBox, which will cause a microtrap to vector 8.

NOTE 25 If there was a WBus parity error, there is a possibility that bad data has been written somewhere, so set the process abort bit (EHSR <25>), and load EBCS <19:16> with a process abort code of 2 (WBus parity error). EBCS <09> will also be set to indicate an EBox data path parity error.

The trace pending (TP) bit is reset to prevent a trace trap when the REI instruction is executed.

NOTE 26 IBESR <28> indicates whether the IBox selected the WBus (logic 1) or GPR (logic 0) data. If IBESR <23> (IBox AMUX PE) is set, IBESR <28> indicates where the data, and parity error came from, WBus or GPR. But, the VMS error formatter interprets IBESR <28> as a WBus parity error if it is set, even if IBESR <23> is clear. Therefore, EHM

will clear IBESR <28> if IBESR <23> is not set.

NOTE 27 If there is an IBox control store parity error or IBox dispatch RAM parity error, they must be corrected before the IBox error is cleared in the EBCS because it is the IBox error bit that holds the CSPE error address.

If there is an IBox DRAM or control store parity error, EHM sets EHSR <28> or <27>, and the parity error will be corrected (see note 50). EBCS <13>, IBox error, is cleared after the error has been corrected.

NOTE 28 EHM checks IBESR <23> to see if there is an IBox AMUX parity error. If there is, it then checks to see if the AMUX is enabled for the WBus or the GPRs. If IBESR <28> is not set, the GPRs were selected and the error is an IBox GPR parity error, so EHSR <00> is set. If IBESR <28> is set, the data from the WBus caused the parity error. In this case, the problem is most likely in the IBox WBus receivers or AMux itself. (If the WBus had been the cause of the parity error the EBox should have detected the error.)

NOTE 29 If the IBox error was an RLOG parity error, as indicated by IBESR <24>, then we cannot do an unwind. EHSR <25>, process abort is set, and a 6 is written to EBCS <19:16> to indicate that the process abort was caused by an RLOG parity error.

It there was an RLOG parity error, EHM will now go to the FBox fixup section.

NOTE 30 With no RLog PE, if EHSR <07> (MBox service request) is set, we have already done an unwind, so control is transferred to the FBox fixup section.

If there is not an MBox service request, unwind the PC's and RLOG, then check the validity of the PC's. If all of the PC's, ESA, ISA, and CPC, are invalid, set EHSR <25> (process abort) and load EBCS <19:16> with a process abort code of 3, (all IBox PC's are invalid).

NOTE 31 In the FBox fixup code the first check is for an FBox interrupt, EHSR <04>. If there was not an FBox interrupt, EHM will load ESC 2C, FBXERR with FFFFFFFF, then reset the FBox and initialize the FBox temporary scratch pad registers.

If there was an FBox interrupt, a further check is made to see if it was caused by an FBox GPR parity error, as indicated by FBXERR <17> being set. If this is the case, EHSR <03> will be set.

NOTE 32 FBXERR <21:19> are checked to see if there was an FBM control store parity error, FBA control store parity error, or FBox dispatch RAM (FDRAM) parity error. If any of these bits are set, the set bits are transferred to the corresponding bits in EHSR <31:29>. The control store parity error will be corrected by the console (see note 50).

After the console completes the correction, EHM will reset the FBox and initialize the FBox temporary scratch pad registers.

NOTE 33 EBCS <02>, MEMORY OR GPR WRITE, will be set when the EBox writes to a GPR, which it will do on a floating point instruction when the FBox passes the data to the EBox.

If there is an FBox error, FBOX PROBLEM will prevent the EBox from writing the GPR, but EBCS <02> will still get set. Therefore, the state of the machine has not been changed, but the EBCS contains an erroneous bit.

For string instructions this is not true, as the string instruction could have previously written a GPR. However, the MEM WRITE bit should never be set if we trap through vector 2, FBOX PROBLEM, and the instruction was not a string instruction.

Therefore, if first part done (FPD, PSL <27>) is clear, EBCS <02>, MEMORY OR GPR WRITE, is cleared.

NOTE 34 At this point, EHM enters the MBox fixup section. The first check is for the case where the MBox sent bad data to the EBox but the EBox didn't detect a parity error (see note 53).

If the EBox didn't detect the OPBus parity error, process abort, EBCS <25> will be set, a process abort code of 4 (EBox failed to detect OPBus byte parity error) will be written to EBCS <19:16>, and EHSR <09>, an unused bit will be set. EHSR <09> will be used as a branch condition at a later time (see note 44).

NOTE 35 If we have either of MSTAT1 <23:22> set, we had a CPR (Cycle Parameter RAM) parity error, and must set the process abort bit, EHSR <25>. A process abort code of 5 (CPR parity error that did not cause an MBox fatal error) will be written to EBCS <19:16>.

NOTE 36 If any of bits MSTAT1 <11:08> are set, we had a TB parity error, and it is necessary to clear the TB location. EHM will clear the entire translation buffer. Also, EHSR <09>, an unused bit, will be set. It is used as a branch condition at a later time (see note 44).

NOTE 37 Here we check to see if we had a cache tag parity error or a cache written bit parity error, MSTAT2 <06:05>. If DMAs are enabled, they will be disabled for a short time.

If the cache in error is not turned on, neither cache will be cleared or swept, but if the cache in error is turned on, the other cache will be swept.

For a cache tag parity error, the offending cache block will be cleared. For a cache written bit parity error, the offending cache will be swept, then cleared.

EHM then clears EBox status, clears the MBox error registers, and turns on the caches as they were to begin with. (The original cache enable bits are saved, then restored, because the cache enable bits are used to force a cache sweep or to clear cache.)

DMAs will be enabled at this time if the DMA enable bit was originally set.

VMS will bugcheck if we had a cache tag parity error with the written bit set. In that case, we don't know whose data got lost. Go to EHM.EXIT to figure out what mode we're in and how to exit EHM.

NOTE 38 We will end up here, at EHM.EXIT, unless there has been an SBE (unrecorded check bit 0 SBE). EBCS <23:21> will be loaded with 0001, the stack frame revision. (The KA8600 was announced with a Rev. 0 stack frame, and this is the first revision to EHM.)

EHSR (ESC DA) is rotated right 8 to get the bits into correct position, then loaded into EHMSTS (ESC 18).

NOTE 39 If EHSR <08> is set, there was a parity error in both sides of the GPR/scratch pad registers, in a location reserved for constants and architectural registers, banks 3 through E.

If, in addition, the microtrap vector is 8 (EHSR <23:16> = 8, EBox error or MBox fatal error) EHM will go into a sunset loop at microPC 20.

NOTE 40 If there was an uncorrectable ESC parity error but the vector was not 8, control will be passed to the EHM post processor code, which will figure out what mode we are currently in and how to exit EHM (see note 58). Control will also be passed to the EHM post processor code if there was not an uncorrectable ESC parity error.

NOTE 41 If CSM was active, ESC DA (EHSR) will be cleared and control passed to CSM.ERROR in CSM microcode (see note 62).

NOTE 42 If VMS ENTERED is set, we have a double error condition. The second error occurred while the VMS machine check handler was active.

CSM.STATUS (ESC C0) <07:00> will be loaded with 5, the CSM entry code for a non-EBox double error. The PC will be loaded from ESASAV and control will be passed to CSM.ENTRY.DE in main EBox microcode (see note 63).

NOTE 43 This is the regular exit from EHM to the VMS machine check handler (CSM is not active and VMS ENTERED is not set).

VMS ENTERED is set (EHSR <05>) and FPD (PSL <27>) is set. Control is passed to IE.EHM.ENTRY (see note 61).

NOTE 44 Due to an MBox bug, a parity error in check bit C0 may not get latched as an SBE. The MBox detects the error and delivers an interrupt, but MDECC <21> (SBE) may not get latched. Check for any single bit errors by first eliminating other MBox errors.

If EHSR <09> is set we had a hidden OPBus parity error or TB parity error. In this case EHM will go to EHM.EXIT to figure out what mode we are in and how to exit EHM (see note 38).

If MSTAT1 indicates any of the following errors, go to EHM.EXIT.

1. CPR B PE (MSTAT1 <23>)
2. CPR A PE (MSTAT1 <22>)
3. ABus data PE (MSTAT1 <21>)
4. ABus command or mask PE (MSTAT1 <20>)
5. ABus address PE (MSTAT1 <19>)
6. CPU write PE (MSTAT1 <07:04>)
7. Cache data PE (read cache) (MSTAT1 <03>)
8. Cache byte write PE (MSTAT1 <00>)

If MSTAT2 indicates any of the following errors, go to EHM.EXIT.

1. ABus bad data code (MSTAT2 <14>)
2. NXM (MSTAT2 <03>)
3. CP I/O buffer error (MSTAT2 <02>)

If MDECC indicates any of the following errors go to EHM.EXIT.

1. Bad data flag, BDF (MDECC <22>)

2. Double bit error, DBE (MDECC <20>)

3. Address parity error, APE (MDECC <19>)

At this point we check EHSR <07> to see if there has been an MBox interrupt. If not, there is no MBox error. Go to EHM.EXIT.

If there are none of the above errors, and there is an MBox interrupt, there must have been an SBE, so check MDECC <21> (SBE). If it is set, go to EHM.SBE.FIXUP (see note 46).

NOTE 45 If MDECC <21> is not set (doesn't indicate an SBE), then we must check to see if it could possibly be the unrecorded check bit parity error. This error can only occur on a CP or ABus refill, so MSTAT 1 <29:26> is checked for these cycles. If not a refill cycle, go to EHM.EXIT.

If it was a refill cycle, then the error must have been an unrecorded SBE, so EHM will set MDECC <21>, SBE, and clear the syndrome bits, MDECC <14:09> (SBE on C0 = 0 syndrome).

NOTE 46 EHM.SBE.FIXUP -- When a single bit error occurs, the MBox fixes the data on the fly, delivers good data to the cache, and requests an interrupt. Because this is a refill cycle, the written bit won't be set, but it should be set for an SBE (the forced writeback will update and correct memory). Because of a timing bug, the written bit doesn't always get set, so we will insure that it gets set by reading the data and writing it back.

MEAR (block address) and MSTAT1 <25:24> (longword in error) are used to generate the cache address, and the EBox will read cache physical, then write the data (from the EMD) back to cache at the same address. This will insure that the written bit gets set.

At this time the MBox will be reset to clear the MBox error registers.

NOTE 47 EHSR (ESC DA) is rotated right 8 to get the bits into correct position and written into EHMSTS (ESC 18).

NOTE 48 The EHM post processor will figure out what mode we are currently in and how to exit EHM.

NOTE 49 We have had a single bit error and finished the fixup code. EHSR is cleared to clear EHM ENTERED and VMS ENTERED.

If the IPL is below 1D, we will trap VMS via SCB vector 54, single bit error.

There is a bug that causes MBox interrupts for SBE at IPL 1F. If the IPL is above 1D, we shouldn't get an interrupt for a SBE at this time, it should not be seen until the IPL drops below 1D. If the IPL is above 1D, we'll just ignore the whole thing and continue with the next macroinstruction.

NOTE 50 EHM.CS.CORRECTION is a routine called by EHM when it detects an FBox or IBox control store parity error. This routine will call the EHM.READ.CSES routine to read the control store error correction data for the CSES register. EHM.READ.CSES will pass control back to EHM.CS.CORRECTION which will immediately pass control back to the calling routine.

When an IBox or FBox control store parity occurs, the box will hold the error information, and report the error to the EBox via error microtrap. The EBox will transfer the control store parity error information from the applicable error register to the EHSR prior to passing control to the control store correction sequence.

A control store parity error correction sequence is initiated by EHM when it transfers EHSR <31:27> to EBCS <31:27>. The setting of the CSPE bits in the EBCS will enable a priority encoder to send a 3-bit code to the console. This code interrupts the console and indicates which control store (or DRAM) has the parity error.

EHM will clear RBUF <07> (DONE) to indicate to the console that it may load the receive buffer (RBUF), then call EHM.READ.CSES (see note 51).

NOTE 51 EHM.READ.CSES -- This routine is called by the control store correction sequence, and by EHM for EBox CSPE correction.

EHM will wait for the console to set DONE (RBUF <07>), an indication that the console has completed the control store parity error correction. EHM will then read the control store correction information from the CBus interface receive buffer, and load it into ESC 2D (CSES).

EBCS <31:27> is reset to clear the control store parity error correction request to the console, clearing the

console interrupt. EHM will wait 16 EBox cycles to allow the console interrupt time to go away.

EHM will then clear DONE (RBUFC <07>) and return control to the calling routine.

NOTE 52 EHM.TST.PE -- EHM calls this routine to check a particular location in the GPR/scratch pad RAMs for parity errors. The routine will source both the A-side and B-side through the AMUX and BMUX. If there is a parity error, the PDP MCA will latch whichever side got the error. In addition, any error will cause a microtrap to 8, but part of the setup for ESC parity testing was to set STATE <07> to allow for the trap to 8. Any trap to 8, with STATE <07> set, will allow control to be passed back to the routine generating the trap, to the microinstruction following the trapped microinstruction.

If there is a GPR/ESC parity error, a 100 is written to EBCS to clear EBox errors (writing a "1" to EBCS <08> clears bits <15>, <12:09>, and <04>).

If there is an error on one side only, it is corrected by copying from the good side to the bad side, and returning by way of the "no error" return. If both sides are bad, the error return is taken.

For error reporting, EHSR <02> is set for GPR B parity errors and EHSR <01> is set for GPR A parity errors. EDPSR <23:16> is loaded with the GPR/ESC parity error address.

NOTE 53 EHM.HIDDEN.OPBUS.PE -- Because the IBox compresses MDBus byte parity into OPBus longword parity before sending the data and parity to the EBox, and the EBox only checks parity on the bytes that it uses, there is a possibility that the EBox will not detect bad parity if it is using an even number of bytes.

To illustrate the situation, lets look at two examples. In each of the examples, the EBox requests a longword, that just happens to be all zeros, and the longword is not in cache, thus a refill is necessary.

Example A-1: OPBus no parity error

The MBox sends four bytes of data, all zeros over the MDBus to the IBox along with four odd parity bits, logic ones, one for each byte of data. The IBox compresses (XOR and invert, or XNOR) the parity bits to provide OPBus longword odd parity, a logic one. (The XOR of four logic 1's provides a logic zero, which is inverted to provide OPBus longword parity.) The IBox sends this odd parity bit along with the four bytes of data. The EBox checks the parity, determines that it is odd (OK) and

continues executing the instruction.

Example A-2: OPBus Parity Error

In this case, we assume that there is a double bit error on the data read from the array.

As a result of the double bit error, the MDBus drivers are disabled at the time the data is to be sent to the IBox, thus the MBox sends the IBox all zeros for data and all zeros byte parity. This combination should provide a parity error.

The IBox compresses the byte parity bits as before to provide OPBus longword parity. In this case, the four zeros will also generate a logic one for OPBus longword odd parity (the XOR of four logic 0s provides a logic 0, which is inverted to provide the logic 1 for OPBus longword parity).

The EBox checks parity, determines that it is odd (OK) and continues to execute the instruction. The EBox has just swallowed bad data.

This is a hole in the CPU error detection network. By the time that it was discovered it was too late to implement a hardware solution. The EHM was modified to check for this condition and take the appropriate action.

The following conditions are needed for this problem to have occurred.

1. EHM entered via trap 4 or 6 (see note 54).
2. One of the following three conditions exist.
   a. CP READ and ABus bad data code (see note 55)
   b. CP READ and cache data parity error and (bad data flag (BDF) or double bit error (DBE) or address parity error (APE)) (see note 56)
   c. CP refill and no byte write and error is on requested longword and (BDF or DBE or APE) (see note 57)

NOTE 54 EHSR <24> is set if there was an MBox trap to 4 and EHSR <07> indicates an MBox interrupt.

NOTE 55 MSTAT1 <29:26> = E (cycle type = CP READ) and MSTAT2 <14> is set, which signifies ABus bad data code. This can only happen for a CP read of I/O space and the SBIA

receives Read Data Substitute (RDS) on the SBI. The SBIA will send data length/status = 11 (bad data code) with the ABus data.

NOTE 56 MSTAT1 <29:26> = E (cycle type = CP READ), MSTAT2 <14> is reset (no ABus bad data code), MSTAT2 <21> (BDF), <19> (DBE), or <18> (APE) is set, and MSTAT1 <03> (cache data parity error) is set.

NOTE 57 MSTAT1 <29:26> = 9 (cycle type = CP READ), MSTAT1 <21> (BDF), <19> (DBE), or <18> (APE) is set, MSTAT2 <15> = 0 (not a byte write), and MSTAT1 <25:24> = MEAR <03:02> (error is on the requested longword).

NOTE 58 EHM.POSTPROC -- The EHM routine EHM.POSTPROC figures out what mode we are currently in and sets up the stack pointer for the stack frame.

CSM.STATUS is loaded into CSM.T0, a CSM temporary register, just in case CSM is active.

If we are on the interrupt stack (PSL <26> = 1) then ESC D8 (EHM.SP, an EHM temporary register) is loaded with the contents of SP. If we are not on the interrupt stack then ISP is transferred to ESC D8. ESC D8 will end up with the stack pointer that will be used by the machine check handler as a pointer for pushing the stack frame.

If ESC C0 (CSM.STATUS) <07:00> is not zero, the error occurred while CSM was active and the exit will be RETURN.A. (See note 41).

If ESC C0 <07:00> is zero and EHSR <05> is set (VMS ENTERED), the error occurred while VMS was active, but after EHM had passed control to VMS, a double error, so exit via RETURN.B. (See note 42).

If neither of the above cases apply, it is the first error (not a double error). EHM will pass control to the VMS machine check handler, to VMS via SCB vector 54 (single bit error), or to VMS to execute the next macro instruction (SBE with IPL at 1F). (See note 43 or 49.)

NOTE 59 EHM.BUILD.MBOX.SF -- This routine will read MBox registers to build the MBox portion of the stack frame, ESC 25 through ESC 2B.

1. Build MSTAT1 (ESC 25) by reading MBox registers 2C, 28, 24, and 20.

2. Build MSTAT2 <15:00> (ESC 26) by reading MBox registers 5C and 58.

3. Build MDECC <23:00> (ESC 27) by reading MBox registers 70, 60, and 50.

4. Build MERG <15:00> (ESC 28) by reading MBox registers 18, and 14

5. Build CSHCTL <07:00> (ESC 29) by reading MBox register 04.

6. If MEAR has been saved (EHSR <26> = 1), load MEAR (ESC 2A) from MEAR.SAV (ESC DB). Otherwise read MBox register 7C and load MEAR.

7. Build MEDR (ESC 2B) by reading MBox register 78.

8. Extract the page number, PA <29:20>, from MEAR and read MBox register 54 (PAMM) at the offset provided by PA <29:20>. Load the PAMM CONFIG into MSTAT2 <20:16>.

NOTE 60 EHM.MBOX.RESET -- This routine is used to reset the MBox error registers and clear all of the MBox fault insertion register bits.

If DMAs are enabled, they will be disabled to prevent clearing any I/O related errors. EHM will wait 16 cycles after setting MBox register 18 <04> (disable DMA) before proceeding further.

MBox register 10 <01>, INV CHK EN, is cleared, as is MBox register 14 <04:00>, the generate even parity bits. Then CMD BAD PAR, MBox register 18 <03> is cleared. This bit is used to generate even ABus parity during an ABus command/address cycle. Finally, all MBox error registers are cleared (EBox microcode MCF).

If DMAs were enabled upon entering this routine, they must be re-enabled prior to returning to the calling sequence. DMAs are enabled by clearing MBox register 18 <04>.

NOTE 61 IE.EHM.ENTRY -- This routine is not apart of the EHM, it is actually the first part of the exception handler microcode. It will transfer the current mode bits (PSL <25:24>) to the previous mode bits (PSL <23:22>).

It will then load the VMQ from SCBB + 4 (the virtual address of the handling routine) and read cache physical. The resulting data is the new PC. Bits <01> and <00> have special meaning. If they equal 00 the kernel stack is to be used unless presently on the interrupt stack. If equal 01 use the interrupt stack. For the machine check handler, the interrupt stack is always used. A value of 10 indicates that user microcode is to be used, and 11 is invalid, halt. For the a value of 11, with the

VAX 8600/8650, that means going to the CSM console wait loop.

For PC <01:00> = 00 or 01, if ESC 2F (old PSL) <26> = 1, we are on the interrupt stack, so we will set PSL <26> (write WBus register 01).

For PC <01:00> = 00 and not on the interrupt stack, ESC E0 (kernel stack pointer) will be transferred to the SP.

For PC <01:00> = 01, and not on the interrupt stack, ESC E4 (ISP) will be transferred to the SP, and PSL <26> will be set.

For any of these cases (PC <01:00> = 00 or 01) the old PSL will be pushed on the stack, the old PC will be pushed on the stack, then the stack frame (ESC 17 through 2D) will be pushed on the stack.

EBox microcode will then enable IBox references and execution will start at the new PC, the virtual address of the machine check handler.

NOTE 62 CSM.ERROR -- If CSM is active, the routine CSM.ERROR, a part of CSM microcode, is entered. The CSM entry code will be checked. If CSM was executing the FIND_64KB or FIND_RPB routines when the error occurred, control will be passed back to the restart address for the routine being executed.

If neither of these routines is being executed, the code will check to see if XBUFC <07> (READY) is set. If not, microcode will loop waiting for the console to set it (signifying that the EBox microcode may use the transmit buffer). When XBUFC <07> is set, the EBox will write a 4 in XBUFC <05:00>, clear XBUFC <07> to indicate to the console that the EBox has written the XBUFC, then go to the CSM wait loop. The value of 4 in XBUFC <05:00> tells the console that there has been a hardware error while CSM was active.

NOTE 63 CSM.ENTRY.DE -- For a non-EBox double error condition, the routine CSM.ENTRY.DE, a part of console support microcode, will be executed. This routine will stop IBox references and load the following scratch pad registers.

1. Load ESC C1 with the contents of the VMQ
2. Load ESC C2 <11:04> with the contents of the SC
3. Load ESC C2 <03:00> with the contents of the SPADR
4. Load ESC C3 with the PSL

5. Load ESC C4 with the contents of the EMD

Control will then be passed to the CSM wait loop.

APPENDIX B

KA86/KA865 CONTROL STORE AND DISPATCH RAM CORRECTION FLOWS

KA86/KA865 CONTROL STORE AND DISPATCH RAM CORRECTION FLOWS

INTRODUCTION

This Appendix contains a brief description and a flow chart that describes how the Console handles interrupt requests from the VAX8600/8650 to correct Control Store (CS) and Dispatch RAM (DRAM) Parity Errors. In summary, upon detection of a CS/DRAM Parity Error the Error Handling Microcode (EHM) will interrupt the Console. If the Console is in Program I/O (PIO) mode, it will attempt to correct the error and return control to the (EHM). The EHM will will build a Machine Check Stack Frame and call the VMS Machine Check Handler which will append the Machine Check to the System Event File (ERRLOG.SYS).

If the correction attempt failed (or if the Console was called to handle an MBox Control Store Parity Error) the Console will print an error message on the terminal (See Chapter 4 Table 10) and declare a Keep Alive Fail Condition. This will result in the generation of a Snap File (SNAPn.DAT). See Appendix D.

NOTE

> Although the Console will attempt to correct MBox Control Store RAM Parity Errors it will not attempt to restart the system. The reason being, the MBox has already acted on the contents of the Control Store word in error. Thus the state of the MBox is unknown and a restart is not feasable.

If the system is in Console I/O (CIO) mode, no correction will be attempted. Instead the Console will print a message on the terminal and return the Console Prompt.

Looking at the hardware, nine of the ten CS/DRAMs in the VAX8600/8650 CPU have parity checking networks. (There is no parity protection on the MBox Access Violation RAM). Of the nine the following seven can be corrected on the fly via the Console.

- The EBox Control Store RAMS
- The IBox Control Store RAMs
- The IBox Dispatch RAMs
- The FBox Dispatch RAMs
- The FBox Adder Module Control Store RAMs
- The FBox Multiplier Module Control Store RAMs
- The MBox Control Store RAMs

When a CS/DRAM Parity Detection Network detects an error it will pass the error signal to the EBE module (print EBE5) where it is latched in EBCS. The signal then goes to a priority encoder (print EBE3) which will generate a 3-bit encoded signal (EBE CPU ERR <2:0> H). This signal goes to the Console (print CL09) where it is OR'ed to become CL09 CPU CS PE L and latched as CL02 BCPU CS PE L. CL02 BCPU CS PE L

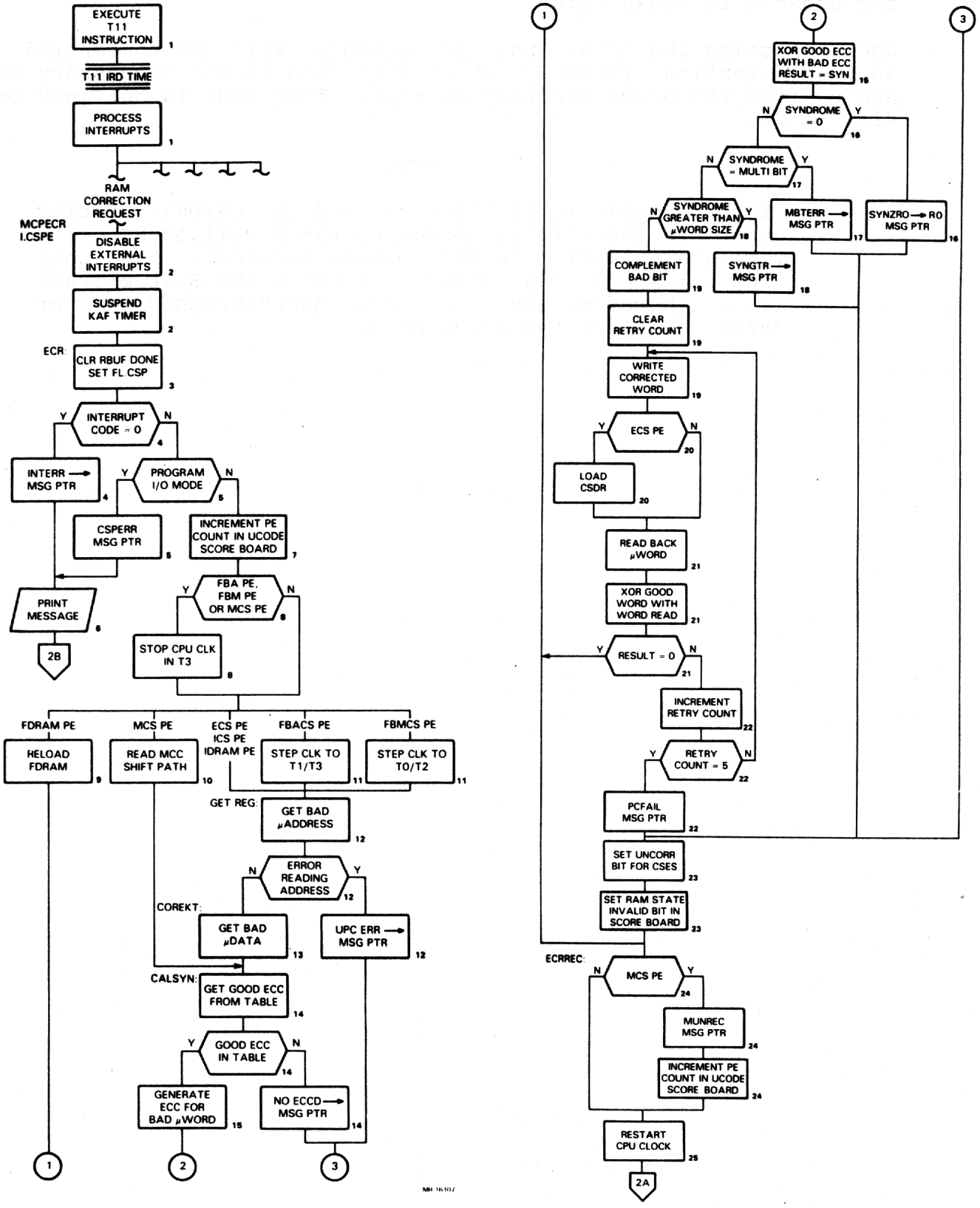
will cause a T-11 interrupt at priority 6, interrupt vector 110. In addition, the 3-bit encoded signal (EBE CPU ERR <2:0> H is latched by the console in MCSR3 <2:0>.

Upon detecting the interrupt, the console will pass control to a software routine (MCPECR) which will load in the neccessary overlay and perform the error recovery attempt. This code is outlined on the flow chart.

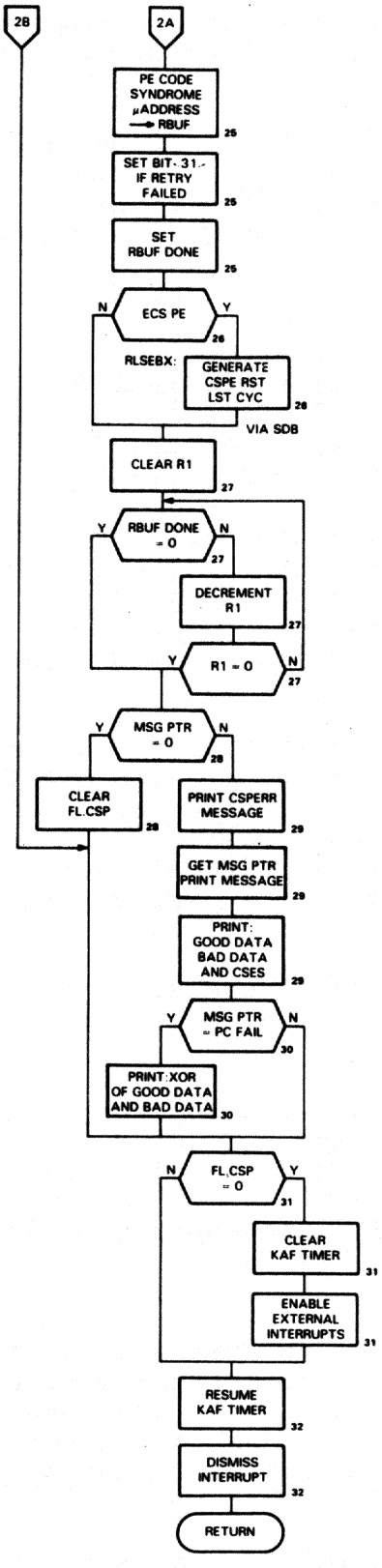
NOTE

If the console is in PIO mode and a CS/DRAM Parity Error is succesfully corrected there will be NO error message indication on the console terminal. The only indication of the error will be in the System Event File. Therefore you must run ANALYZE/ERROR_LOG or SPEAR to gather the error data.

KA86/KA865 CONTROL STORE AND DISPATCH RAM CORRECTION FLOWS



VAX 8600 Console CS/DRAM (ECC) Correction Flow (Part 1 of 2)



VAX 8600 Console CS/DRAM (ECC) Correction Flow (Part 2 of 2)

NOTE 1 The T11 Console processes interrupts at Instruction Register Decode (IRD) time. If the VAX8600/8650 has posted an interrupt request to correct a Control Store or Dispatch Ram parity error it will be processed at this time.

NOTE 2 This is the beginning of the Console Control Store/Dispatch RAM (CS/DRAM) correction routine. (MCPECR). Parts of the routine are resident and other parts are overlays that are read from the Console Load device as needed.

NOTE

The names of the subroutines that are called by this routine are labled on the flow chart. For example "I.CSPE:" is the name of the subroutine that disables external interrupts and suspends the KAF Timer.

The Console disables external interrupts (thus giving the ECC correction request the highest external priority). The Console also suspends the Keep Alive Fail (KAF) timer. This prevents a KAF time out from occuring while the Console is in the process of correcting a CS/DRAM parity error.

NOTE 3 The Console assures that RBUF "DONE" is cleared. Later RBUF "DONE" will be set to notify the VAX8600/8650 (Error Handling Microcode) that the Console has completed the correction request. The Console also sets a software status flag (FL.CSP). This flag will remain set until the Console has successfully corrected the parity error.

NOTE 4 The Console Checks to determine if the Interrupt Code is zero which is invalid. The only valid interrupt codes are: 1 through 7.

If the interrupt code is zero the console moves the value "INTERR" into the Message Pointer. Later this pointer will be used as an index to print the following message:

ERC-E-INTERR CSPE interrupt, code invalid (0)
(see Chapter 4 Table 10)

NOTE 5 The Console checks to determine if it is in Console I/O Mode. If it is it will not attempt correction. Instead it will move the value "CSPERR" into the Message Pointer. Later this pointer will be used as an index to print the following message: (Where "ram id" will correspond to the RAM that had the parity error.)

DCN-E-CSPERR "ram id" Control Store Parity Error
(see Chapter 4 Table 10)

NOTE 6 The Console uses the Message Pointer as an index and thus identifies and prints the approprate Error Message. Refer to the section on Console Messages for a detailed description of the message.

NOTE 7 The Console increments the count in the Microcode Scoreboard. This information is then displayed via the Console "SHOW UCODE" command.

NOTE 8 The Console stops the clock at T3 in order to correct FBox or MBox Control Store Parity Errors. The clock is stopped for FBox CS parity errors because the FBox uses the clocks differently than the rest of the system. The clock is stopped for MBox CS parity errors because the MCC Shift Path must be used to read the CS Address and Data. The other CS/DRAM parity errors can be corrected with clock running.

NOTE 9 The FBox does not latch the micro-address and data for FBox Dram parity errors. Therefore, the Console must reload the entire RAM. Note that the Console does not check to assure that the FDRAM loaded properly. Multiple FBox errors will eventually result in the VMS Machine Check Handler turning off the FBox.

NOTE 10 The Console reads the MCC Shift Path via the SDB. The Shift Path returns the the Micro-addesss and bad data.

NOTE 11 The Console steps the clock to the correct state for correcting FBA or FBM Control Store parity errors.

NOTE 12 The Console uses the SDB to read the Micro-address associated with the RAM error. If the Console encounters and error while reading the Micro-address it sets the UPCERR status flag. Evenually this will result in a non correctable error.

NOTE 13 The Console uses the SDB to read the Microword associated with the RAM error.

NOTE 14 The Console has a set of ECC tables for each location in each loadable RAM File. The Console uses the Micro-address it read via the SDB to look up the ECC character. If the Console is unable to find the ECC character it sets the NOECCD status flag. Evenually this will result in a non correctable error.

NOTE 15 After looking up the correct ECC character in the approprate table the console generates an ECC character for the Microword it read via the SDB. It then does an exclusive "or" of the two ECC characters. The result is the syndrome that will be used to correct the Microword.

NOTE 16 If the syndrome equals zero the console sets the SYNZRO status flag. This is a recoverable error condition.

NOTE 17 If the syndome indicates that a multiple bit parity error was detected the Console will set the MBTERR status flag. Evenually this will result in a non correctable error.

NOTE 18 If the syndrome is greater than the RAM size the Console will set the SYNGTR status flag. For example, if the syndrome indicates that IDRAM bit 22 is in error this flag will be set because the IDRAM is a 20 Microword. Evenually setting this flag will result in a non correctable error.

NOTE 19 The Console complements the bad bit the Microword, clears the retry count, and writes the corrected word to the RAM (via the SDB).

NOTE 20 If the Console was called to correct an EBox Control Store Parity Error it Loads the Control Store Data Register (CSDR) in the EBox. This will clear the EBox Control Store Parity Error condition.

NOTE 21 The Console reads the Microword back via the SDB and compares it to the corrected word it just wrote. If the write was successful the Console clears the CSPERR error status flag.

NOTE 22 If the Console was unable to write the Microword correctly the first time it will try 4 more times. If all attempts fail the Console will set the PCFAIL Error Status Flag. Evenually setting this flag will result in a non correctable error.

NOTE 23 With the exception of MBox Control Store Parity Errors, all non recoverable error conditions converge at this point. The Console sets the UNCORRECTABLE Error Status Flag and then sets the approprate "RAM STATE INVAILD" bit in to Microcode Scoreboard.

NOTE 24 Currently MBox Control Store Parity Errors are considered non correctable because they may leave the MBox in an unpredictable state. The Console sets the MUNREC status flag and increments the MBox CS parity error count in the Scoreboard.

NOTE 25 The Console restarts the CPU clock and then builds the CSES status word that will be sent to the VAX8600/8650. When the status word is built the console will set RBUF "DONE". If the Error Handling Microcode was in a loop waiting for the Console to correct a RAM, setting RBUF "DONE" will release the loop and allow the EHM to read the RBUF, build CSES, and continue processing the error.

NOTE 26 If the Console was called directly (by the hardware) to correct an EBox Control Store parity error then the Console will (via the SDB) generate "CSBR CLR ERR SERV REQ". This in turn clears the EBox Control Store Parity Error and generates "EBDA CSPE RST LAS CYC" which in turn starts the Error Handling Microcode at Vector 8.

NOTE 27 The Console then enters a timeout loop waiting for the EBox (EHM) to clear RBUF "DONE". The Console does not proceed until either the EHM clears RBUF "DONE" or the timeout count expires.

NOTE 28 At this point the Console checks to see if any of the non recoverable error status flags are set. If not the Console clears the Control Store Parity Error Flag (FL.CSP) and joins a common exit routine.

NOTE 29 If the Console was unable to correct the parity error it:

1. Sets the CSPERR status flag

2. Prints the following error message:
   DCN-E-CSPERR "ram id" Control Store Parity Error
   (see Chapter 4 Table 2)

3. Prints one of the following messages that explains why the parity error was non correctable (see Chapter 4 Table 10).

   ECR-E-MBTERR "ram id" Multiple bit Error, UNCORRECTABLE
   ECR-E-MUNREC MCS MBox Not Recoverable
   ECR-E-NOECCD "ram id" No ECC Data in Table
   ECR-E-PCFAIL "ram id" Correction Attempt Failed
   ECR-E-SYNGTR "ram id" Syndrome > RAM Size, UNCORRECTABLE
   ECR-E-SYNZRO "ram id" Syndrome = 0, UNCORRECTABLE
   ECR-E-UPCERR "ram id" Can't Read Box Address

4. Print the "GOOD" Data, "BAD" Data, and the status word sent to CSES.

NOTE 30 If the Console was unable to read the micro-address associated with the parity error it will print the exclusive "or" of the Good and Bad Data.

NOTE 31 This is a common exit point. If the parity error was corrected the console will clear (reset) the Keep Alive Timer and enable the the external interrupt mechinism. If the Console was unable to correct the error then it will leave the external interupts disabled. Eventually this will result in a Keep Alive Fail.

NOTE 32 Finally, the Console enables the Keep Alive Fail Timer and dismisses the interrupt. If the error was correctable no error message will have been printed and the EHM will continue to build the Machine Check Stack Frame and report the error to VMS. If the Console was unable to correct the error then a Keep Alive fail will occurr and the Console will Snap Shot the state of the system.

APPENDIX C

VMS MACHINE CHECK HANDLER (VMSMCK) FLOWS

This Appendix contains a set of flow charts for the VAX 86008650 System Control Block (SCB) Entry Vectors listed below. The flow charts describe how the VMS Machine Check Handler (MCK) evaluates each error condition and determines what action to take. The Table preceding the flow charts is included for reference purposes. It lists the Entry Vectors for the entire VAX 8600/8650 System Control Block.

Entry Vector	Name	Type	Code
004	Machine Check	Fault/Abort	1
054	Array Single Bit Error	Interrupt	1
058	SBI 0 Alert	Interrupt	1
05C	SBI 0 Fault	Interrupt	1
060	SBIA 0 Error	Interrupt	1

System Control Block for the VAX8600/8650

Entry Vector	Name	Type	Code
000	Unused		3
004	Machine Check	Fault/Abort	1
008	Kernel Stack Not Valid	Abort	1
00C	CPU Power Fail	Interrupt	1
010	Reserved DEC Opcodes and Privileged Instructions	Fault	0
014	Reserved Customer Opcodes	Fault	0
018	Reserved Operands	Fault/Abort	0
01C	Reserved Addressing Modes	Fault	0
020	Access Control Violation	Fault	0
024	Translation Not Valid	Fault	0
028	Trace Pending	Fault	0
02C	Breakpoint	Fault	0
030	Compatibility Mode	Fault	0
034	Arithmetic	Fault/Abort	0
038-03C	Unused	Trap/Fault	0
040	CHMK Opcode		3
044	CHME Opcode	Trap	0
048	CHMS Opcode	Trap	0
04C	CHMU Opcode	Trap	0
050	SBI 0 Silo Compare	Trap	0
054	Array Single Bit Error	Interrupt	1
058	SBI 0 Alert	Interrupt	1
05C	SBI 0 Fault	Interrupt	1
060	SBIA 0 Error	Interrupt	1
064	SBI 0 Fail	Interrupt	1
068-080	Unused	Interrupt	1
084	Software Request 01		3
088	Software Request 02 or AST Delivery	Interrupt	1
08C	Software Request 03	Interrupt	0
090	Software Request 04	Interrupt	1
094	Software Request 05	Interrupt	1
098	Software Request 06	Interrupt	1
09C	Software Request 07	Interrupt	1
0A0	Software Request 08	Interrupt	1
0A4	Software Request 09	Interrupt	1
0A8	Software Request 0A	Interrupt	1
0AC	Software Request 0B	Interrupt	1
0B0	Software Request 0C	Interrupt	1
0B4	Software Request 0D	Interrupt	1
0B8	Software Request 0E	Interrupt	1
0BC	Software Request 0F	Interrupt	1
0C0	Interval Timer	Interrupt	1
0C4-0EC	Unused		3
0F0	Console Block Storage	Interrupt	1
0F4	Unused		3
0F8	Console Terminal Receive	Interrupt	1
0FC	Console Terminal Transmit	Interrupt	1
100-13C	SBI 0 Req 4/Unibus BR 4	Interrupt	1
140-17C	SBI 0 Req 5/Unibus BR 5	Interrupt	1
180-1BC	SBI 0 Req 6/Unibus BR 6	Interrupt	1
1C0-1FC	SBI 0 Req 7/Unibus BR 7	Interrupt	1

System Control Block for the VAX 8600/8650 (cont.)

Entry Vector	Name	Type	Code
200-24C	Unused		3
250	SBI 1 Silo Compare	Interrupt	1
254	SBI 1 Fail	Interrupt	1
258	SBI 1 Alert	Interrupt	1
25C	SBI 1 Fault	Interrupt	1
260	SBIA 1 Error	Interrupt	1
264-2FC	Unused		3
300-33C	SBI 1 Req 4/Unibus BR 4	Interrupt	1
340-37C	SBI 1 Req 5/Unibus BR 5	Interrupt	1
380-3BC	SBI 1 Req 6/Unibus BR 6	Interrupt	1
3C0-3FC	SBI 1 Req 7/Unibus BR 7	Interrupt	1

Vector	Name	Type	Code
400-5FC	Unused		3

Vector	Name	Type	Code
600-7FC	Unused		3

Code	Description
0	Service on Kernel Stack (or if selected Interrupt Stack)
1	Service on Interrupt Stack
2	Service via WCS (Halt if there is no WCS)
3	Reserved

* Note: The Interrupt Vectors for SBI-1 are offset from SBI-0 by a page (200 words). The offset is added by the EBox Interrupt Handling Microcode after it examines CSLINT <22:21> and determines which Adapter needs service.

FLOW CHART CODING DESCRIPTION

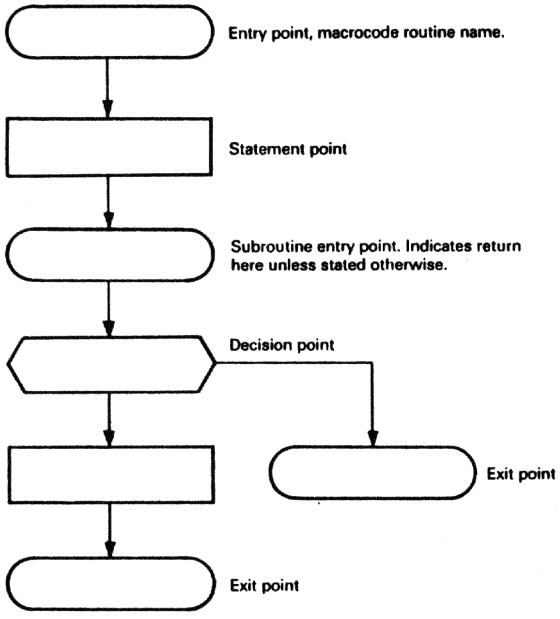
The following flowcharts contain terminal points, decision points, modified decision blocks, and statement blocks.

The terminal points at the top of a flowchart contain the macrocode name designating the entry point for that routine.

The terminal points within a routine are subroutine entry points where some action takes place and then the routine resumes processing through the flowchart from that point.

Terminal points at the end of a routine (or decision point) are exit points.

Rectangular blocks within the routines are statement points.



CAUTION
These flowcharts apply to VMS version 4.7 and are subject to change.

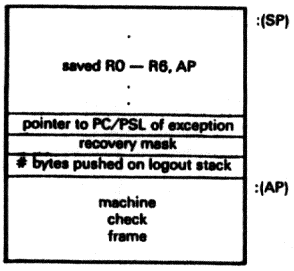
MR-1087-0775

VMS Machine Check Handler Flow Chart (Part 1 of 50)

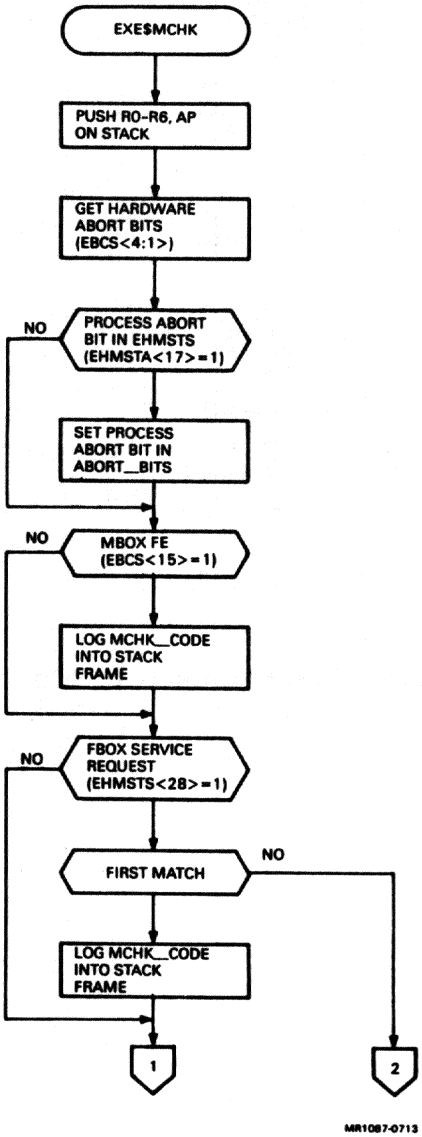
EXE$MCHK

All machine checks are vectored to this entry point.

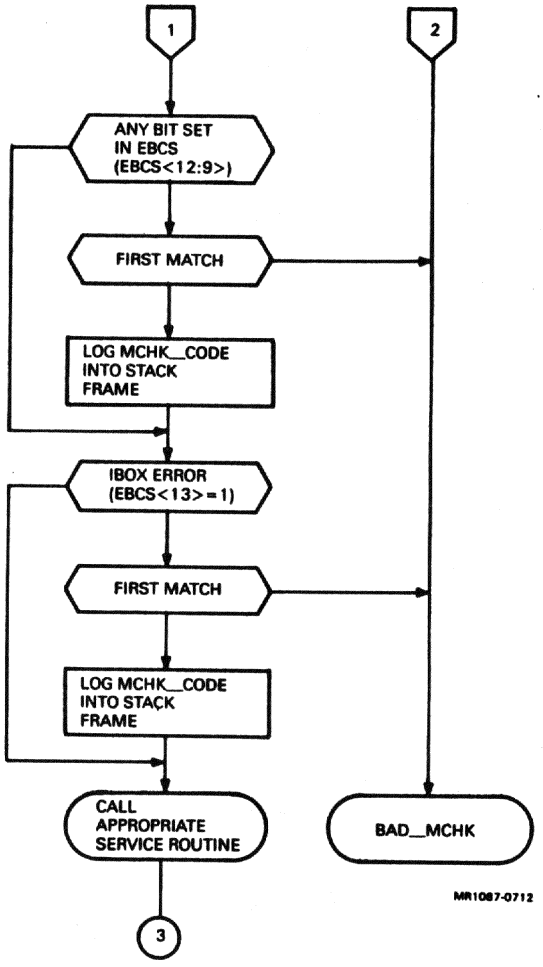
As soon as the machine check handler is invoked, it sets up the stack as follows:



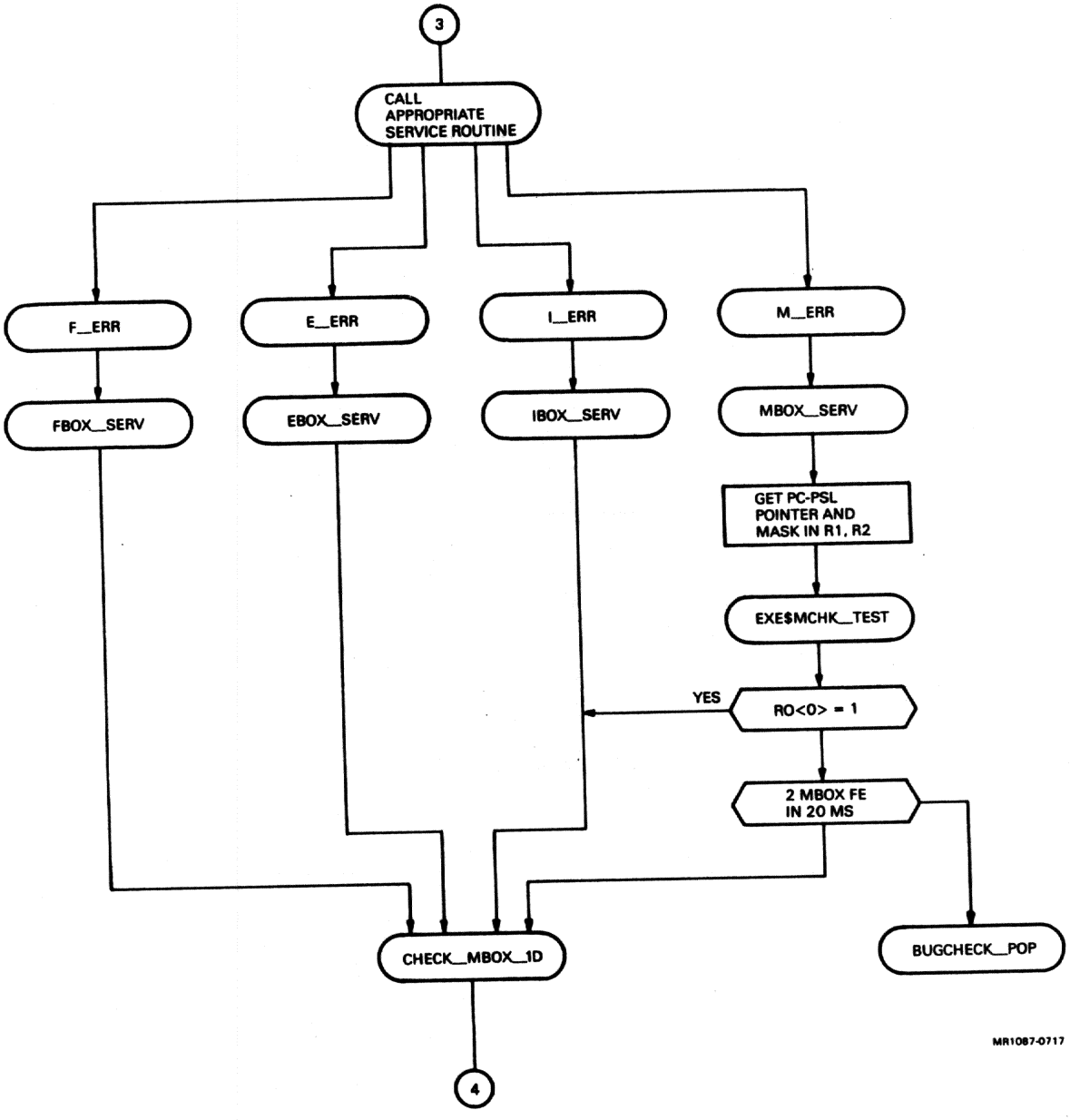
AP points to the beginning of the machine check log on the stack. Two longwords are immediately pushed on top of the machine check log, and are referenced as negative offsets from AP. These two longwords are input arguments to a routine that is called to check for a user-declared machine check recovery block. The routine which is called before bugchecking expects the mask and the pointer to the exception PC/PSL to be on top of the machine check log on the stack.



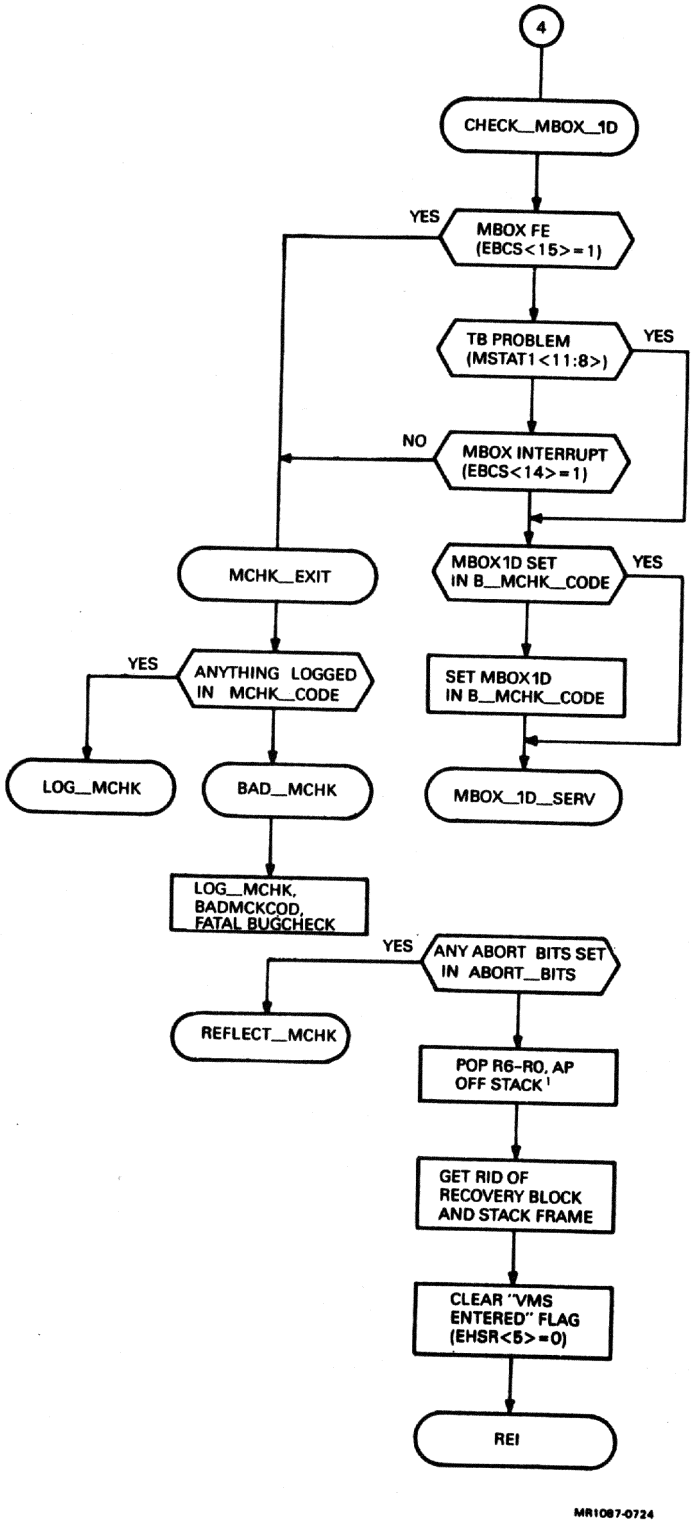
VMS Machine Check Handler Flow Chart (Part 2 of 50)



VMS Machine Check Handler Flow Chart (Part 3 of 50)



VMS Machine Check Handler Flow Chart (Part 4 of 50)



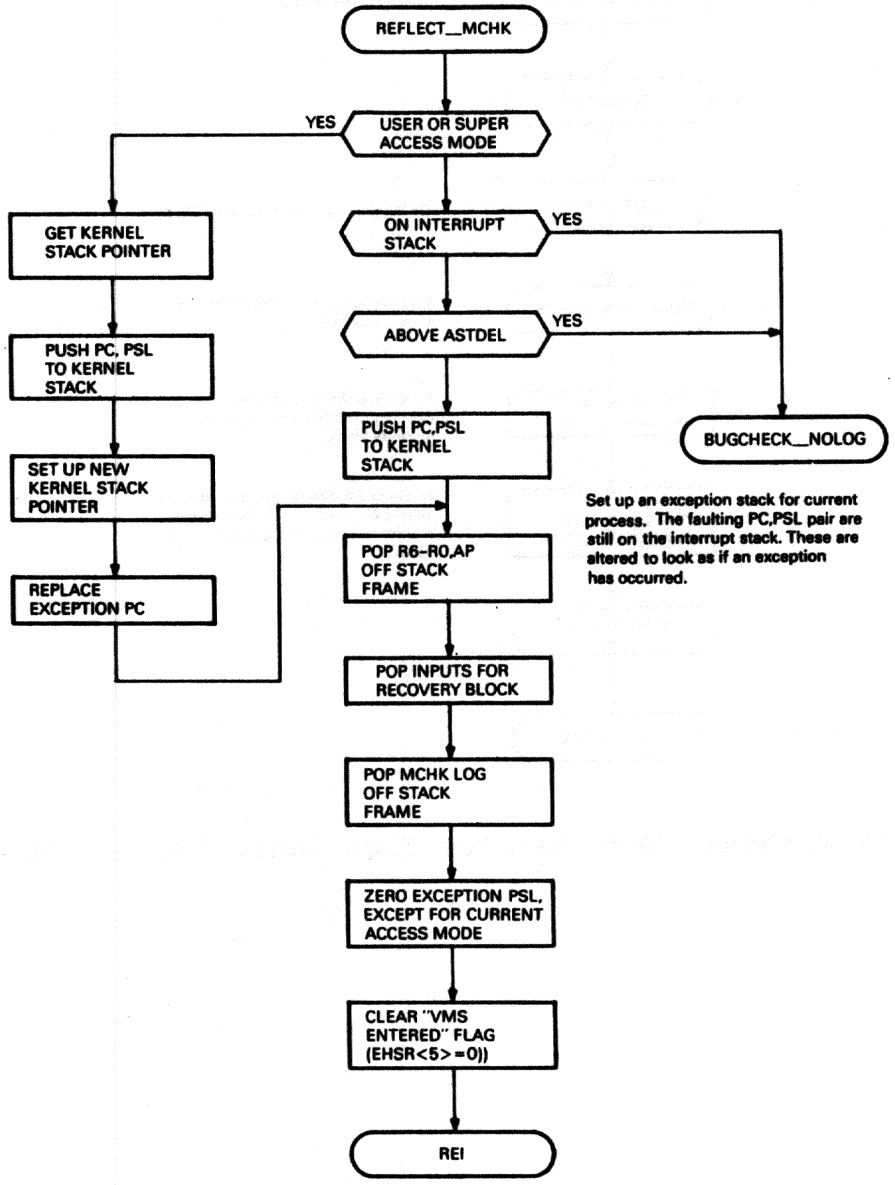
VMS Machine Check Handler Flow Chart (Part 5 of 50)

REFLECT_MCHK:

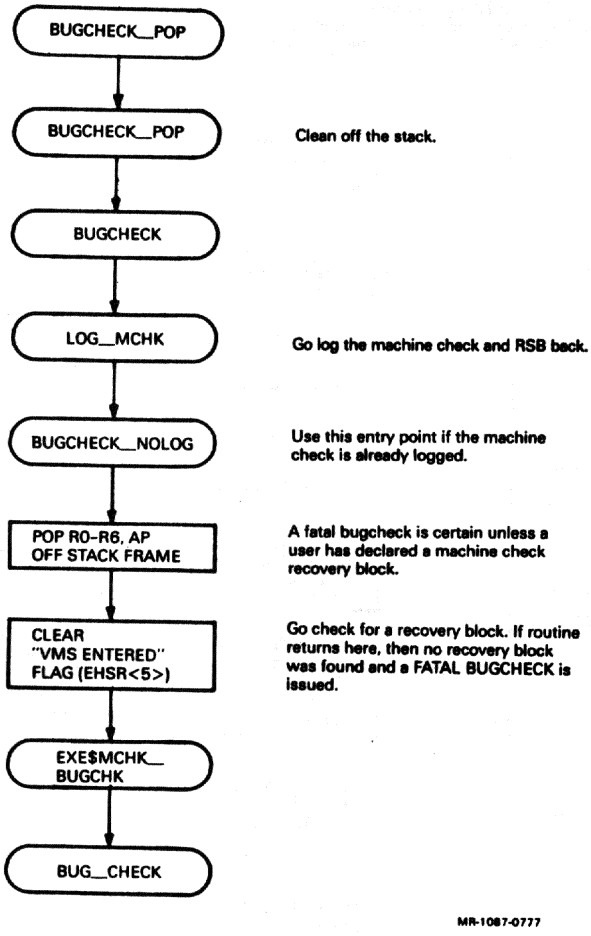
This code is entered if the machine check was fatal. It determines if it was just fatal to the process which caused it (current process is in USER or SUPER mode), or if it was fatal to the entire system (current process is in EXEC or KERNEL mode).

If current process is in USER or SUPER mode, set up an exception on user's stack and REI to it. If current process is in EXEC or KERNEL mode and above ASTDEL or on ISP issue a fatal bugcheck.

STACK CONTENTS:
00(SP): saved R0,R1,R2,R3,R4,R5,R6,AP
20(SP): 2 longword inputs for recovery block check
28(SP): (also AP) machine check log — 1st longword is a byte count.



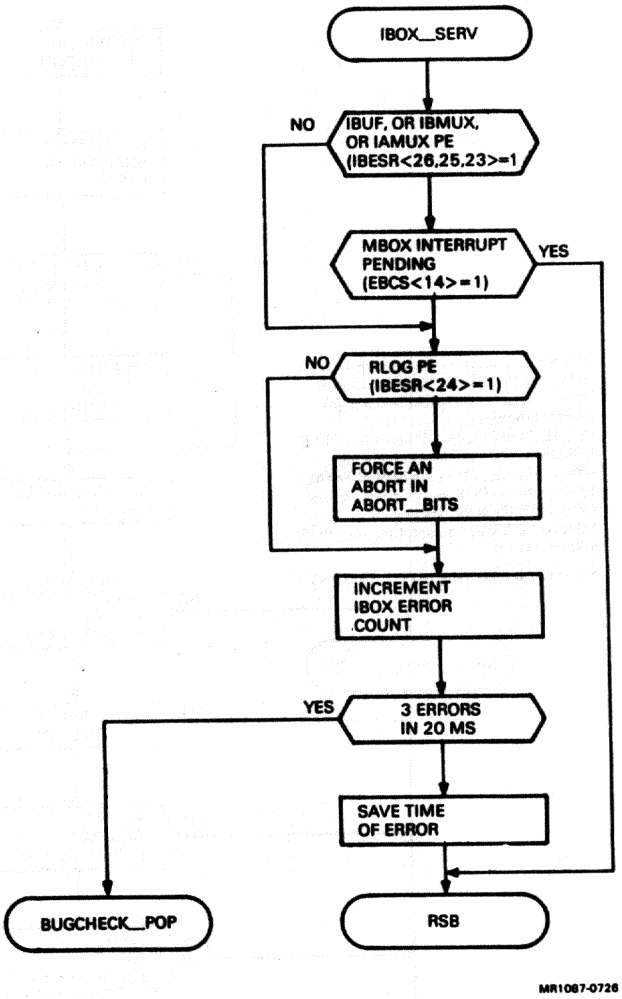
VMS Machine Check Handler Flow Chart (Part 6 of 50)



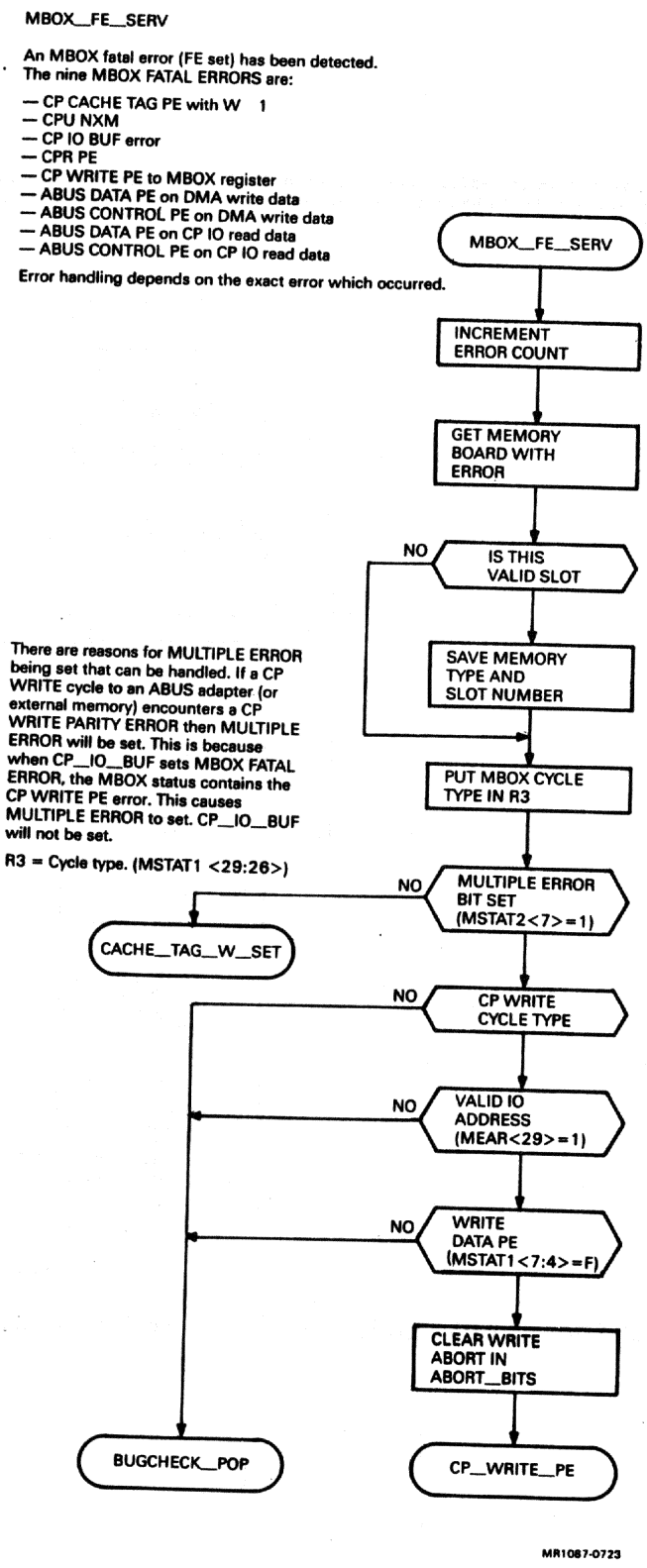
VMS Machine Check Handler Flow Chart (Part 7 of 50)

IBOX_SERV

An IBOX error has been detected. All Ibox errors are all treated alike. The only exception is that the abort flag must be set on RLOG_PE since instruction backout may have failed. This is cause for a bugcheck if they are too frequent.



VMS Machine Check Handler Flow Chart (Part 8 of 50)



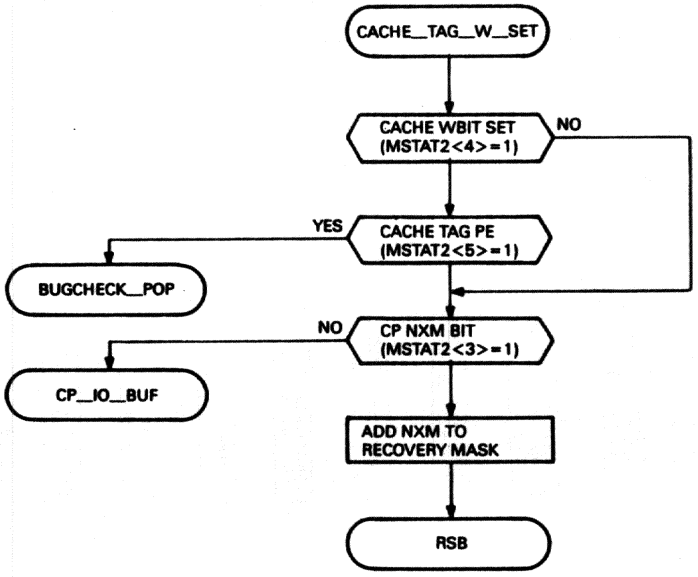
VMS Machine Check Handler Flow Chart (Part 9 of 50)

CACHE__TAG__W__SET

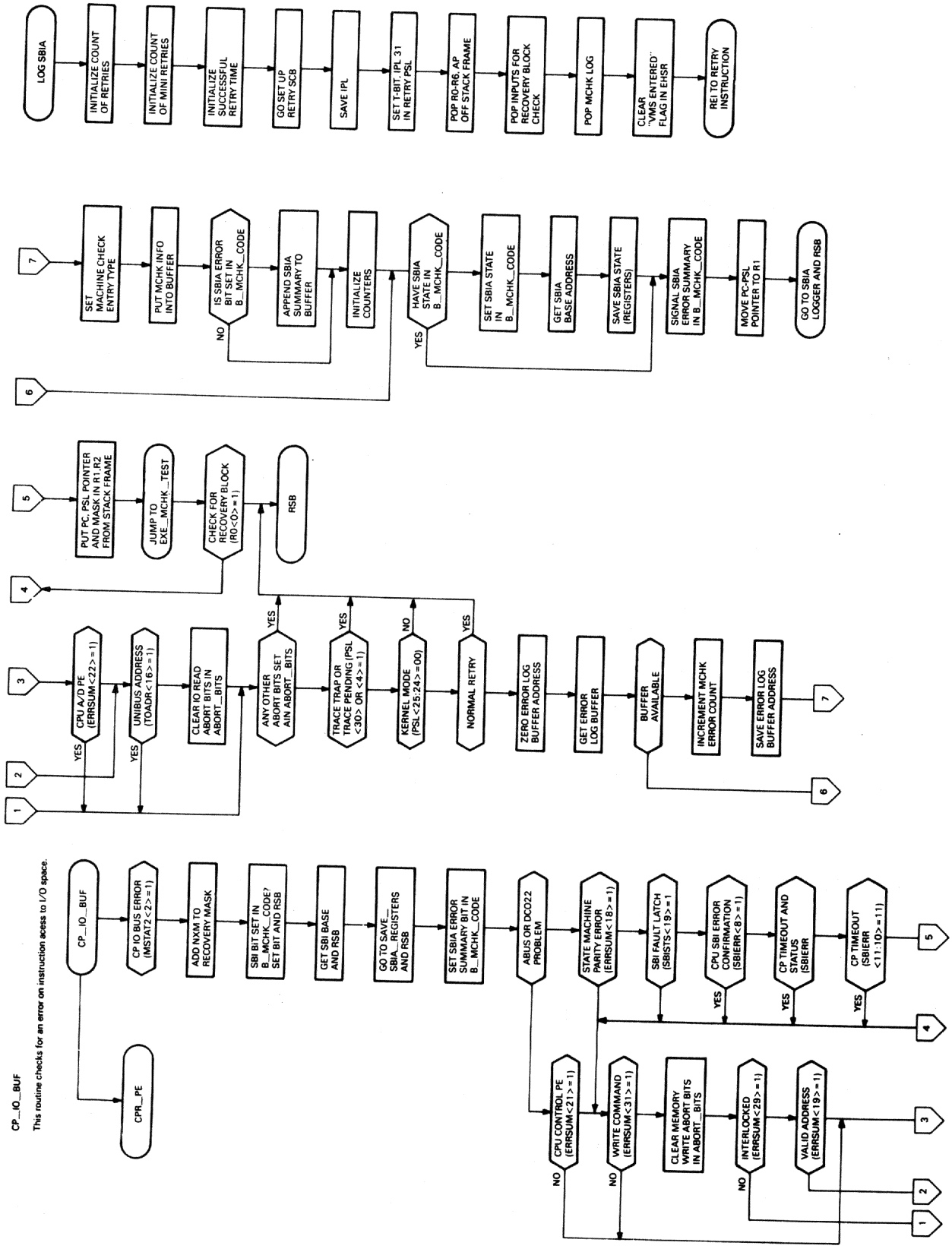
This is a check for CACHE TAG PARITY errors with the written bit set. When set, it is unknown where the modified data belongs. For this reason system rather than user must BUG__CHECK.

The latter part of this routine checks to see if it was a CP reference to a nonexistent memory. More than one error of this type is fatal.

R3 = CYCLE TYPE (MSTAT1 <29:26>)



VMS Machine Check Handler Flow Chart (Part 10 of 50)



MR1087-0721

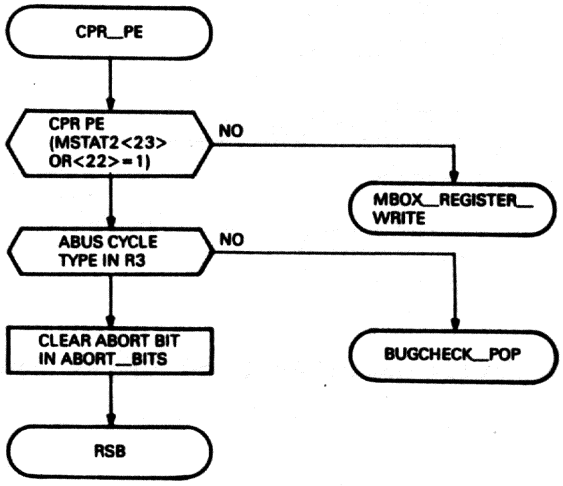
VMS Machine Check Handler Flow Chart (Part 11 of 50)

CPR_PE

When the MBOX grants a CPU port's request for access, the Memory Control Field (MCF) lines for that port encode the request type. The MCF can be viewed as an "OPCODE" which the MBOX executes to address the MBOX Cycle Parameter RAMs (CPR). CPR parity error is an MBOX FATAL ERROR because when one occurs on a CPU request an unpredictable operation takes place in the MBOX.

NOTE: DMA does not use the CPRs although it is possible for whatever CPR bits are being read out during a DMA micro-routine to have bad parity resulting in an MBOX FATAL ERROR trap. This type of CPR parity error is recoverable.

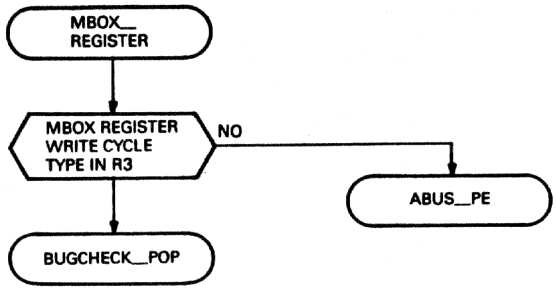
R3 = CYCLE TYPE (MSTAT1 <29:26>)



VMS Machine Check Handler Flow Chart (Part 12 of 50)

MBOX__REGISTER__WRITE

This routine looks for an error writing an MBOX register. Recovery is not possible because MBOX operations are no longer predictable at this point.



VMS Machine Check Handler Flow Chart (Part 13 of 50)

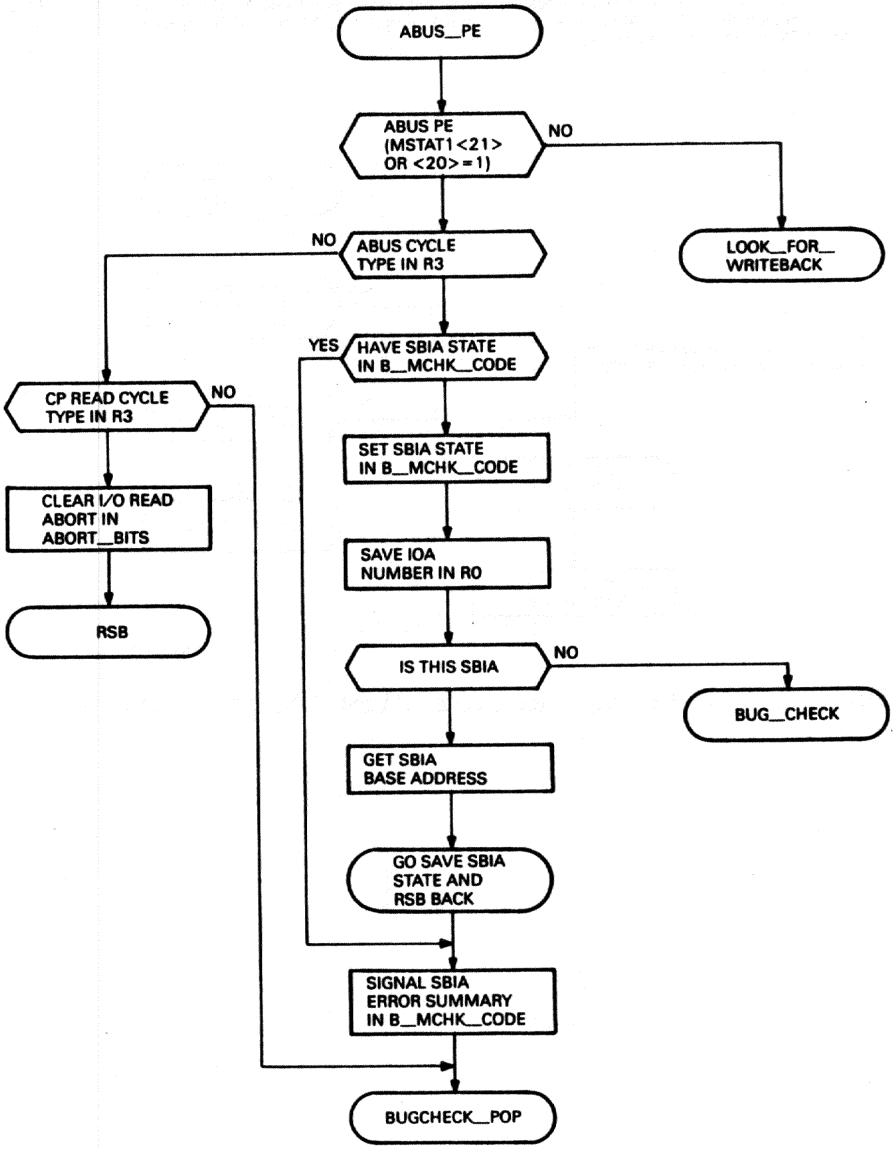
ABUS__PE

Abus parity errors detected by MBOX FATAL ERR come from two sources. DMA WRITE DATA and CP READ DATA from an I/O address.

Parity errors during the Command/Adress cycle vector the MBOX micro code to a loop that will cause KAF. An error on a C/A that is stalled because the target array is busy can cause an intended CONTROL STORE PARITY ERROR. If the stall continues beyond one cycle, the micro stack becomes corrupt. The KAF still occurs but the indications may not be ABUS.

Recovery is only possible from ABUS PE that occur for ABUS DATA cycles or CP read requests; i.e., a CP read of an I/O operation.

R3 = MSAT1 <29:26>

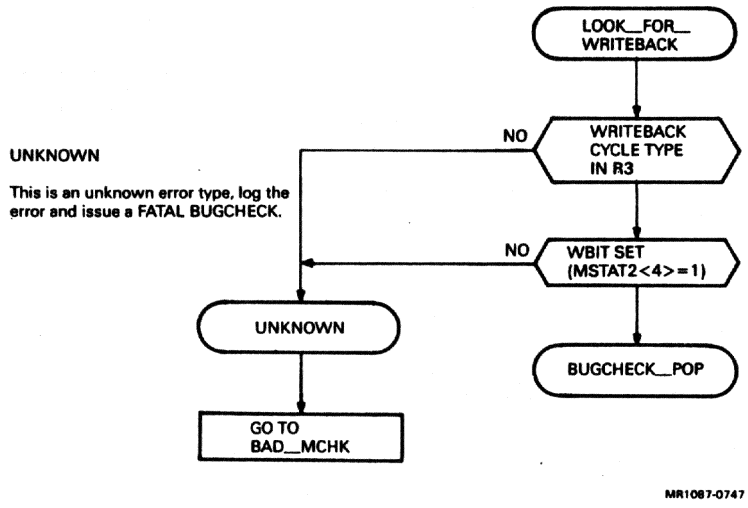


VMS Machine Check Handler Flow Chart (Part 14 of 50)

LOOK_FOR_WRITEBACK

At this point MBOX FE is set but a valid reason (error bit set) has not been found. For CACHE TAG PARITY ERRORS with the W bit set during WRITEBACK, the CACHE TAG PARITY ERROR bit (MSATA1 <6>) may not be set. Check for a WRITEBACK cycle and the W bit set.

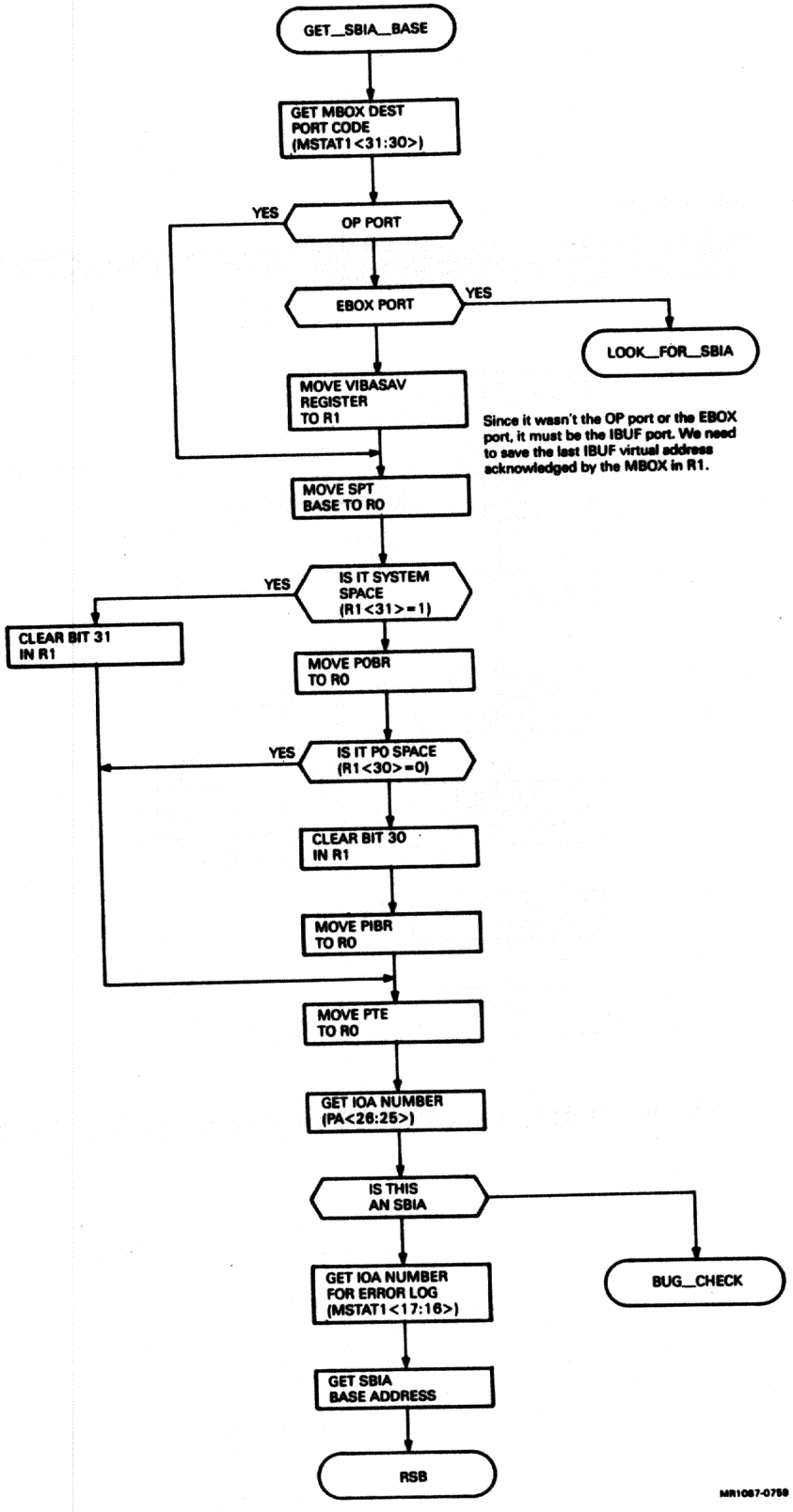
R3 = CYCLE TYPE



VMS Machine Check Handler Flow Chart (Part 15 of 50)

GET_SBIA_BASE

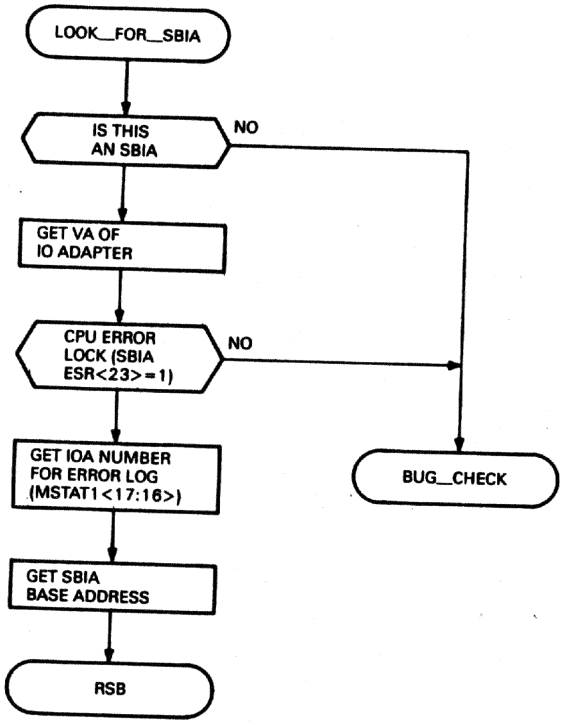
Calculate the base address of the SBIA involved with this machine check. If not SBIA then BUG_CHECK.



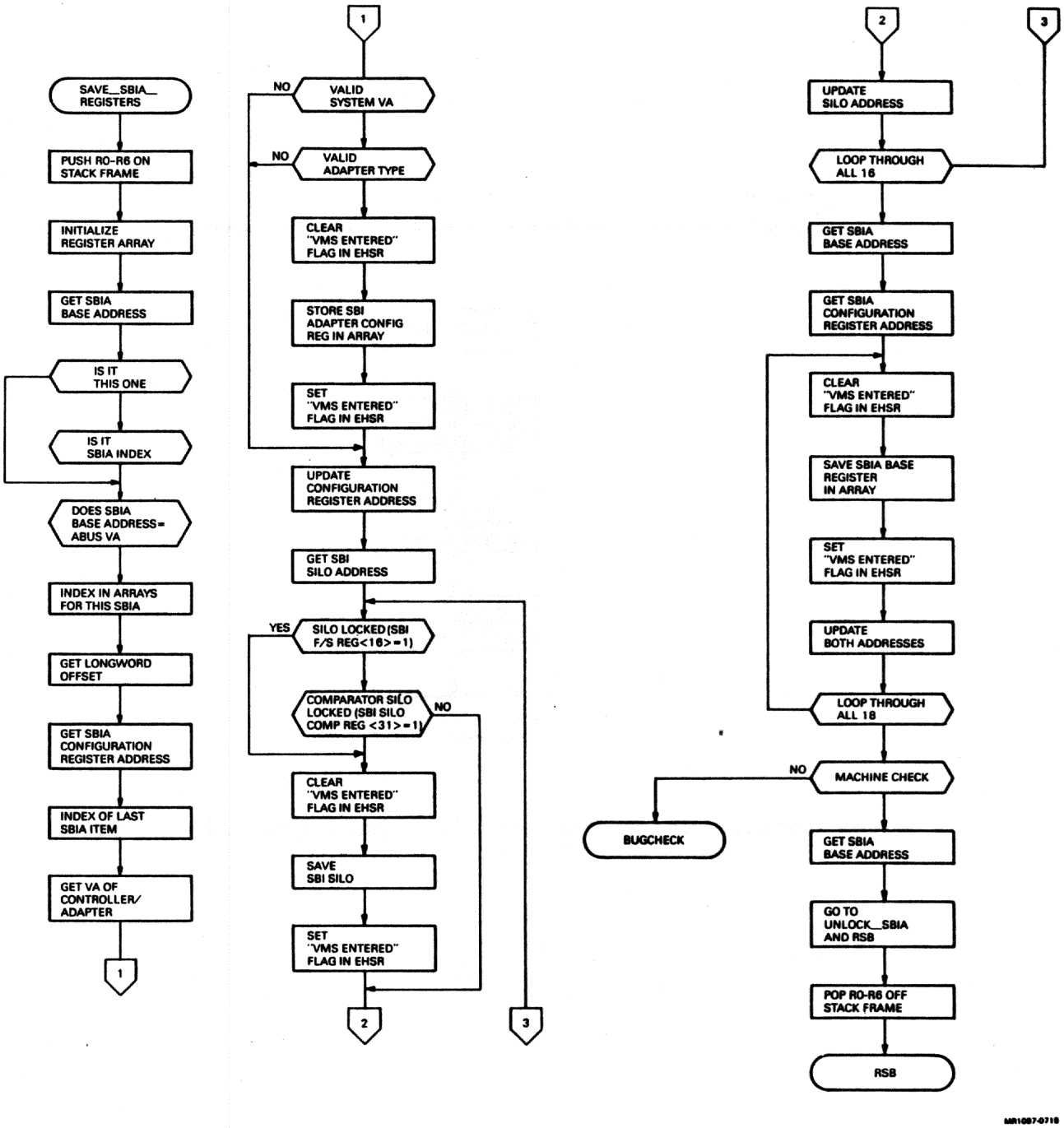
VMS Machine Check Handler Flow Chart (Part 16 of 50)

LOOK_FOR_SBIA

For CP_IO_BUF errors when the requesting port is EBOX the SBIA summary registers are polled looking for CPU BUF ERROR LOCK. This is required because the EBOX does PHYSICAL and VIRTUAL reads of IO space. It is not possible to determine if the reference was either PHYSICAL or VIRTUAL.



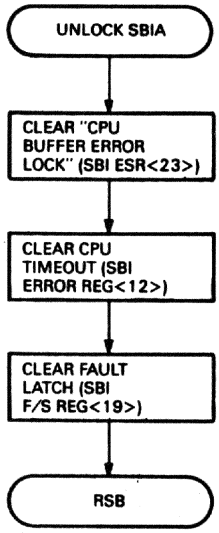
VMS Machine Check Handler Flow Chart (Part 17 of 50)



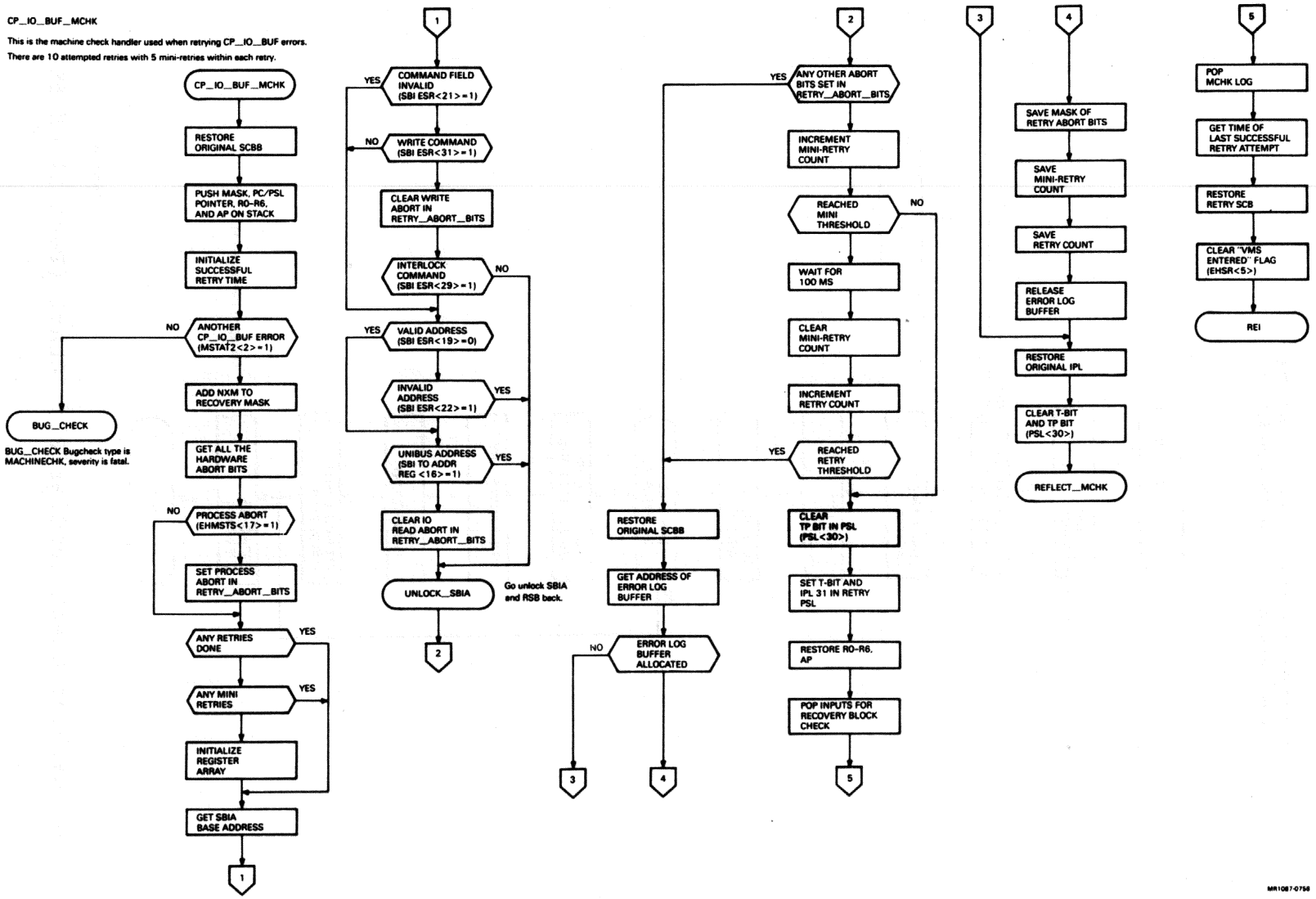
VMS Machine Check Handler Flow Chart (Part 18 of 50)

UNLOCK SBIA

Unlock the SBIA registers after an error. Clear all the error locks and the CPU timeout bit and the fault latch.



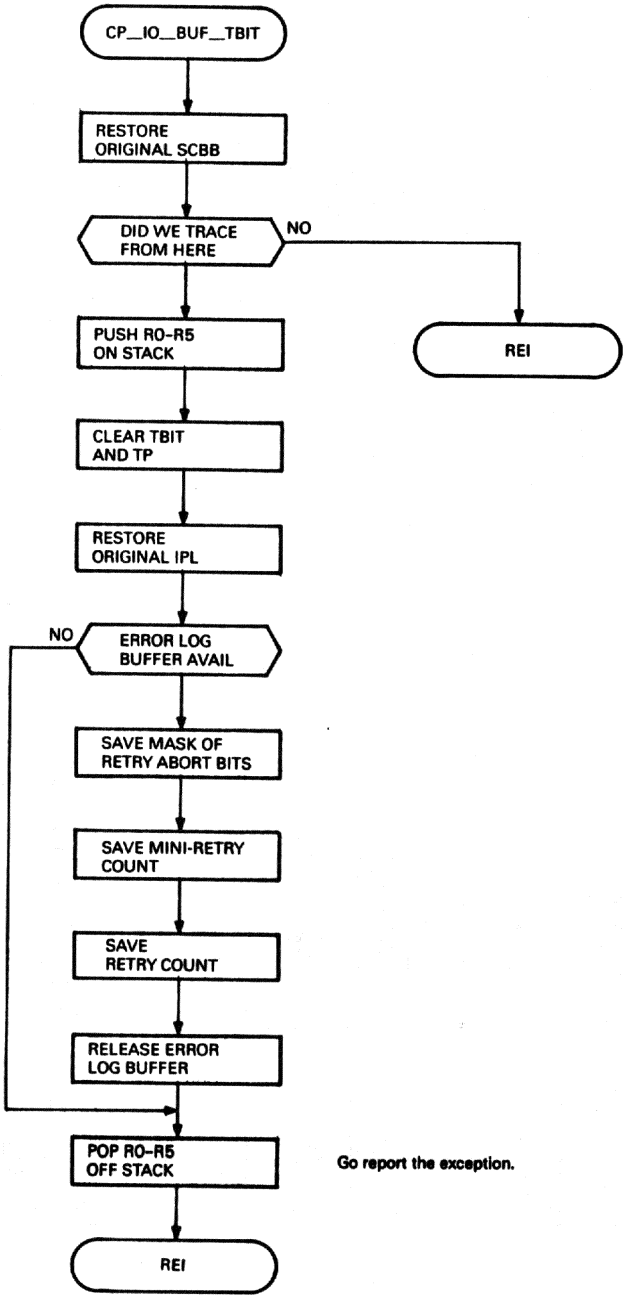
VMS Machine Check Handler Flow Chart (Part 19 of 50)



VMS Machine Check Handler Flow Chart (Part 20 of 50)

CP_IO_BUF_TBIT

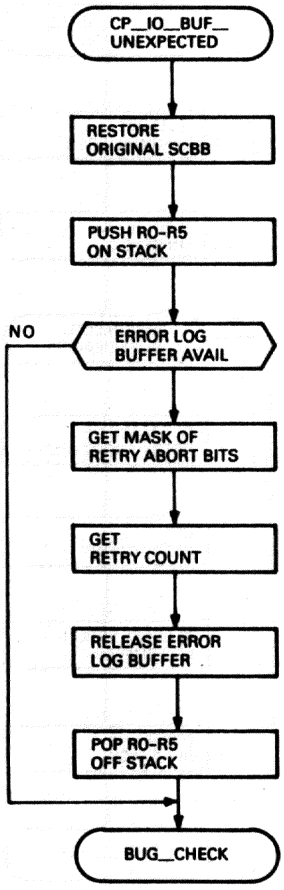
This is the trace handler used when retrying CP_IO_BUF errors.
There are 10 retries with 5 mini-retries within each retry.



VMS Machine Check Handler Flow Chart (Part 21 of 50)

CP__IO__BUF__UNEXPECTED

This is the unexpected exception handler used when retrying CP__IO__BUF errors.



BUGCHECK type is MACHINECHK, severity is fatal.

MR1087-0739

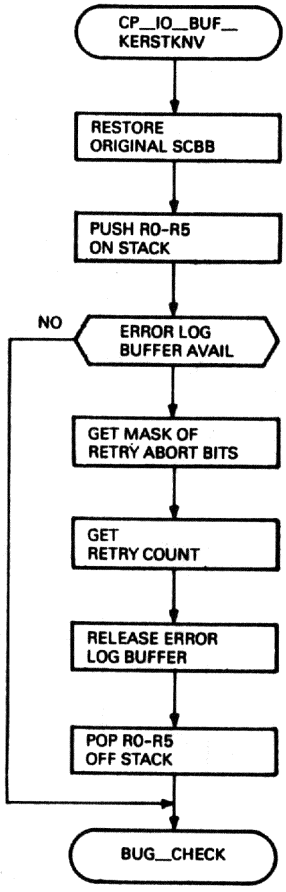
VMS Machine Check Handler Flow Chart (Part 22 of 50)

CP__IO__BUF__KERSTKNV:

This is the Kernel Stack Not Valid exception handler used when retrying CP__IO__BUF errors.

INTERRUPT stack on entry is:

00(SP) = EXCEPTION PC
04(SP) = EXCEPTION PSL



BUGCHECK type is KRNLSTAKNV, severity is fatal.

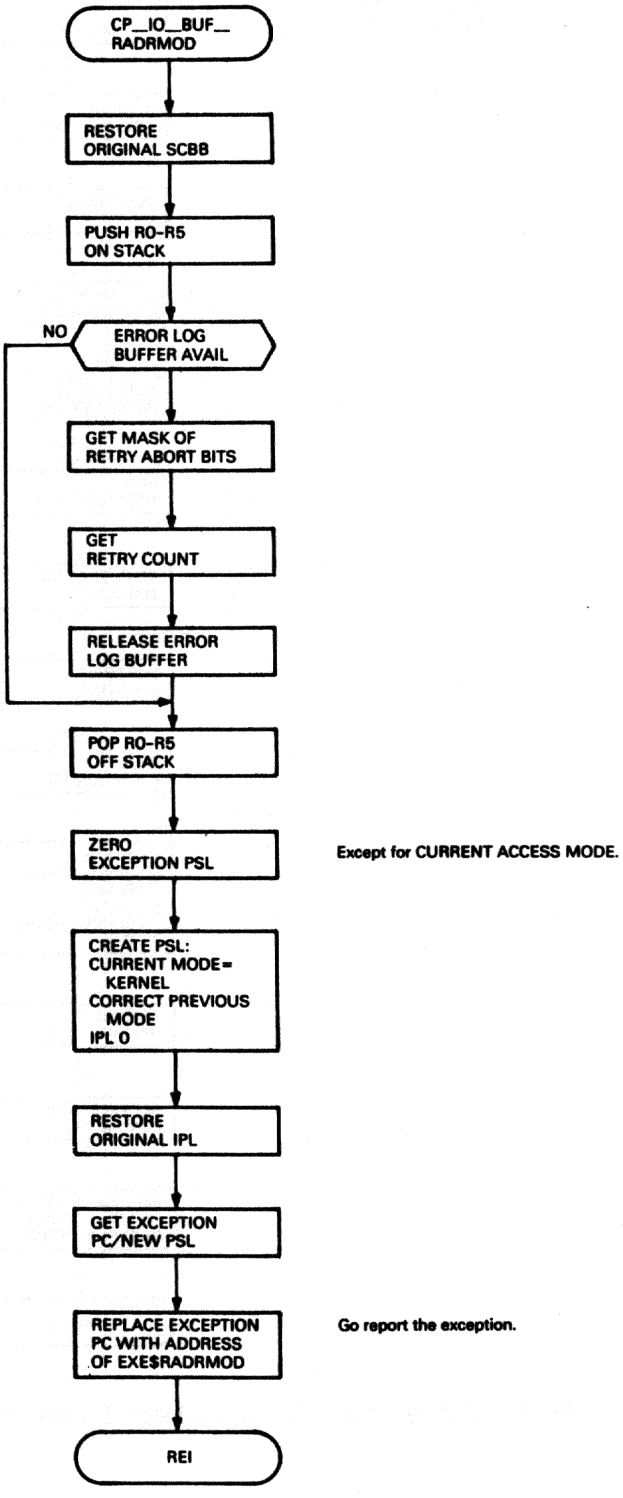
MR1087-0718

VMS Machine Check Handler Flow Chart (Part 23 of 50)

CP__IO__RADRMOD

This is the Reserved Addressing Mode Fault exception handler used when retrying CP__IO__BUF errors.

KERNEL stack on entry is:
00(SP) = EXCEPTION PC
04(SP) = EXCEPTION PSL



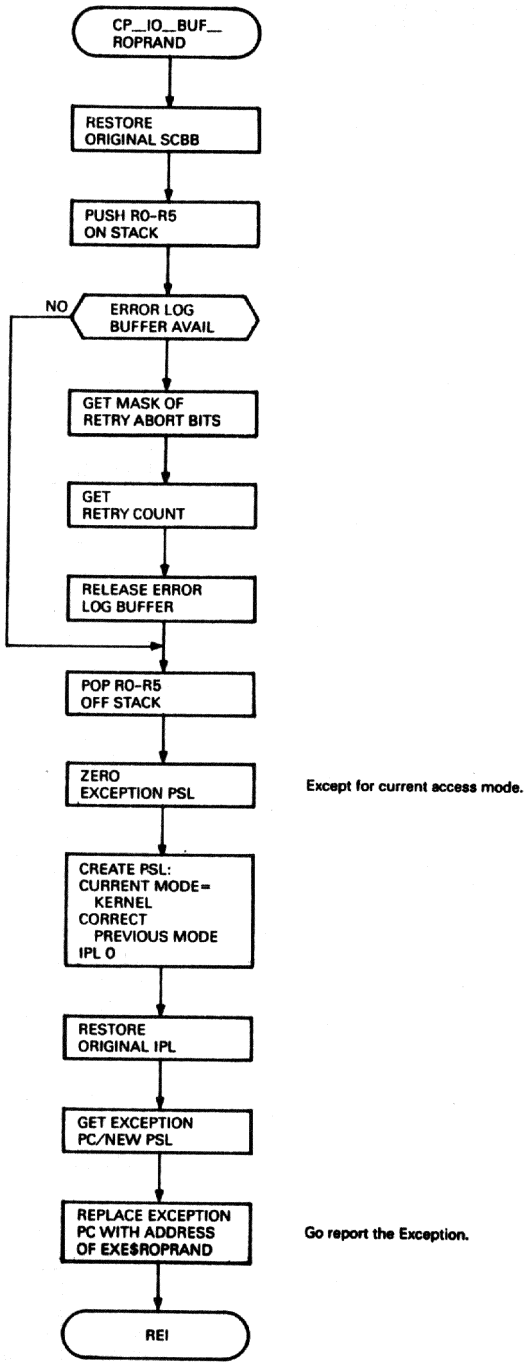
VMS Machine Check Handler Flow Chart (Part 24 of 50)

CP__IO__BUF__ROPRAND

This is the Reserved Operand Fault exception handler used when retrying CP__IO__BUF errors.

KERNEL stack on entry is:

00(SP) = EXCEPTION PC
04(SP) = EXCEPTION PSL

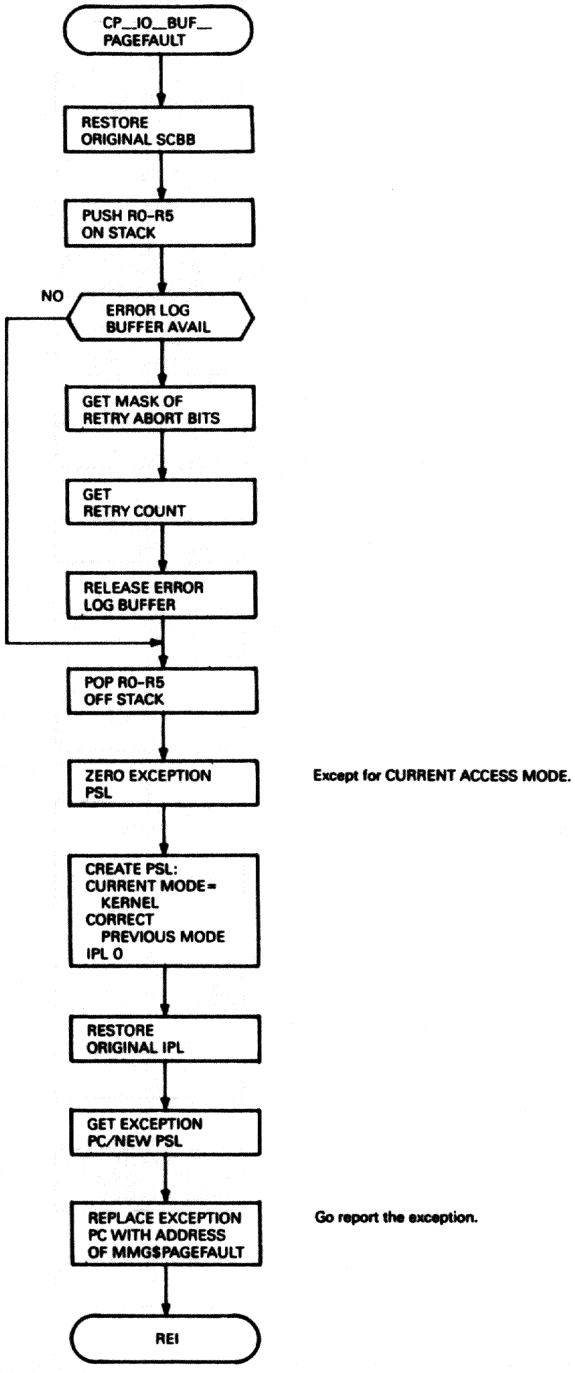


VMS Machine Check Handler Flow Chart (Part 25 of 50)

CP__IO__BUF__PAGEFAULT

This is the Translation Not Valid Fault exception handler used when retrying CP__IO__BUF errors.

KERNEL stack on entry is:
00(SP) = FAULT PARAMETER LONGWORD
04(SP) = SOME VIRTUAL ADDRESS IN FAULTING PAGE
08(SP) = EXCEPTION PC
12(SP) = EXCEPTION PSL

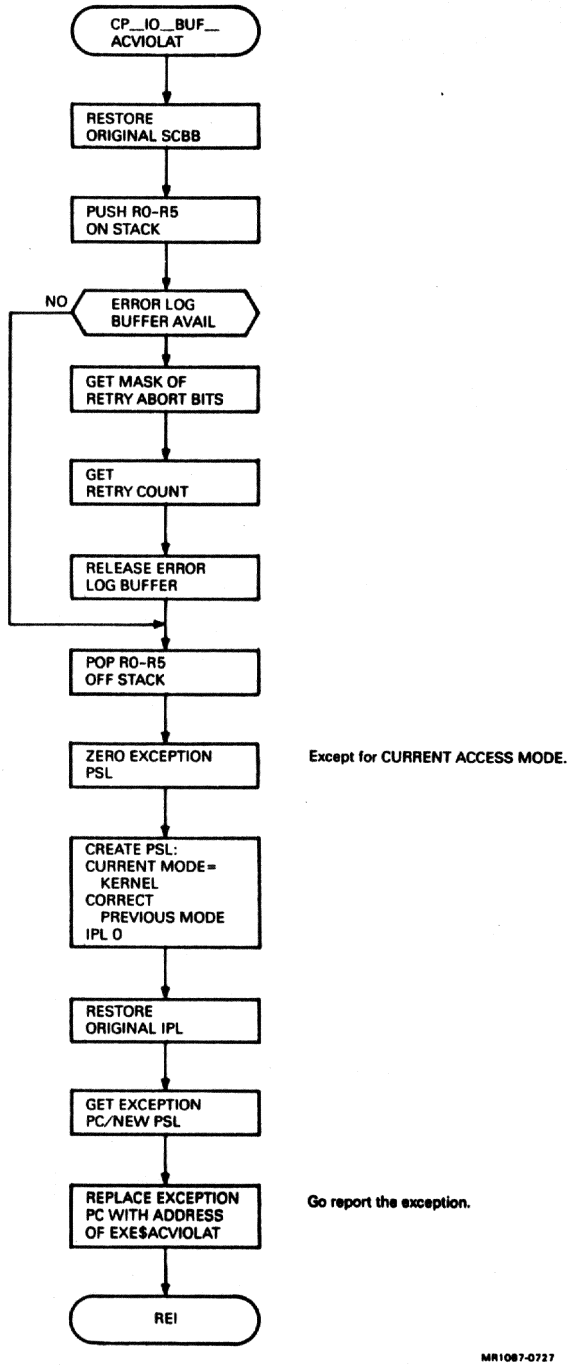


VMS Machine Check Handler Flow Chart (Part 26 of 50)

CP__IO__BUF__ACVIOLAT

This is the Access Violation Fault exception handler used when retrying CP__IO__BUF errors.

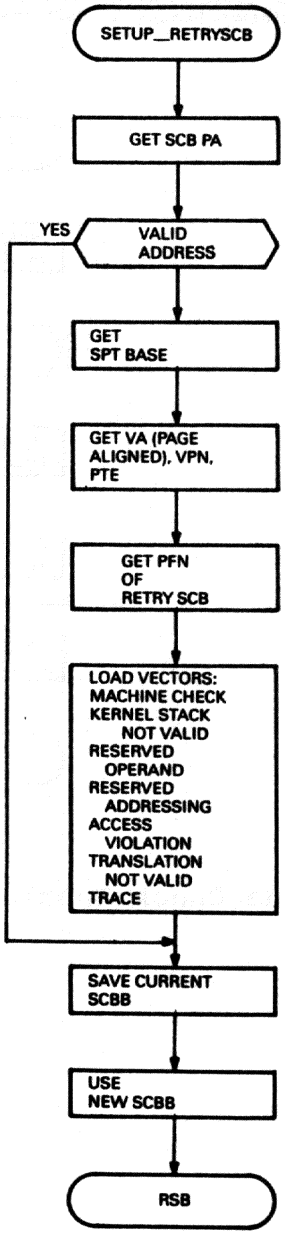
KERNEL stack on entry is:
00(SP) = ACCESS VIOLATION REASON MASK
04(SP) = ACCESS VIOLATION VIRTUAL ADDRESS
08(SP) = EXCEPTION PC
12(SP) = EXCEPTION PSL



VMS Machine Check Handler Flow Chart (Part 27 of 50)

SETUP_RETRYSCB

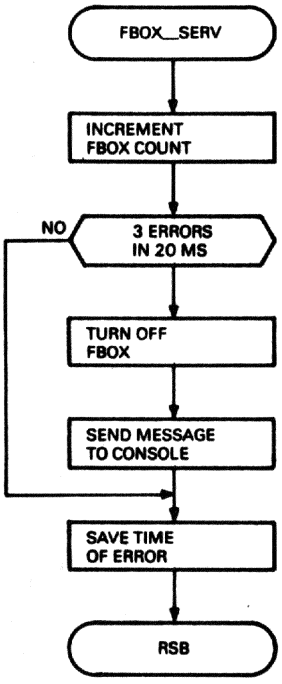
This routine sets up an appropriate SCB for retries of CP_IO_BUFFER errors



VMS Machine Check Handler Flow Chart (Part 28 of 50)

FBOX_SERV

FBOX errors that are too frequent will cause the MACHINE CHECK HANDLER to turn off the FBOX and report this to the console. This is not a reason to bugcheck as EBOX can assume floating point functions.

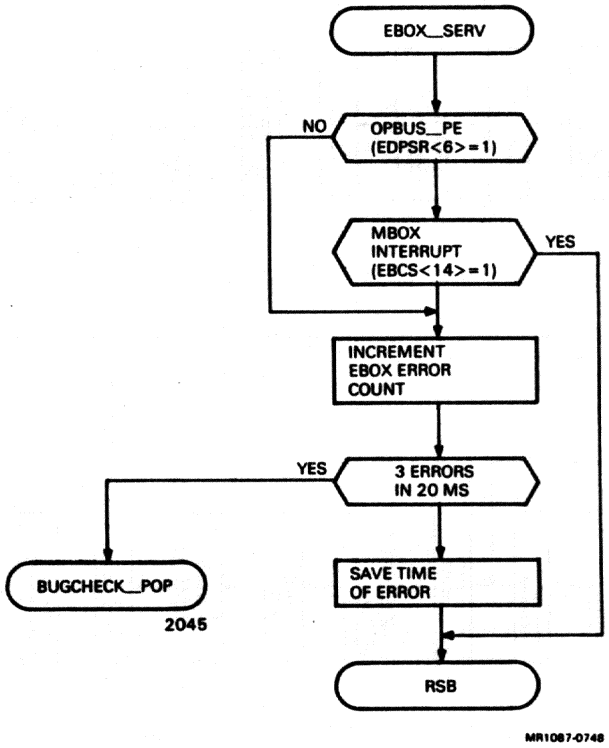


MR1087-0734

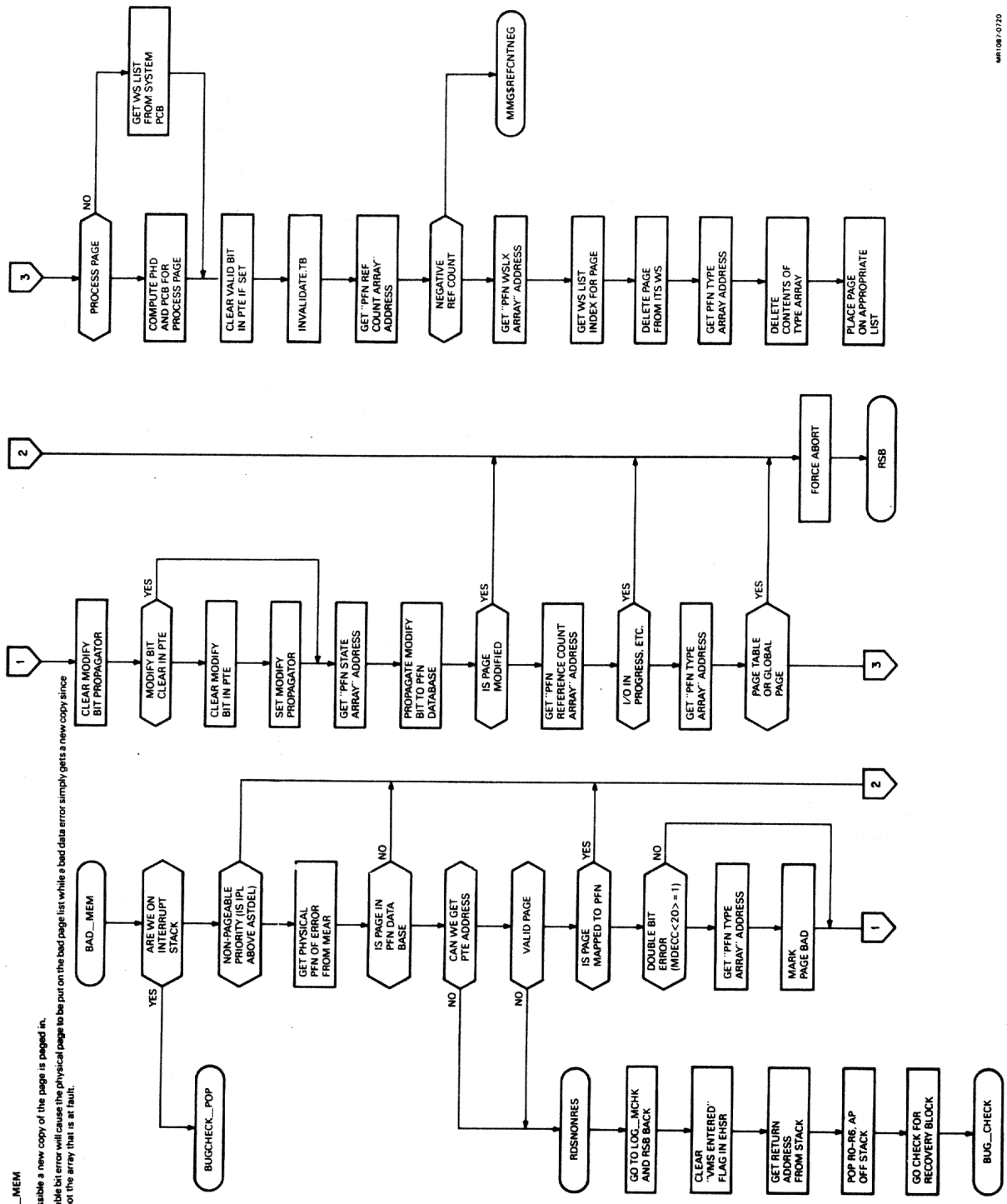
VMS Machine Check Handler Flow Chart (Part 29 of 50)

EBOX_SERV

An EBOX error has been detected. These are all treated alike. This is cause for a bugcheck if they are too frequent.



VMS Machine Check Handler Flow Chart (Part 30 of 50)



VMS Machine Check Handler Flow Chart (Part 31 of 50)

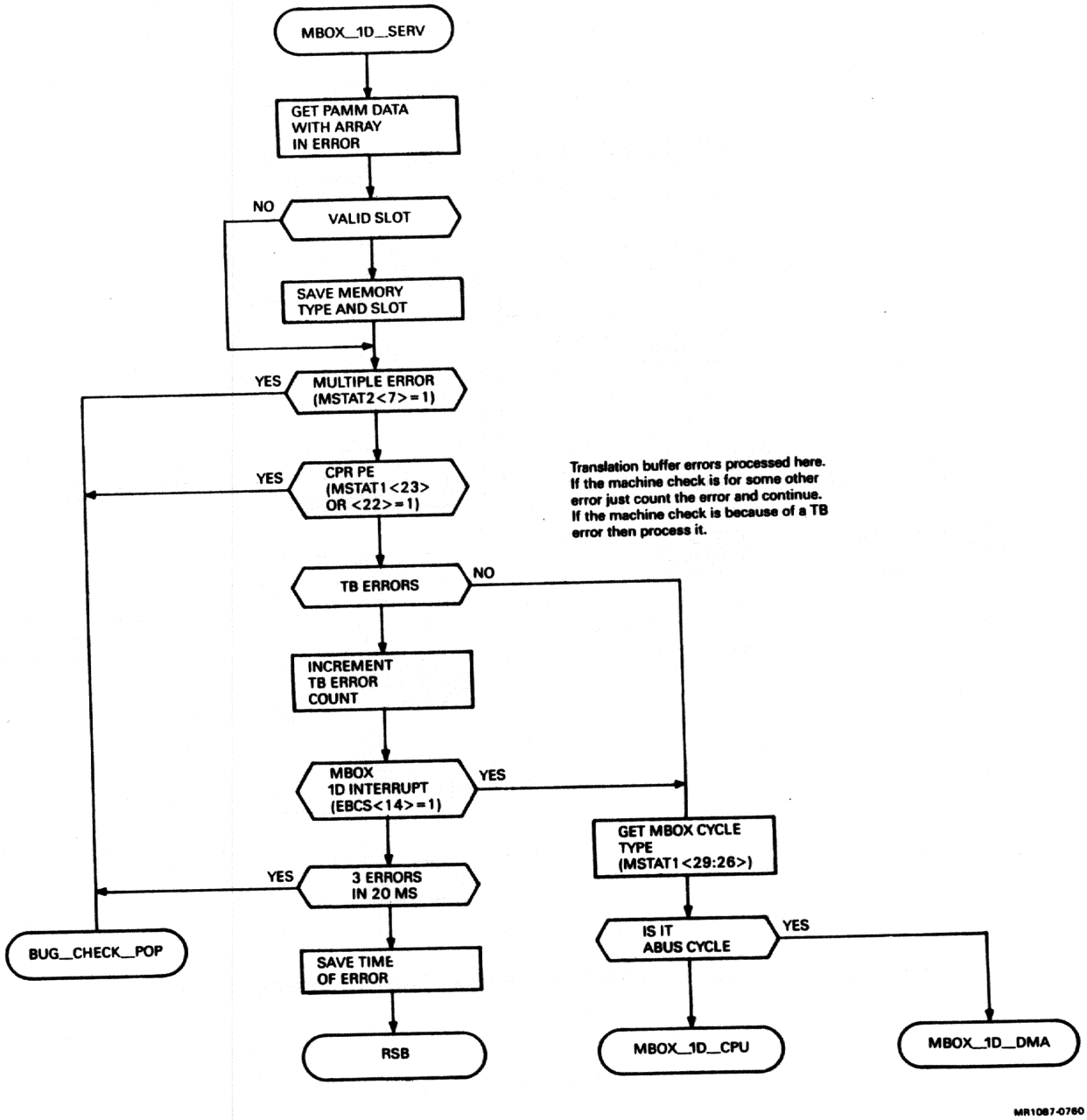
MBOX 1D

An MBOX error was reported when IPL dropped below 1D. These are not directly related to instruction execution. From here we will dispatch to either asynchronous CPU errors or DMA errors.

The eight MBOX errors reported via the interrupt at 1D mechanism are

— CPR PARITY ERROR (This is a fail safe, normally reported via MBOX FE)
— ECC ERROR (CP or DMA)
— CACHE W BIT PARITY ERROR (CP or DMA)
— CACHE TAG PARITY ERROR (CP or DMA)
— CACHE DATA PARITY ERROR (CP or DMA)
— CP WRITE PARITY ERROR (CP reference)
— ABUS BAD DATA CODE (CP reference)
— MBOX DETECTED ABUS PARITY ERROR (DMA)

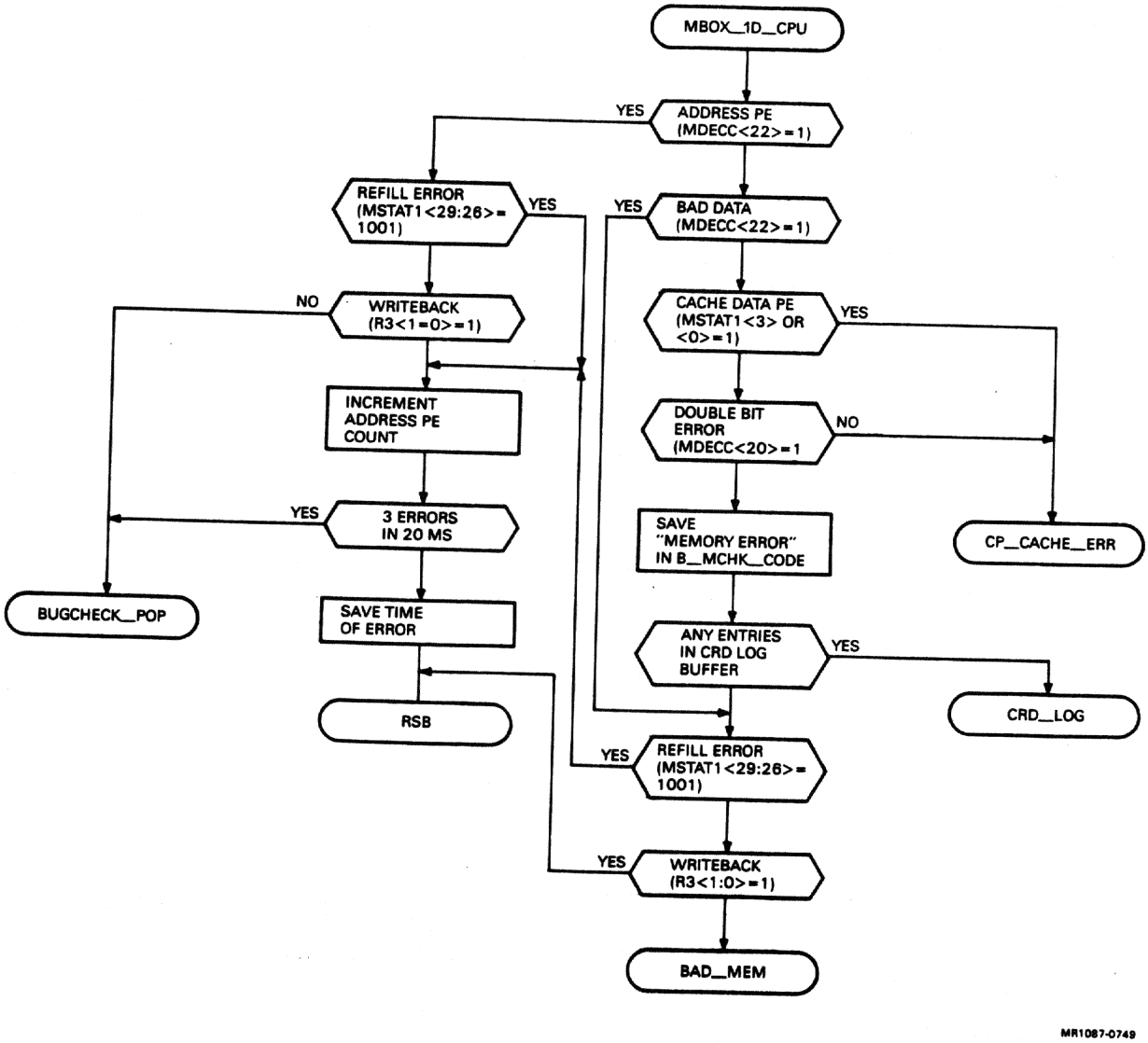
TB errors do not set either MBOX FE or MBOX INTERRUPT. However they are serviced here. They do not latch the cycle type bits in MSTAT1.



VMS Machine Check Handler Flow Chart (Part 32 of 50)

MBOX_1D_CPU

MBOX 1D error triggered by a CPU reference. Bugcheck only if the error rate is too high.
When BAD DATA is written, the error is ignored until someone reads it.
When BAD DATA is consumed, the assumption is that it will be used and the error is treated like the MBOX FE case.
R3 = CYCLE TYPE (MSTAT1 <29:26>)



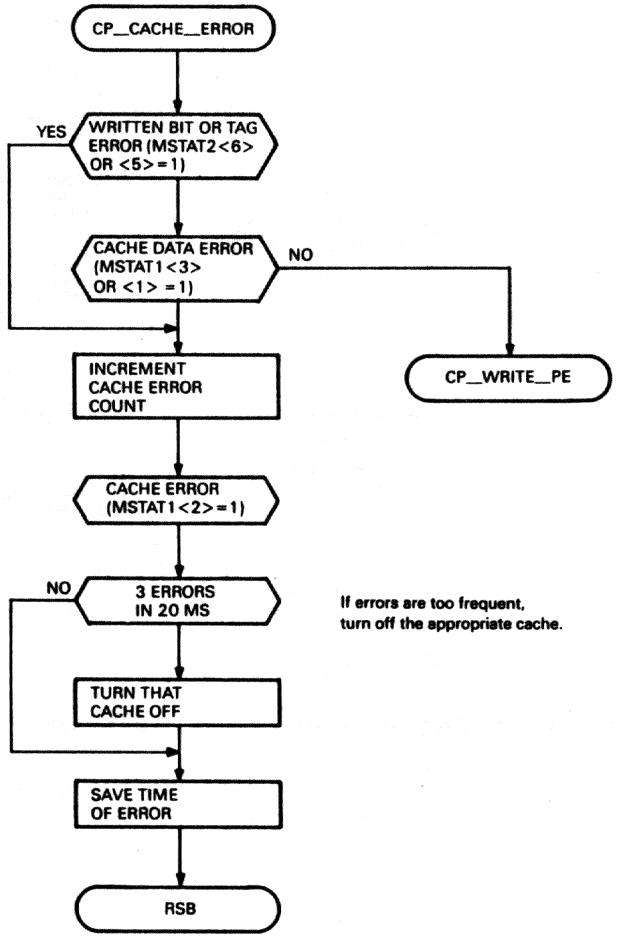
MR1087-0749

VMS Machine Check Handler Flow Chart (Part 33 of 50)

CP_CACHE_ERROR

Cache errors that reach here are,

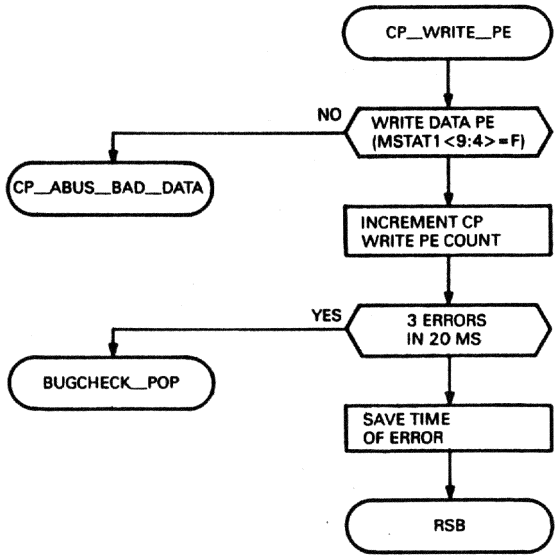
— CACHE TAG PARITY ERROR
— CACHE W PARITY ERROR
— CP BYTE WRT CACHE DATA PE
— CACHE DATA PE



VMS Machine Check Handler Flow Chart (Part 34 of 50)

CP_WRITE_PE

CP WRITE PARITY ERROR reported via MBOX 1D interrupt occurs for those writes destined for memory arrays. The data has been written along with the BAD DATA ERROR flag. Because the PC and this error are not synchronous the failing instruction cannot be retried. When the BAD DATA is read, that USER (or the SYSTEM) can be aborted at that time.

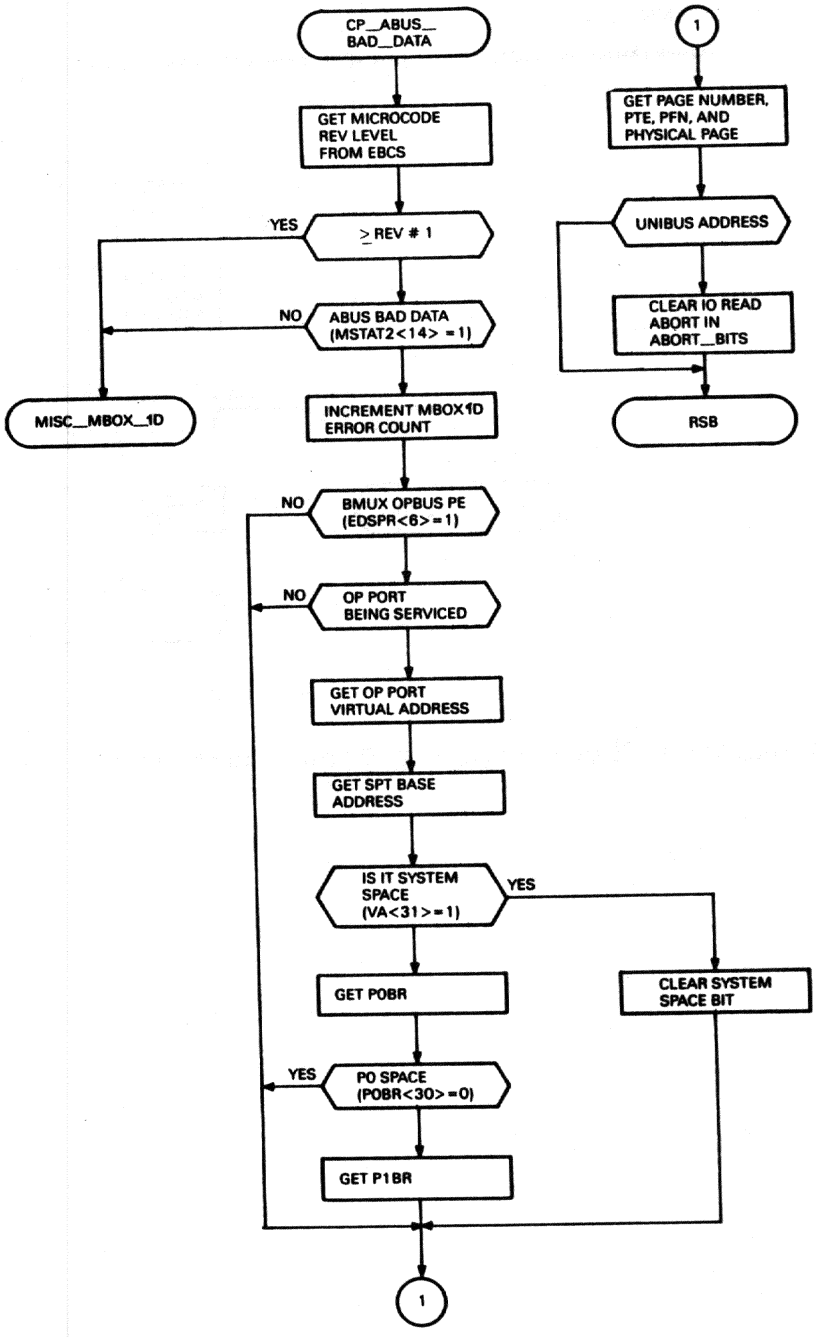


VMS Machine Check Handler Flow Chart (Part 35 of 50)

CP_ABUS_BAD_DATA:

ABUS BAD DATA ERROR flag gets set by the SBIA because of a READ DATA TAG and a MASK field of two from the SBI (RDS).

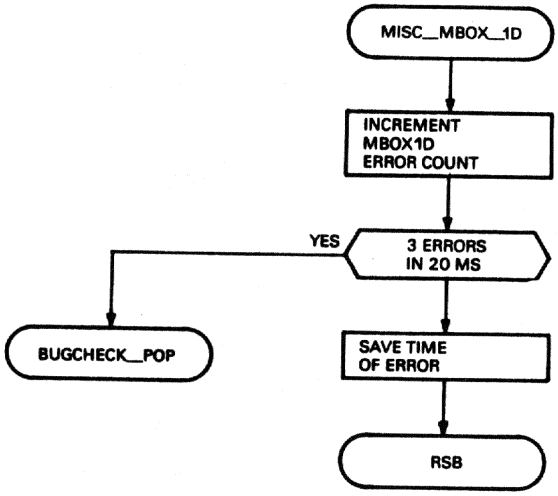
The ABUS BAD DATA ERROR flag gets set by the SBIA because of a READ DATA TAG and a MASK field of two from the SBI (RDS).
The DATA was consumed by the CPU and probably resulted in an Ebox "B OPBUS" error, EDPSR <6>.



VMS Machine Check Handler Flow Chart (Part 36 of 50)

MISC_MBOX_1D

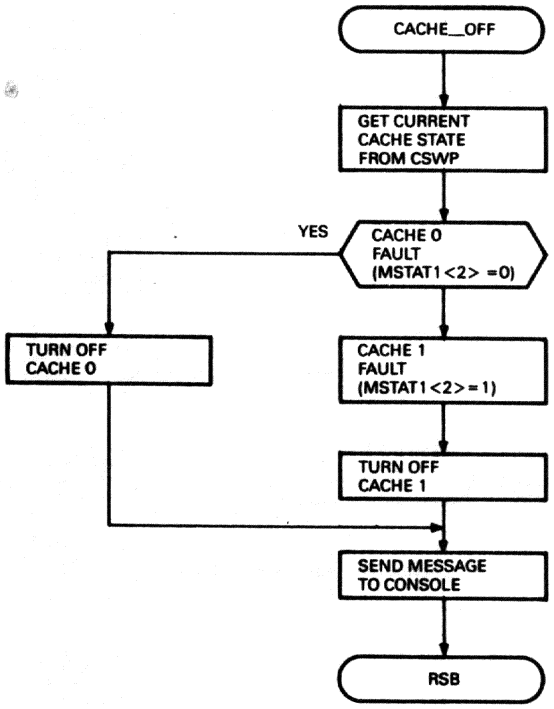
This routine handles any MBOX_1D error not caught by previous MBOX machine checks.



VMS Machine Check Handler Flow Chart (Part 37 of 50)

CACHE_OFF

This routine is called from CP CACHE ERROR and DMA ERROR. The purpose is to turn off failing half of cache if error thresholds are exceeded.



VMS Machine Check Handler Flow Chart (Part 38 of 50)

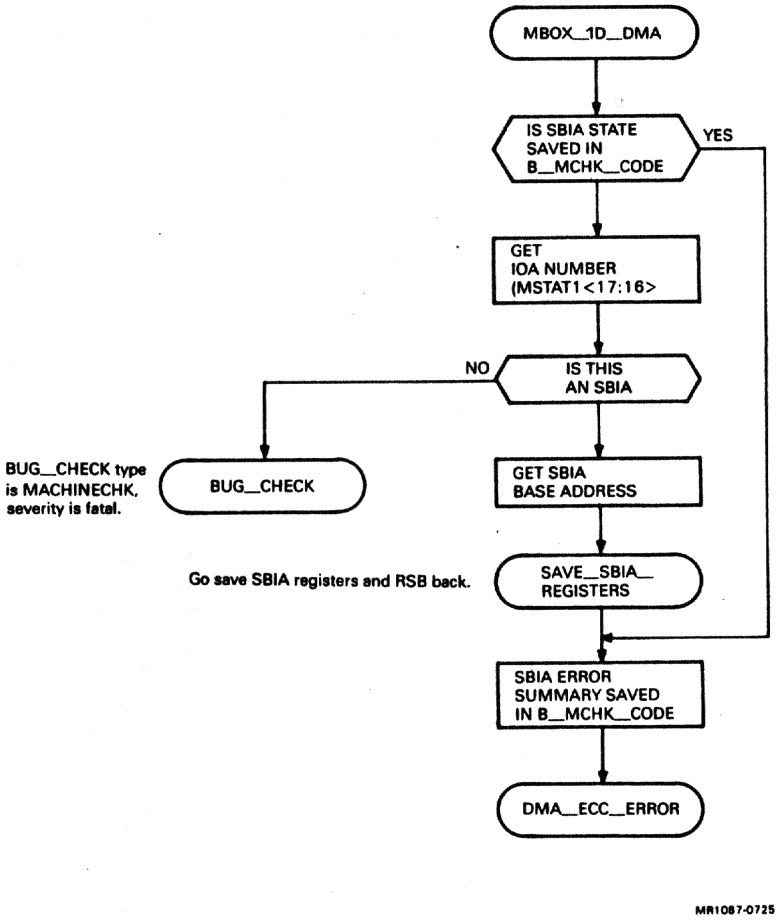
MBOX__1D__DMA

MBOX 1D error triggered by an ABUS reference.

In general, these are non-fatal. Bugcheck only if the error rate is too high. Whenever bad data is written, the error is ignored until someone reads it.

Errors on reading are handled by device drivers.

R3 = Cycle type (MSTAT1 <29:26>)



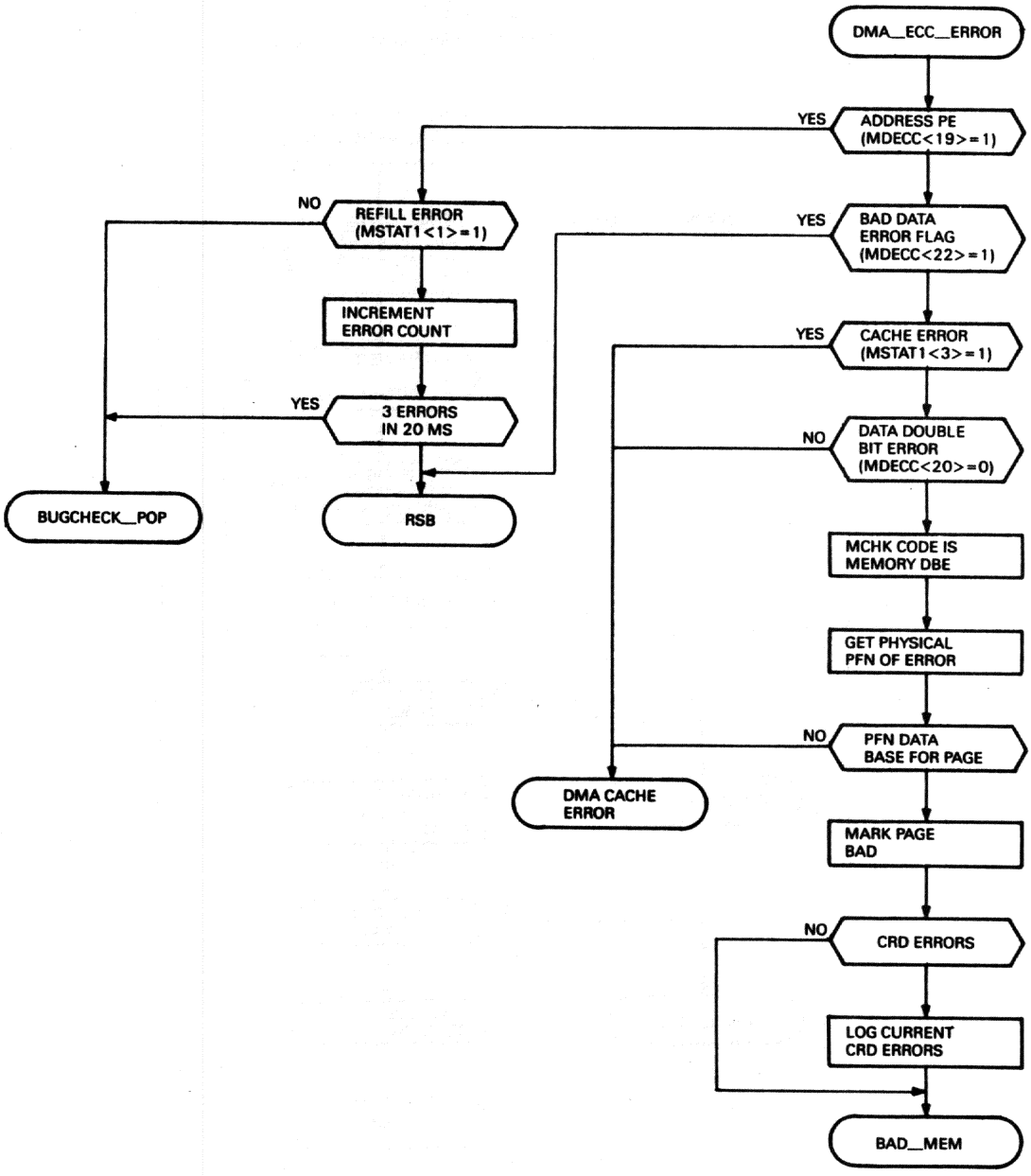
VMS Machine Check Handler Flow Chart (Part 39 of 50)

DMA__ECC__ERROR

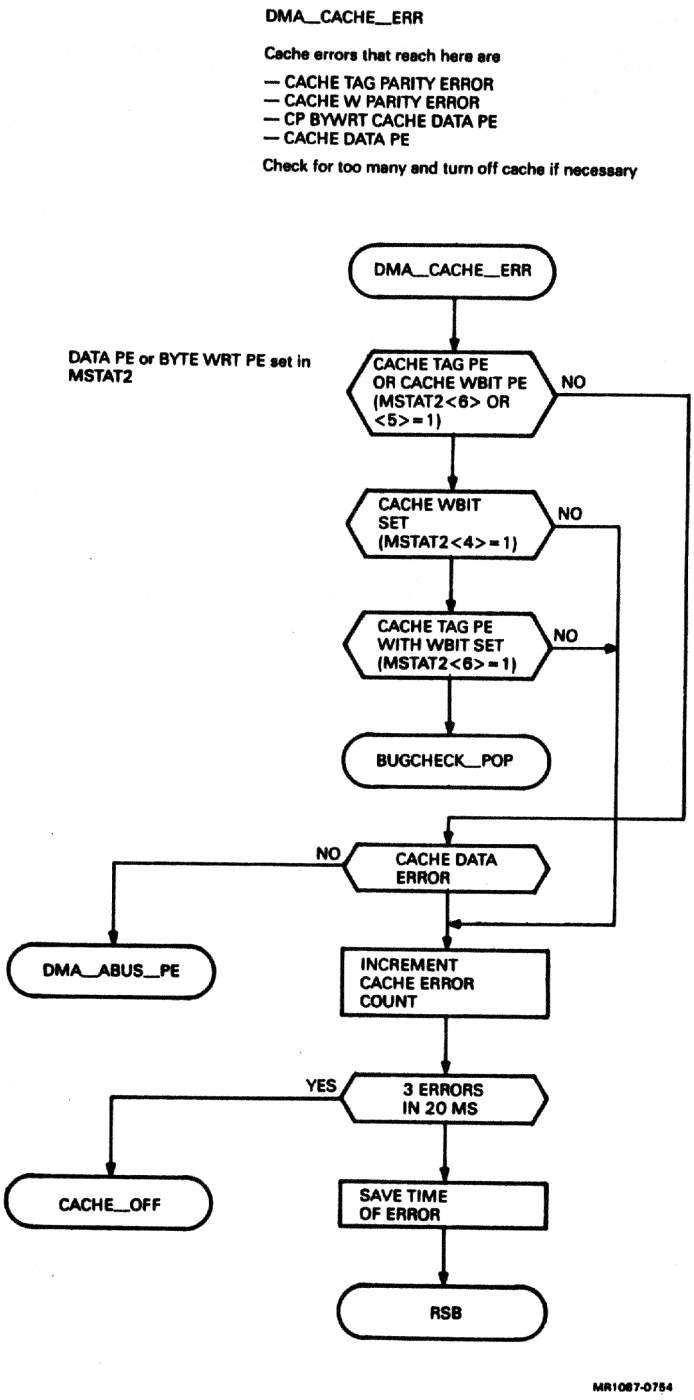
ADDRESS PARITY ERROR — If ABUS REFILL just exit, the driver will do the rest.

DATA DOUBLE BIT or BAD DATA ERROR — just log them, the device driver will do the rest. For DATA DOUBLE BIT put the page on the bad page list.

R3 = CYCLE TYPE (MSTAT1 <29:26>)



VMS Machine Check Handler Flow Chart (Part 40 of 50)

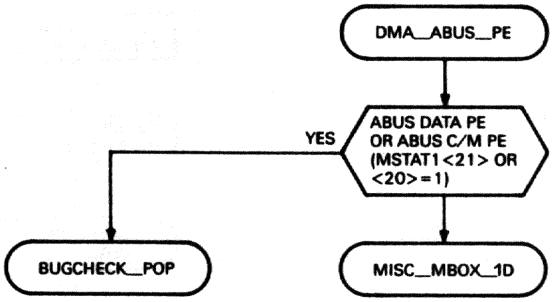


VMS Machine Check Handler Flow Chart (Part 41 of 50)

DMA__ABUS__PE

ABUS parity errors detected by MBOX 1D interrupts come from a DMA write cycle with a CONTROL parity error and the first write overlapping a REFILL.

R3 = CYCLE TYPE (MSTAT1 <29:26>)



VMS Machine Check Handler Flow Chart (Part 42 of 50)

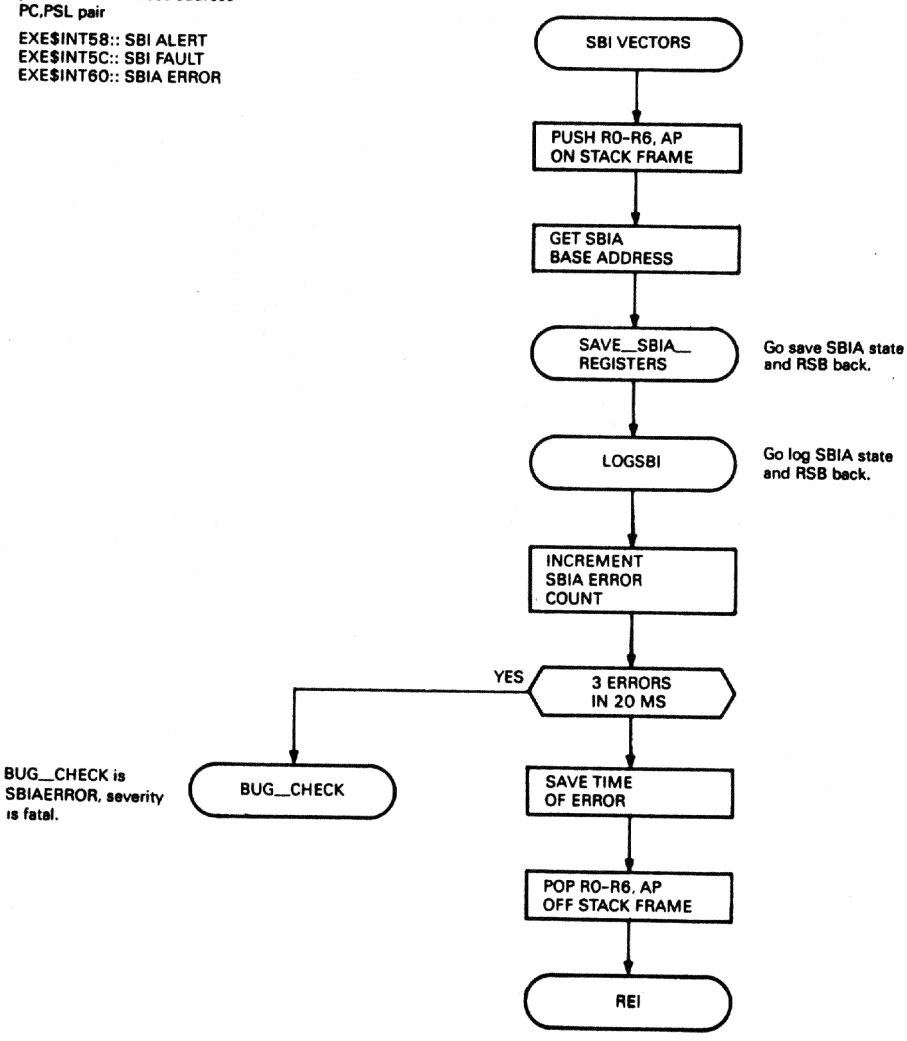
SBI VECTORS

SBI ALERT ERROR and FAULT interrupts are handled here.
All interrupts cause a full SBIA log.
SBI FAIL is treated like power fail and is handled elsewhere.

The SBIA has a micro-sequencer that is used for CPU operations to the SBI. Its control word is parity protected. If a parity error is detected while the CPU is making a reference to the SBI then it will be reported with MBOX fatal error and CP__IO__BUF. When the sequencer is not in use for CPU references it loops on a single control word. If a parity error is detected in this case we come here via an interrupt.

Stack on entry:
pointer to SBIA base address
PC,PSL pair

EXE$INT58:: SBI ALERT
EXE$INT5C:: SBI FAULT
EXE$INT60:: SBIA ERROR



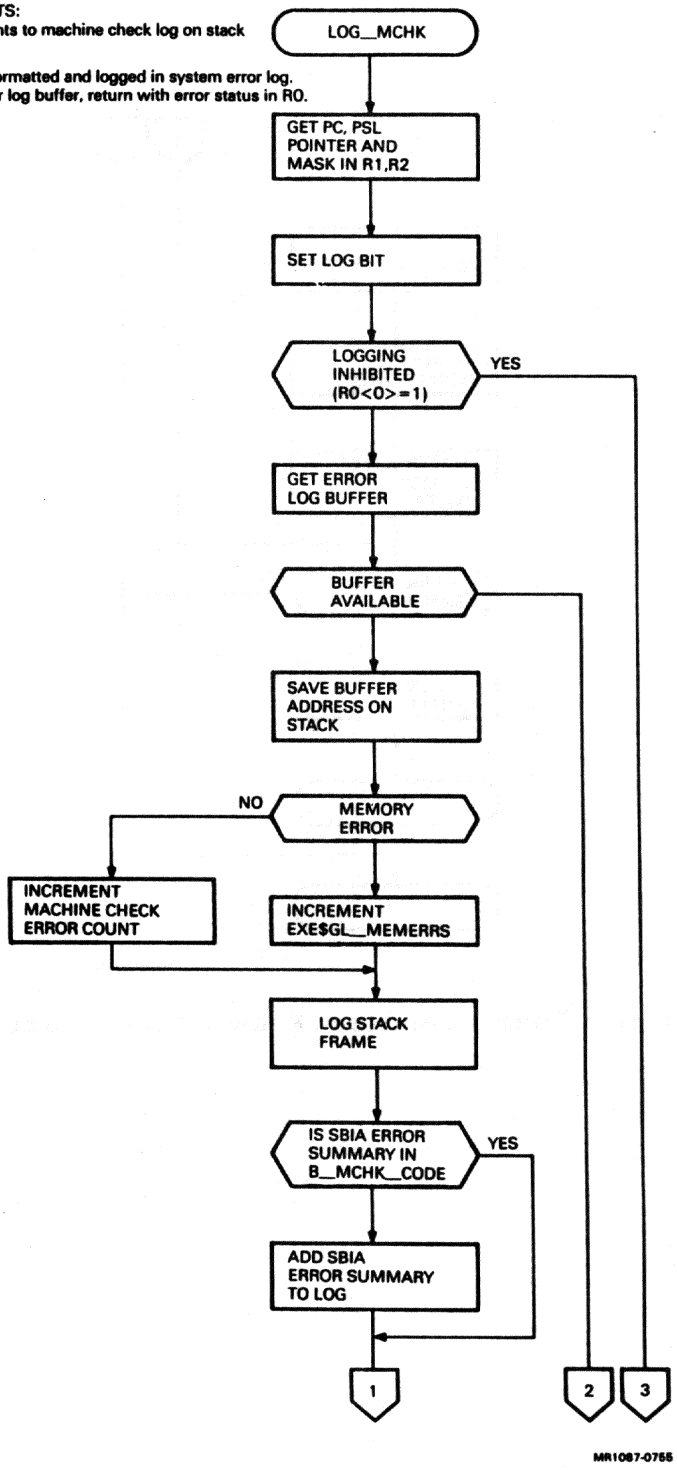
VMS Machine Check Handler Flow Chart (Part 43 of 50)

MCHECK

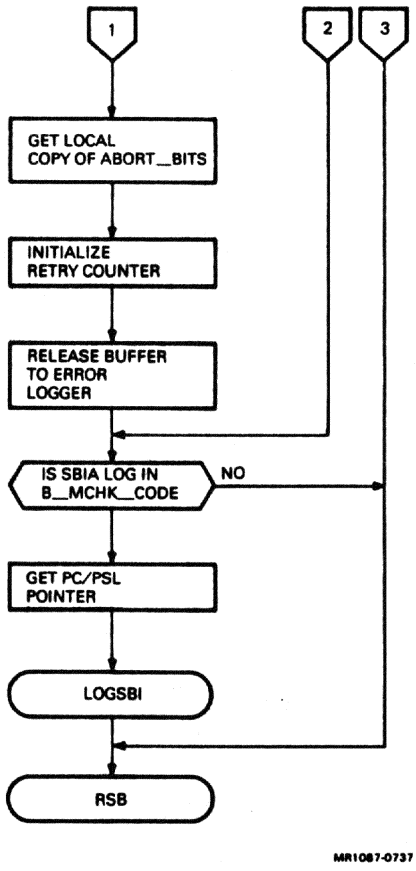
This routine formats information for error logging and checks if a machine check recovery block has inhibited logging.

IMPLICIT INPUTS:
(AP): points to machine check log on stack

OUTPUTS:
Error is formatted and logged in system error log.
If no error log buffer, return with error status in R0.



VMS Machine Check Handler Flow Chart (Part 44 of 50)



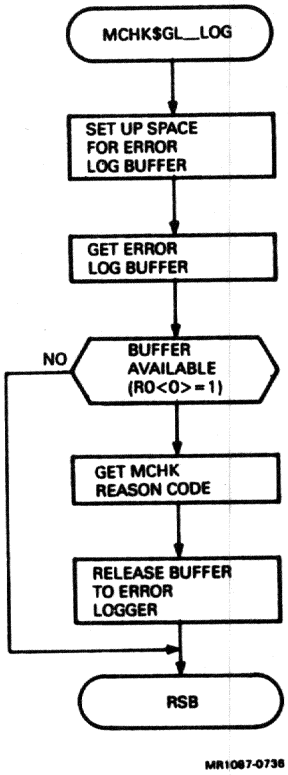
VMS Machine Check Handler Flow Chart (Part 45 of 50)

MCHK$GL_LOG

The error is formatted and logged in the system error log.
If there is no error log buffer, return with error status in R0.

INPUTS:
R3: error type code
R4: length of frame
R5: address of frame

OUTPUTS:
Error is formatted and logged in system error log.
If no error log buffer, return with error status in R0.
R0—R5 destroyed.

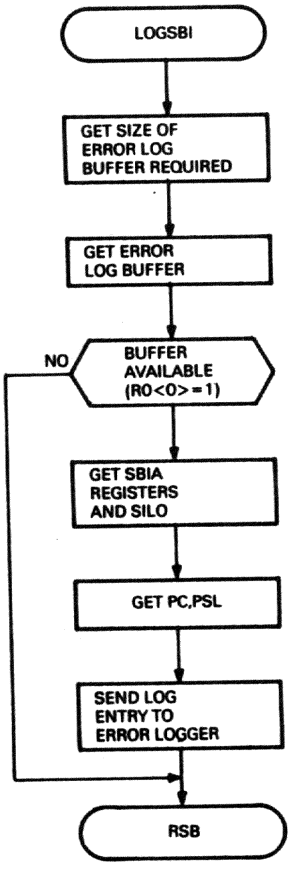


LOGSBI

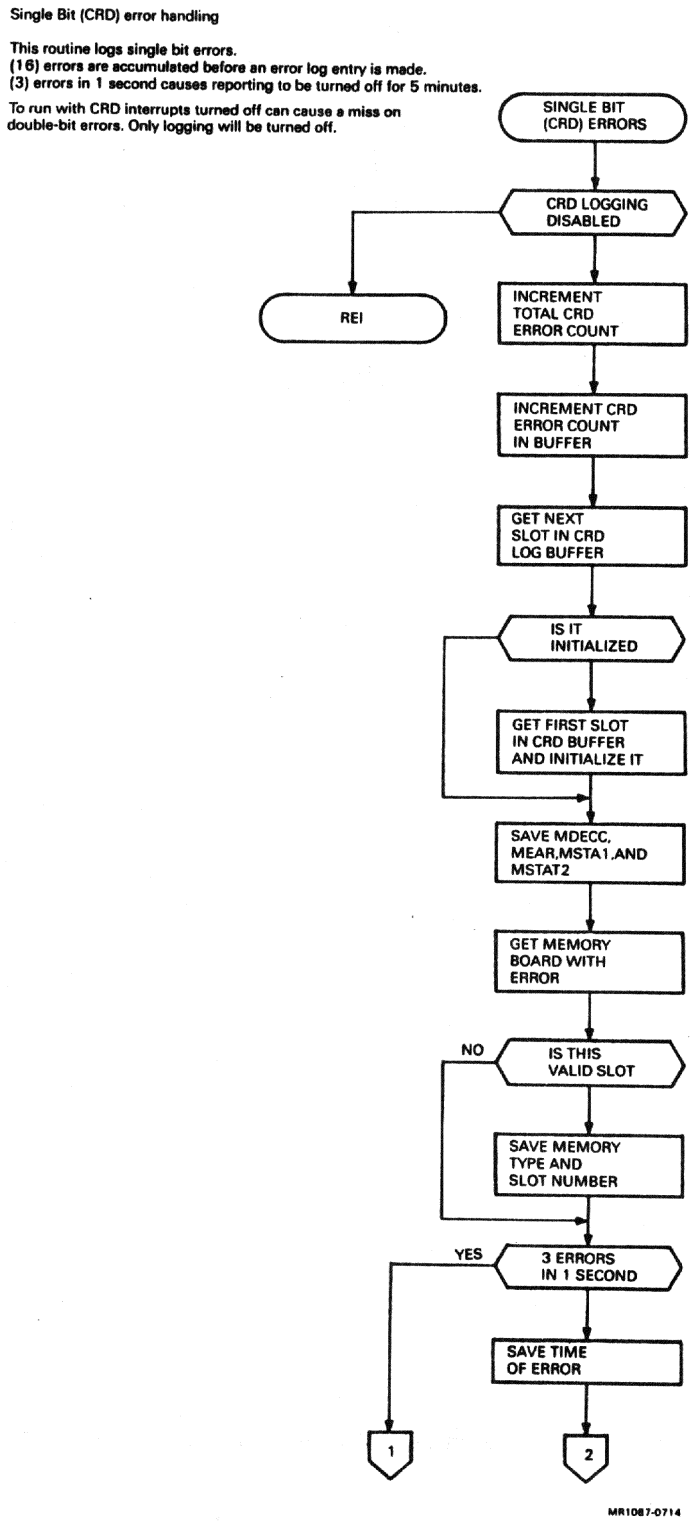
This routine is used to log the following SBIA (IOA) and SBI registers and related information.

COMP
CNFGR
DCONTROL
DMAACA
DMAAID
DMABCA
DMABID
DMACCA
DMACID
DMAICA
DMAIID
ERROR
FAULT
MAINT
SILO1
STATUS
SUMMARY
TMOADDR

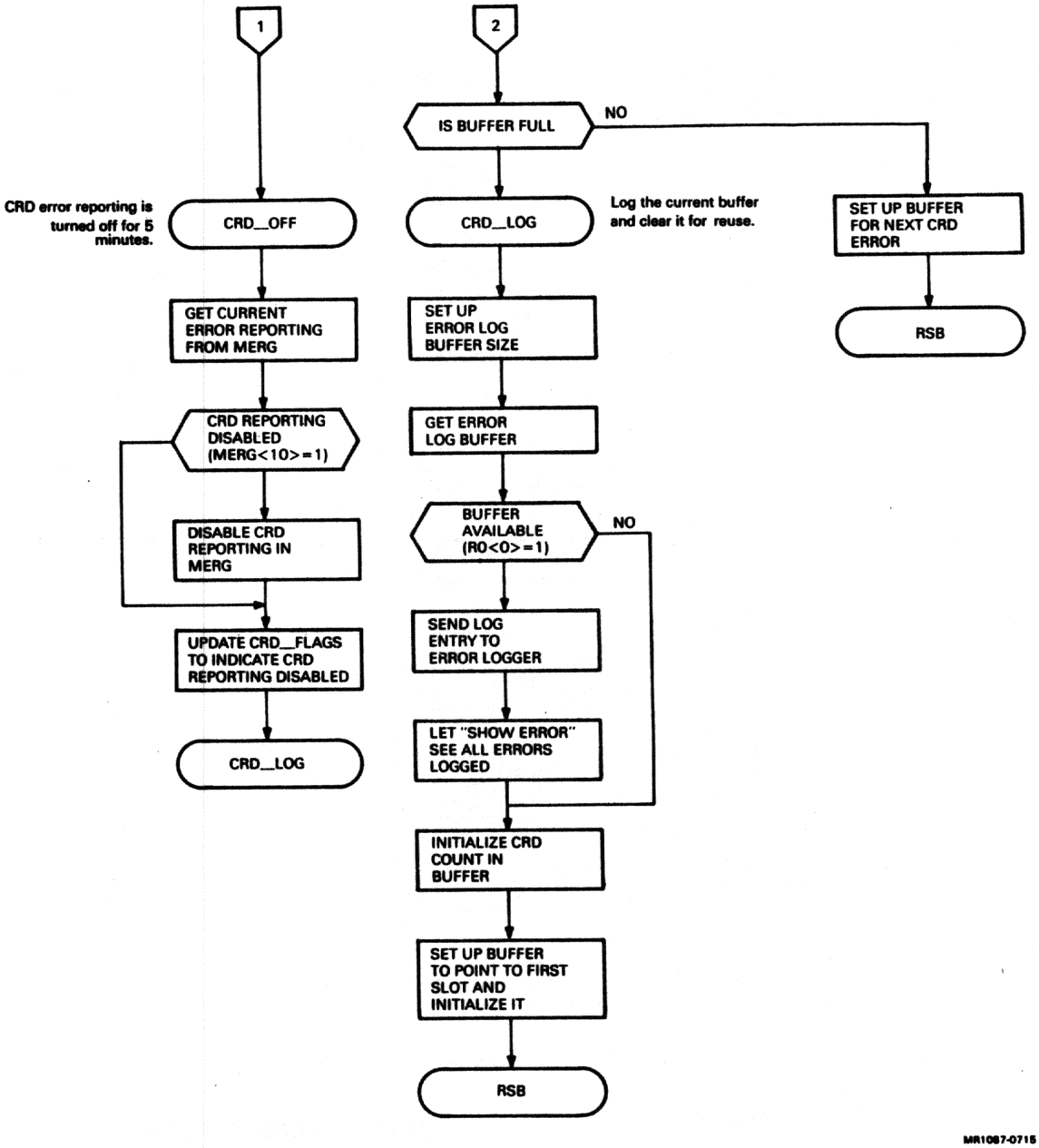
INPUTS:
R1 -> Address of PC, PSL



VMS Machine Check Handler Flow Chart (Part 46 of 50)



VMS Machine Check Handler Flow Chart (Part 47 of 50)

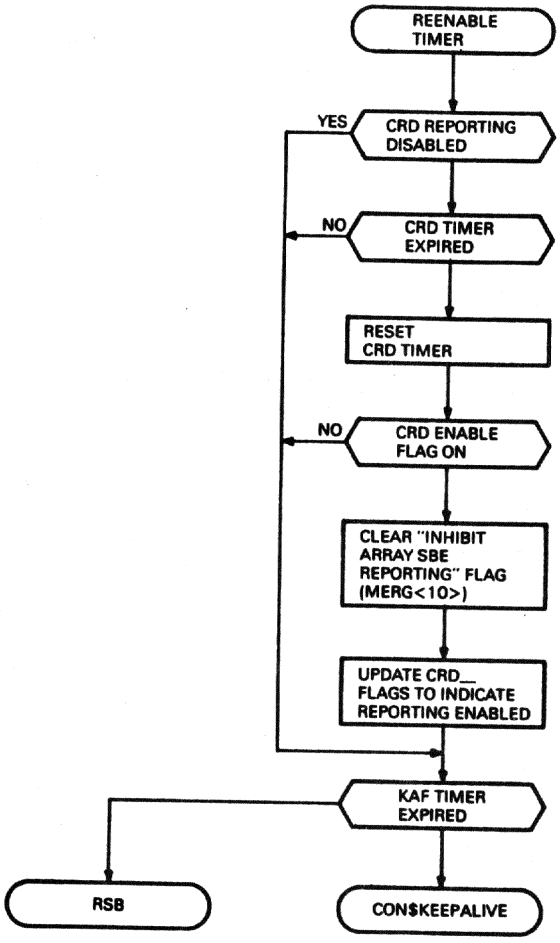


VMS Machine Check Handler Flow Chart (Part 48 of 50)

REENABLE TIMER

This routine is called by the system clock routine. If CRD reporting is disabled and the timer has expired, reporting is reenabled if allowed by the SYSGEN parameter.

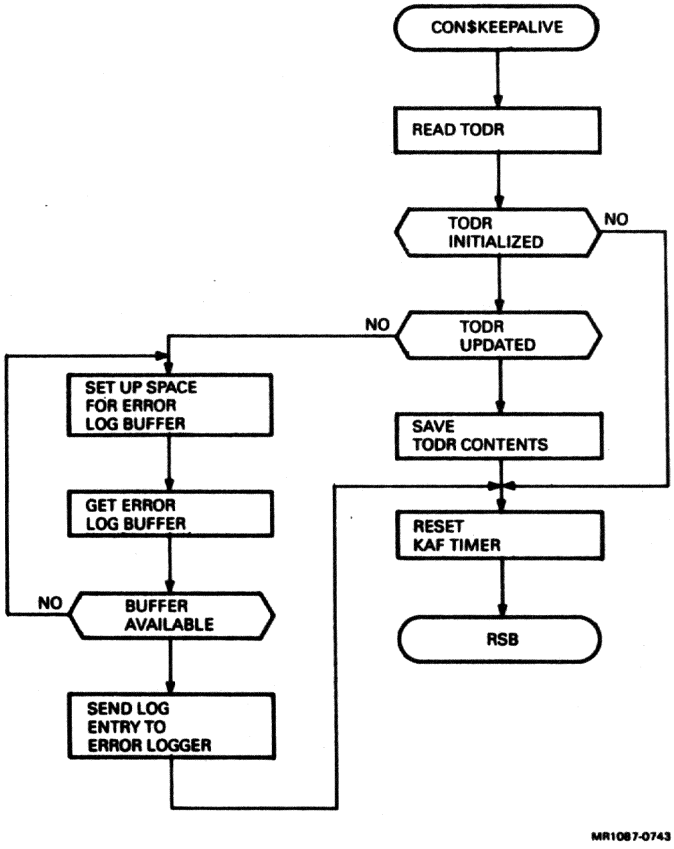
Routine does not turn off interrupts, only logging.



VMS Machine Check Handler Flow Chart (Part 49 of 50)

CON$KEEPALIVE

This routine is called every 90 seconds to read the TODR register. The purpose is to determine if the console is functioning normally by updating the TODR. If the value is unchanged, the console is dead or hung and a REBOOT is attempted.



VMS Machine Check Handler Flow Chart (Part 50 of 50)

APPENDIX D

VAX8600/8650 KEEP ALIVE FAIL (KAF) FLOW AND SNAP FILE DESCRIPTION

INTRODUCTION

The Keep Alive Fail (KAF) mechanism provides a method for the Console to monitor the overall activity of the VAX8600/8650. If the Console determines that the EBox has not entered IRD Time during the last 300 milliseconds it declares a Keep Alive Fail Condition and Snapshots (captures) the state of the system.

The Snapshot process (See Figure D-1 VAX8600/8650 Keep Alive Flow) involves controlling the CPU Clock (ON/OFF) and capturing the state of:

1. The SDB visibility channels (Tables 3 and 4).
2. The console control and status registers (Table 5).
3. The EMM control and status registers (Table 6).
4. The EBox scratch pad RAMs (Table 7).
5. The CPU internal (micro, macro, and miscellaneous) registers (Table 8).
6. The contents of the PAMM (Table 9).
7. The last 64 Longwords that were pushed on the Interrupt Stack, the 25 longword machine check stack frame (if it is on the interrupt stack after the top 64 longwords), and the bottom 64 longwords on the interrupt stack (Table 10).
8. The major ABus adapter (SBIA) and nexus control and status registers (Table 11).
9. The clock status, clock alignment, and a 25 microstep uPC trace (Table 12).

The overall organization and format of a snap file is illustrated in Figure D-2). Note that each record, excluding the master header record, has a standard SNAPSHOT (data) header (Table 2).

After the data has been collected the snap shot is appended to the master header record (See Table 1) and the console checks the VERIFY (or NOVERIFY) software switch to determine if it should reload and verify the contents of the VAX8600/8650 RAMs. If the switch is set the console will reload all the RAMs and verify their contents. Errors detected during the verify stage will be recorded in the master header record. If the switch is not set, the console will skip the reload and verify step and proceed as follows.

The console will write the snapshot information to a snap file on the RL02 (SNAP1.DAT or SNAP2.DAT). It will then attempt to reboot the system.

NOTE

> If both SNAP1.DAT and SNAP2.DAT files are already valid (two snap shots have been taken without VMS transferring them to ERRSNAP.LOG) neither SNAP1.DAT or SNAP2.DAT will be written.

If the system reboot is successful, the SNAP file will be transferred to the VMS side of the system and written to SYS$SYSROOT:[SYSERR]ERRSNAP.LOG;n. The version number (n) is incremented each time a new SNAP file is written.

If the console is unable to reboot the system it will build a second SNAPSHOT file (SNAP2.DAT) and than loop attempting to reboot the system.

SNAP FILE EXAMPLES

Following the tables there are two example printouts. The first example was produced by VSRBLD. VSRBLD runs under VMS and is located in the SYS$SYSROOT:[SYSEXE] directory. VSRBLD translates ERRSNAP.LOG into a more readable and self explanatory format. To save space, only a sample of the SDB Visibility translation is included.

The second example was generated using the Console command "SHOW SNAP". This example is highlighted to identify the data headers and other key information described in the Tables 1 through 12. The purpose of this example is to help you quickly locate information when you are working with a SNAPSHOT file in this format.

Finally, there is a list of key SDB visibility signals (Tables 14 and 15). The text preceding the list explains that the state of these signals can be used to manually evaluate the state of a hung or stalled system.

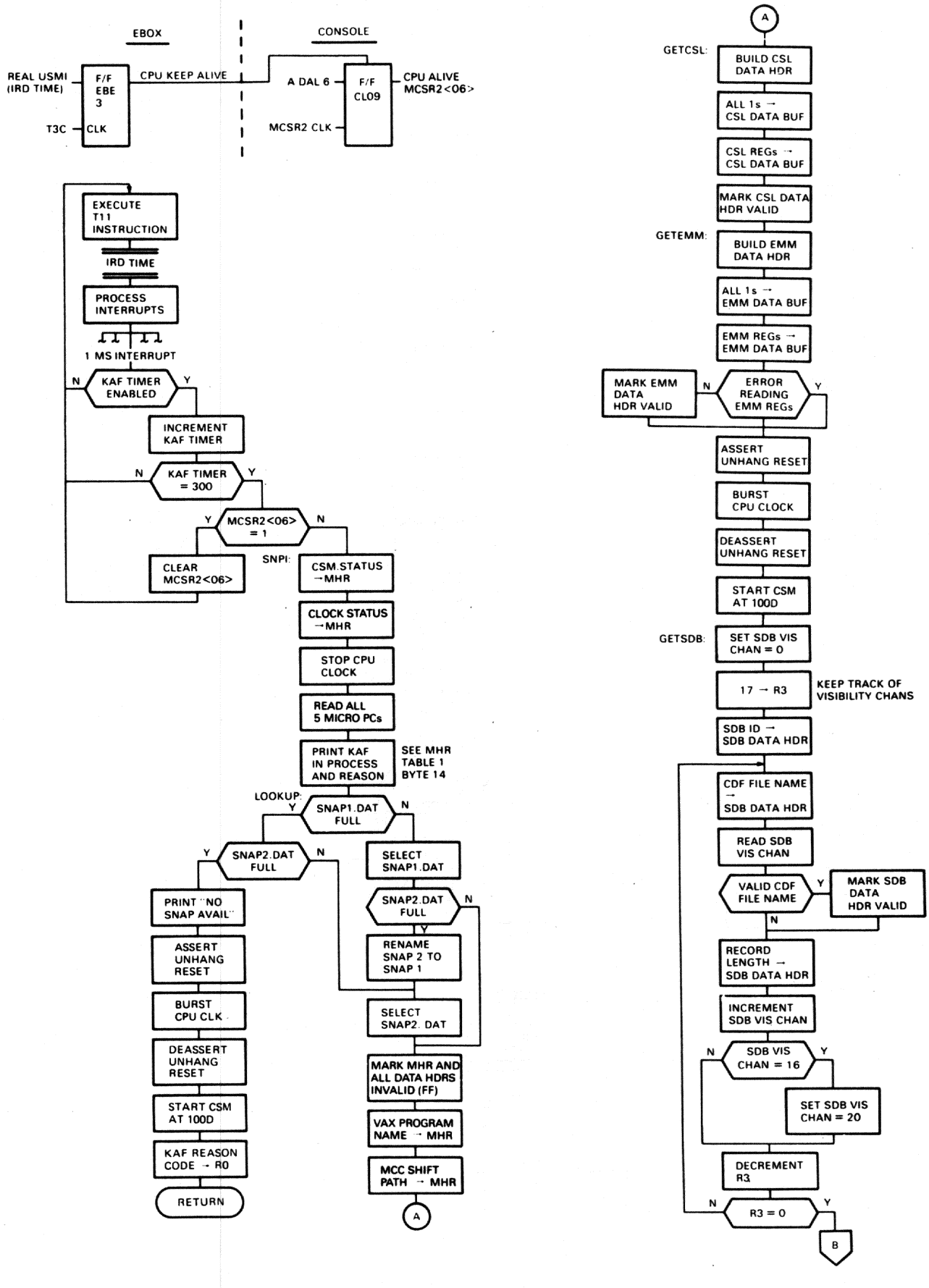


Figure D-1 VAX8600/8650 Keep Alive Flow (Part 1 of 2)

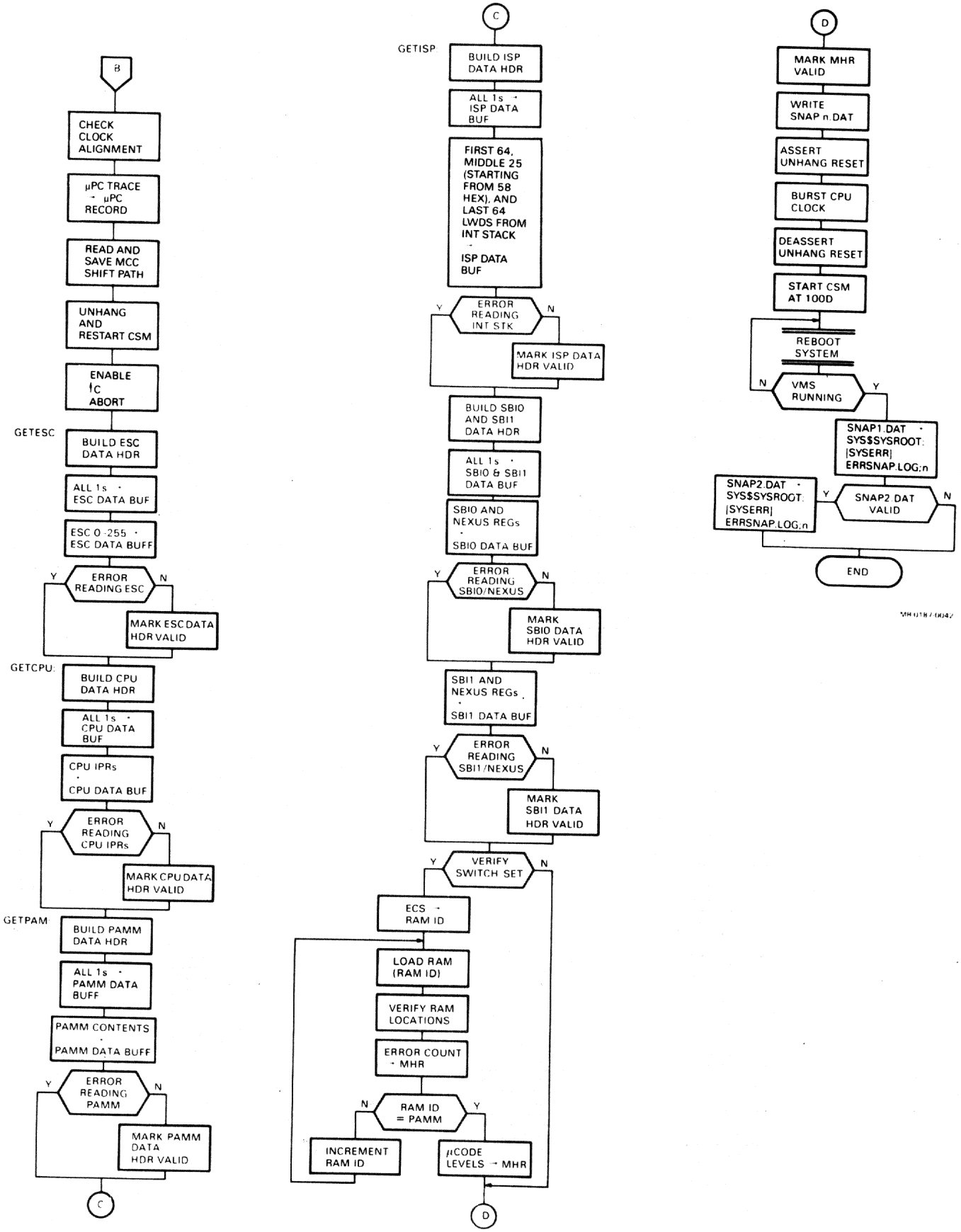


Figure D-1 VAX8600/8650 Keep Alive Flow (Part 2 of 2)

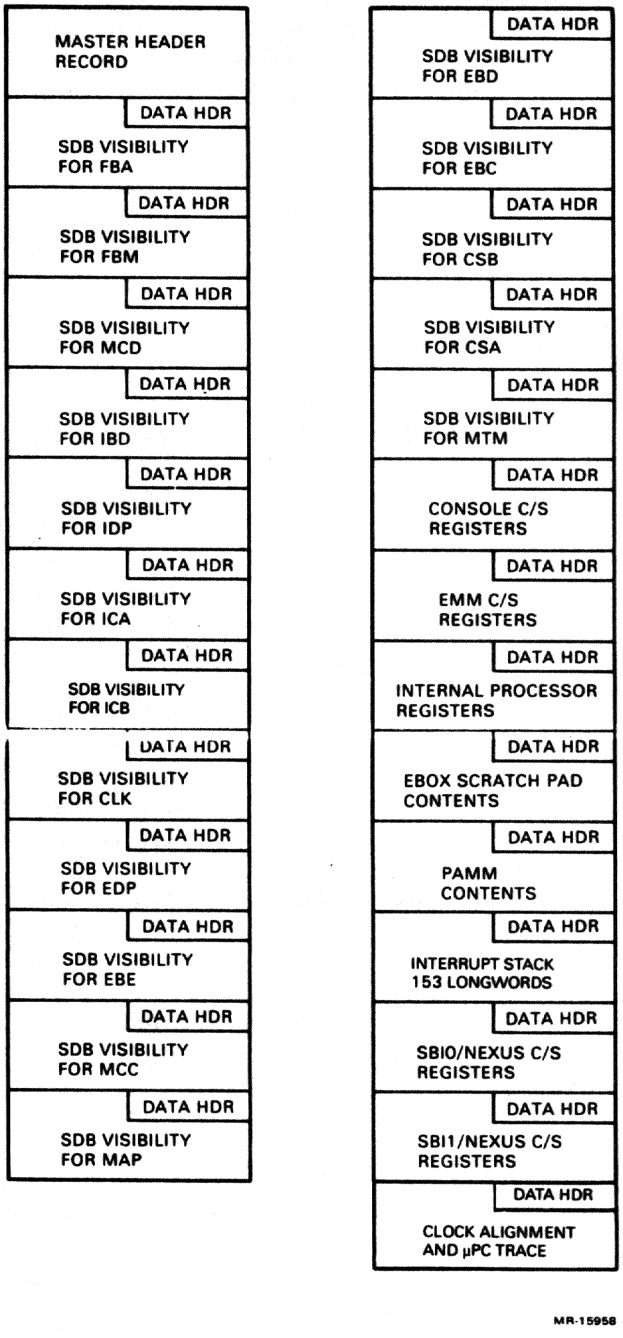


Figure D-2 KAF Snap File Organization

Table 1 SNAPSHOT Master Header Record Format

Field Name	Byte Position	Number of Bytes	Description
Header ID	00	(1)	Contains "20" indicates MASTER HEADER
Status Code	01	(1)	Contains one of the following values: FF = invalid file 01 = valid file
Record Length	02-03	(2)	Total number of bytes in the master header record.
CPU Program	04-09	(6)	Name of the last file loaded by CSL into CPU memory. "VMB" indicates that VMS was running at the time of the KAF. The name is padded with NULLs if it is less than 6 characters long. Note If this field is empty (all null) it indicates that the console was rebooted since the last LOAD and the information is lost.
File Length in Bytes	0A-0B	(2)	Total number of bytes in the entire SNAPx.DAT file.
CSM Status Code	0C	(1)	CPU ERROR HALT codes as defined in DEC STD 032: 0 = CSM could not be forced to run by the console program after the KAF. 4 = Interrupt Stack not valid 5 = Non-Ebox double error 6 = Kernel mode HALT 7 = SCB vector with <1:0> = 3 8 = SCB vector with <1:0> = 2 but WCS was not loaded. 9 = pending error on HALT A = CHMx with IS = 1 B = CHMx vector <1:0> not 0

Table 1 SNAPSHOT Master Header Record Format (Cont)

Field Name	Byte Position	Number of Bytes	Description
KAF Reason Code	0D	(1)	Contains one of the following values: 18 = Parity Error in both ESC A & B. Ebox is looping at UPC 20. 19 = EHM was in the process of handling an error when a second error trap occurred. Ebox is looping at address 21. 1A = Ebox detected a WBus parity error. EHM is looping at uPC 24. This will not happen after Console RL02 Rev. 3.0. 1B = CPU ERROR HALT encountered 1C = Non-correctable PE 1D = Power system failure. Indicates that the EMM sensed a DC LO condition without a preceding AC LO. 1E = Unidentified KAF occurred 1F = MBox/SBIA DMA command error or NXM.
Time Stamp Integer	0E-11	(4)	VAX formatted 32-bit TOY integer
Console PROM Revision	12	(1)	Prom revision number.
EMM PROM Revision	13	(1)	Prom revision number.
Minimum HW Revision	14-15	(2)	Note: All 2-byte revs have the following format: Byte 0 - Major Revision Byte 1 - Minor Revision
EBox Ucode Revision	16-17	(2)	
IBox Ucode Revision	18-19	(2)	
MBox Ucode Revision	1A-1B	(2)	
FBoxA Ucode Revision	1C-1D	(2)	
FBoxM Ucode Revision	1E-1F	(2)	
ACCESS Ucode Revision	20-21	(2)	
CPR Ucode Revision	22-23	(2)	
MCF Ucode Revision	24-25	(2)	
CONTEXT Ucode Revision	26-27	(2)	
Ucode Release Revision	28-29	(2)	

Table 1 SNAPSHOT Master Header Record Format (Cont)

Field Name	Byte Position	Number of Bytes	Description
EBox Ucode Verification	2A	(1)	Note: All verification bytes will contain the total number of verification errors detected 0 = None 255 = Maximum
MCF Ucode Verification	2B	(1)	
CONTEXT Ucode Verification	2C	(1)	
FBoxA Ucode Verification	2D	(1)	
FBoxM Ucode Verification	2E	(1)	
FDRAM Ucode Verification	2F	(1)	
IBox Ucode Verification	30	(1)	
IDRAM Ucode Verification	31	(1)	
MBox Ucode Verification	32	(1)	
CPR Ucode Verification	33	(1)	
ACCESS Ucode Verification	34	(1)	
PAMM Verification	35	(1)	
MCC Shift Channel	36-40	(11)	Saved state of the MBox Shift Path (Micro Address and Data)
Spare	41	(1)	
Console Software Version	42-43	(2)	Console Software Revision (Major, Minor).
CLK.ST	44-45	(2)	CPU Clock State (Word) <15> Rate 1/5 (1) or full (0) <14:08> Frequency in MHz, 1 if external <07:06> MBZ <05> Mark stop <04> Clock running <03:00> Bit<n> = phase n if clock stopped
Calculated CSPE Syndrome	46-49	(4)	CSPE syndrome. FFFFFFFF if no CSPE

Total size = 74 bytes

Table 2 Standard SNAPSHOT Record Header Format

Field Name	Byte Position	Number of Bytes	Description
Header ID	00	(1)	Contains one of the following codes: 21 = FBA 2E = EBC 22 = FBM 2F = CSB 23 = MCD 30 = CSA 24 = IBD 31 = MTM 25 = IDP 32 = CSL 26 = ICA 33 = EMM 27 = ICB 34 = IPRs 28 = CLK 35 = ESC 29 = EDP 36 = PAMM 2A = EBE 37 = ISP 2B = MCC 38 = SBI0 2C = MAP 39 = SBI1 2D = EBD 3C = uPC Trace
Status Code	01	(1)	Contains one of the following values: 01 = Valid record data FF = Invalid record data - a record could be invalid if, for instance, CSM is not able to provide the console program with the data.
Record Length	02-03	(2)	Total number of bytes in this record (Header and Data)
CDF filename	04-09	(6)	Name of the .CDF file used to extract the SDB data contained in this record. CDF files will always be 6 characters. For non SDB records this field will contain an ASCII record name in the form 'NAMrec')

Total size = 10 bytes

Table 3 SNAPSHOT Record Format for FBA and FBM SDB-Visability Channels

Field Name	Byte Position	Number of Bytes	Description
Record Header	00-09	(10)	See Standard SNAPSHOT Record Header (Table 2)
SDB Data Byte 0	0A-0B	(2)	Bits <11:0> of MUX SEL 0, ENA 0
SDB Data Byte 1	0C-0D	(2)	Bits <11:0> of MUX SEL 1, ENA 0
SDB Data Byte 2	0E-0F	(2)	Bits <11:0> of MUX SEL 2, ENA 0
SDB Data Byte 3	10-11	(2)	Bits <11:0> of MUX SEL 3, ENA 0
:			:
:			:
SDB Data Byte 30	46-47	(2)	Bits <11:0> of MUX SEL 6, ENA 3
SDB Data Byte 31	48-49	(2)	Bits <11:0> of MUX SEL 7, ENA 3

Total size = 74 bytes

NOTE

Only bits <11:00> of each 2-byte set in the FBA and FBM records contain valid data. Bits <15:12> are zero and should be ignored.

Table 4 Standard SDB Visability Data Format (excluding FBA and FBM)

Field Name	Byte Position	Number of Bytes	Description
Record Header	00-09	(10)	See Standard SNAPSHOT Record Header (Table 2)
SDB Data Byte 0	0A	(1)	Bits <7:0> of MUX SEL 0, ENA 0
SDB Data Byte 1	0B	(1)	Bits <7:0> of MUX SEL 1, ENA 0
SDB Data Byte 2	0C	(1)	Bits <7:0> of MUX SEL 2, ENA 0
SDB Data Byte 3	0D	(1)	Bits <7:0> of MUX SEL 3, ENA 0
SDB Data Byte 4	0E	(1)	Bits <7:0> of MUX SEL 4, ENA 0
SDB Data Byte 5	0F	(1)	Bits <7:0> of MUX SEL 5, ENA 0
SDB Data Byte 6	10	(1)	Bits <7:0> of MUX SEL 6, ENA 0
SDB Data Byte 7	11	(1)	Bits <7:0> of MUX SEL 7, ENA 0
SDB Data Byte 8	12	(1)	Bits <7:0> of MUX SEL 0, ENA 1
SDB Data Byte 9	13	(1)	Bits <7:0> of MUX SEL 1, ENA 1
:			:
:			:
SDB Data Byte 30	29	(1)	Bits <7:0> of MUX SEL 6, ENA 3
SDB Data Byte 31	2A	(1)	Bits <7:0> of MUX SEL 7, ENA 3

Total size = 42 bytes

Table 5 SNAPSHOT Record Format for Console Registers

32

Field Name	Byte Position	Number of Bytes	Description
Record Header	00-09	(10)	See Standard SNAPSHOT Record Header (Table 2)
MCSR0	0A	(1)	State of Console Register
MCSR1	0B	(1)	State of Console Register
MCSR2	0C	(1)	State of Console Register
MCSR3	0D	(1)	State of Console Register
ERSR	0E	(1)	State of Console Register
LRSR	0F	(1)	State of Console Register
RRSR	10	(1)	State of Console Register
spare	11	(1)	
QCSR0	12	(1)	State of Console Register
QCSR1	13	(1)	State of Console Register
QCSR2	14	(1)	State of Console Register
QCSR3	15	(1)	State of Console Register
SID0	16	(1)	State of Console Register
SID1	17	(1)	State of Console Register
SID2	18	(1)	State of Console Register
SID3	19	(1)	State of Console Register
RL CTRL Status	1A-1B	(2)	State of Console Load Device
RL Drive Status	1C-1D	(2)	State of Console Load Device

Total size = 30 bytes

Table 6 SNAPSHOT Record Format for EMM Registers

Field Name	Byte Position	Number of Bytes	Description
Record Header	00-09	(10)	See Standard SNAPSHOT Record Header (Table 2)
POWREG	0A	(1)	Regulator On/Off State
MARGEN	0B	(1)	Margen Enable Register
MARHILO	0C	(1)	Margen High/Low Register
MODOK	0D-0E	(2)	Module OK Register
MISREG	0F	(1)	Misc. Hardware Status
SWREG	10	(1)	Misc. Software Status
PROM revision	11	(1)	Prom Revision Number
REGULATOR_A VOLTAGE	12-13	(2)	Measurement of +5V
REGULATOR_B VOLTAGE	14-15	(2)	Measurement of +5V
REGULATOR_C VOLTAGE	16-17	(2)	Measurement of +5V
REGULATOR_D VOLTAGE	18-19	(2)	Measurement of -2V
REGULATOR_E VOLTAGE	1A-1B	(2)	Measurement of -2V
REGULATOR_F VOLTAGE	1C-1D	(2)	Measurement of -5.2V
REGULATOR_H VOLTAGE	1E-1F	(2)	Measurement of -5.2V
GND CURRENT VALUE	20-21	(2)	Measurement of Ground current
REGULATOR_L + VOLTAGE	22-23	(2)	Measurement of +12V
REGULATOR_L - VOLTAGE	24-25	(2)	Measurement of -12V
REGULATOR_K + VOLTAGE	26-27	(2)	Measurement of +15V
REGULATOR_K - VOLTAGE	28-29	(2)	Measurement of -15V
T1 TEMPERATURE VOLTAGE	2A-2B	(2)	Measurement of Input Thermistor
T2 TEMPERATURE VOLTAGE	2C-2D	(2)	Measurement of Output Thermistor
T3 TEMPERATURE VOLTAGE	2E-2F	(2)	Measurement of Output Thermistor
T4 TEMPERATURE VOLTAGE	30-31	(2)	Measurement of Output Thermistor

Total size = 50 bytes

Table 7 EBox Scratch Pad Contents (Location 000 thru 255)

Field Name	Byte Position	Number of Bytes	Description
Record Header	00-09	(10)	See Standard SNAPSHOT Record Header (Table 2)
Escratch location 0	0A-0D	(4)	32-bit Ebox scratchpad ram data
Escratch location 1	0E-11	(4)	32-bit Ebox scratchpad ram data
Escratch location 2	12-15	(4)	32-bit Ebox scratchpad ram data
:			:
:			:
Escratch location 254	402-405	(4)	32-bit Ebox scratchpad ram data
Escratch location 255	406-409	(4)	32-bit Ebox scratchpad ram data

Total size = 1034 bytes

Table 8 SNAPSHOT Record Format for IPR Register

Field Name	Byte Postion	Number of Bytes	Description
Record Header	00-09	(10)	See Standard SNAPSHOT Record Header (Table 2)
KSP	0A-0D	(4)	Internal Processor Registers
ESP	0E-11	(4)	
SSP	12-15	(4)	
USP	16-19	(4)	
ISP	1A-1D	(4)	
P0BR	1E-21	(4)	
P0LR	22-25	(4)	
P1BR	26-29	(4)	
P1LR	2A-2D	(4)	
SBR	2E-31	(4)	
SLR	32-35	(4)	
PCBB	36-39	(4)	
SCBB	3A-3D	(4)	
IPL	3E-41	(4)	
ASTLVL	42-45	(4)	
SISR	46-49	(4)	
ICCS	4A-4D	(4)	
ICR	4E-51	(4)	
TODR	52-55	(4)	
RXCS	56-59	(1)	
RXDB	5A-5D	(4)	
TXCS	5E-61	(4)	
ACCS	62-65	(4)	
MAPEN	66-69	(4)	
PME	6A-6D	(4)	
SID	6E-71	(4)	
PAMACC	72-75	(4)	
PAMLOC	76-79	(4)	
CSWP	7A-7D	(4)	
MDECC	7E-81	(4)	
MENA	82-85	(4)	
MDCTL	86-89	(4)	
MCCTL	8A-8D	(4)	
MERG	8E-91	(4)	
EHSR	92-95	(4)	
STXCS	96-99	(4)	
STXDB	9A-9D	(4)	Internal Registers
VPCBITS	9E-A1	(4)	
EDPSR	A2-A5	(4)	
EBCS	A6-A9	(4)	
CSLINT	AA-AD	(4)	
EDMC	AE-B1	(4)	
IBESR	B2-B5	(4)	
EMD	B6-B9	(4)	
IVASAV	BA-BD	(4)	

Table 8 SNAPSHOT Record Format for IPR Register (Cont)

Field Name	Byte Postion	Number of Bytes	Description
VIBASAV 46	BE-C1	(4)	Internal Registers
ESASAV 47	C2-C5	(4)	
ISASAV 48	C6-C9	(4)	
CPC 49	CA-CD	(4)	
MSTAT1 50	CE-D1	(4)	
MSTAT2 51	D2-D5	(4)	
MEDR 52	D6-D9	(4)	
MEAR 53	DA-DD	(4)	
CSHCTL 54	DE-E1	(4)	
CSES 55	E2-E5	(4)	Miscellaneous Registers
IBGPR 56	E6-E9	(4)	
PSL 57	EA-ED	(4)	
SPADR 58	EE-F1	(4)	
STATE 59	F2-F5	(4)	
EVMQSAV 60	F6-F9	(4)	

Total size = 250 bytes

Table 9 SNAPSHOT Record Format PAMM 36

Field Name	Byte Position	Number of Bytes	Description
Record Header	00-09	(10)	See Standard SNAPSHOT Record Header (Table 2)
PAMM location 0	0A	(1)	Contents of PAMM Array
PAMM location 1	0B	(1)	:
PAMM location 2	0C	(1)	:
:			:
:			:
PAMM location 1022	408	(1)	:
PAMM location 1023	409	(1)	Contents of PAMM Array

Total size = 1034 bytes

NOTE

Only the low-order byte of the PAMM data is included, as all other bits are irrelevant.

Table 10 SNAPSHOT Record Format for the Interrupt Stack

Field Name	Byte Position	Number of Bytes	Description
Record Header	00-09	(10)	See standard SNAPSHOT record header (Table 2)
000 (ISP)	0A-0D	(4)	1st longword on top of stack
004 (ISP)	0E-11	(4)	2nd longword on top of stack
:			:
248 (ISP)	102-105	(4)	63rd longword on top of stack
252 (ISP)	106-109	(4)	64th longword on top of stack
BYTCNT	10A-10D	(4)	BYTCNT (58) of MCHK stack frame, if any
EHMSTS	10E-111	(4)	2nd longword of MCHK stack frame
:			:
PC	166-169	(4)	24th longword of MCHK stack frame
PSL	16A-16D	(4)	25th longword of MCHK stack frame
348 (ISP)	16E-171	(4)	64th longword on bottom of stack
352 (ISP)	172-175	(4)	63th longword on bottom of stack
:			:
604 (ISP)	266-269	(4)	2nd longword on bottom of stack
608 (ISP)	26A-26D	(4)	1st longword on bottom of stack

Total size = 622 bytes

NOTE

The snapshot for release 3.0 of the Console RL02 pack now contains 622 bytes. Following the last 64 longwords placed on the interrupt stack (top of stack) are 25 locations for the stack frame, then the first 64 longwords pushed on the interrupt stack.

First, 64 longwords are popped off the top of the interrupt stack and placed in the ISP record. Then, a search begins for 00000058, the stack frame byte count. If it is found, the 25 longword stack frame is placed in the ISP record. If 00000058 is not found, these 25 longwords are left at FFFFFFFF. Finally, the 64 longwords on the bottom of the interrupt stack is placed in the ISP record.

Keep in mind that the search for 00000058 does not take place until the top of the stack is already in the ISP record. Therefore, the middle 25 longwords cannot be duplicated in the first 64 longwords in the ISP record, but they may be duplicated in the last 64 longwords.

If there are less than or equal 64 longwords of valid data on the interrupt stack, the 25 longwords for the machine check and the first 64 longwords pushed on the stack will be FFFFFFFF. Only the top of the stack is valid.

38

Table 11 SNAPSHOT Record Format for ABus Adapters

Field Name	Byte Position	Number of Bytes	Description	
Record Header	00-09	(10)	See Stadard SNAPSHOT Header Record (Table 2)	
Configuration	0A-0D	(4)	Configuration	2x08 0000
Control/Status	0E-11	(4)	Control/Status	2x08 0004
Error Summary	12-15	(4)	Error Summary	2x08 0008
Diag Control	16-19	(4)	Diagnostic Control	2x08 000C
DMAI Cmd/Addr/ID	1A-21	(8)	DMAI Cmd/Addr and ID	2x08 0010
DMAA Cmd/Addr/ID	22-29	(8)	DMAA Cmd/Addr and ID	2x08 0018
DMAB Cmd/Addr/ID	2A-31	(8)	DMAB Cmd/Addr and ID	2x08 0020
DMAC Cmd/Addr/ID	32-39	(8)	DMAC Cmd/Addr and ID	2x08 0028
SBI Silo	3A-7A	(64)	SBI Silo (4x16.)	2x08 0030
SBI Error	7B-7E	(4)	SBI Error.	2x08 0034
SBI Time-out Address	7F-82	(4)	SBI Time-out Address	2x08 0038
SBI Fault Status	83-86	(4)	SBI Fault Status	2x08 003C
SBI Silo Comparator	87-8A	(4)	SBI Silo Comparator	2x08 0040
SBI Maintanence	8B-8E	(4)	SBI Maintenance.	2x08 0044
TR1 NEXUS	8F-92	(4)	TR1 NEXUS CSR	2x0n n000
Error Summary	93-96	(4)	Error Summary	2x0n n008
Error	97-9A	(4)	Error	2x0n n034
Fault Status	9B-9E	(4)	Fault Status	2x0n n03C
TR2 NEXUS	9F-A2	(4)	TR2 NEXUS CSR	2x0n n000
Error Summary	A3-A6	(4)	Error Summary	2x0n n008
Error	A7-AA	(4)	Error	2x0n n034
Fault Status	AB-AE	(4)	Fault Status	2x0n n03C
TR3 NEXUS	AF-B2	(4)	TR3 NEXUS CSR	2x0n n000
Error Summary	B3-B6	(4)	Error Summary	2x0n n008
Error	B7-BA	(4)	Error	2x0n n034
Fault Status	BB-BE	(4)	Fault Status	2x0n n03C
TR4 NEXUS	BF-C2	(4)	TR4 NEXUS CSR	2x0n n000
Error Summary	C3-C6	(4)	Error Summary	2x0n n008
Error	C7-CA	(4)	Error	2x0n n034
Fault Status	CB-CE	(4)	Fault Status	2x0n n03C
TR5 NEXUS	CF-D2	(4)	TR5 NEXUS CSR	2x0n n000
Error Summary	D3-D6	(4)	Error Summary	2x0n n008
Error	D7-DA	(4)	Error	2x0n n034
Fault Status	DB-DE	(4)	Fault Status	2x0n n03C
TR6 NEXUS	DF-E2	(4)	TR6 NEXUS CSR	2x0n n000
Error Summary	E3-E6	(4)	Error Summary	2x0n n008
Error	E7-EA	(4)	Error	2x0n n034
Fault Status	EB-EE	(4)	Fault Status	2x0n n03C

Table 11 SNAPSHOT Record Format for ABus Adapters (Cont)

Field Name	Byte Position	Number of Bytes	Description	
TR7 NEXUS	EF-F2	(4)	TR7 NEXUS CSR	2x0n n000
Error Summary	F3-F6	(4)	Error Summary	2x0n n008
Error	F7-FA	(4)	Error	2x0n n034
Fault Status	FB-FE	(4)	Fault Status	2x0n n03C
TR8 NEXUS	FF-102	(4)	TR8 NEXUS CSR	2x0n n000
Error Summary	103-106	(4)	Error Summary	2x0n n008
Error	107-10A	(4)	Error	2x0n n034
Fault Status	10B-10E	(4)	Fault Status	2x0n n03C
TR9 NEXUS	10F-112	(4)	TR9 NEXUS CSR	2x0n n000
Error Summary	113-116	(4)	Error Summary	2x0n n008
Error	117-11A	(4)	Error	2x0n n034
Fault Status	11B-11E	(4)	Fault Status	2x0n n03C
TR10 NEXUS	11F-122	(4)	TR10 NEXUS CSR	2x0n n000
Error Summary	123-126	(4)	Error Summary	2x0n n008
Error	127-12A	(4)	Error	2x0n n034
Fault Status	12B-12E	(4)	Fault Status	2x0n n03C
TR11 NEXUS	12F-132	(4)	TR11 NEXUS CSR	2x0n n000
Error Summary	133-136	(4)	Error Summary	2x0n n008
Error	137-13A	(4)	Error	2x0n n034
Fault Status	13B-13E	(4)	Fault Status	2x0n n03C
TR12 NEXUS	13F-142	(4)	TR12 NEXUS CSR	2x0n n000
Error Summary	143-146	(4)	Error Summary	2x0n n008
Error	147-14A	(4)	Error	2x0n n034
Fault Status	14B-14E	(4)	Fault Status	2x0n n03C
TR13 NEXUS	14F-152	(4)	TR13 NEXUS CSR	2x0n n000
Error Summary	153-156	(4)	Error Summary	2x0n n008
Error	157-15A	(4)	Error	2x0n n034
Fault Status	15B-15E	(4)	Fault Status	2x0n n03C
TR14 NEXUS	15F-162	(4)	TR14 NEXUS CSR	2x0n n000
Error Summary	163-166	(4)	Error Summary	2x0n n008
Error	167-16A	(4)	Error	2x0n n034
Fault Status	16B-16E	(4)	Fault Status	2x0n n03C
TR15 NEXUS	16F-172	(4)	TR15 NEXUS CSR	2x0n n000
Error Summary	173-176	(4)	Error Summary	2x0n n008
Error	177-17A	(4)	Error	2x0n n034
Fault Status	17B-17E	(4)	Fault Status	2x0n n03C

Total size = 382 bytes

NOTE

x = the ABus Adapter Number (SBIA0 or SBIA1)
nn = the Transfer Request Level for the corresponding NEXUS

Table 12 SNAPSHOT Record Format for Clock Alignment and uPC Trace

3C

Field Name	Byte Position	Number of Bytes	Description
Record Header	00-09	(10)	See standard SNAPSHOT record header (Table 2)
CLK.ST	0A-0B	(2)	CPU Clock State (Word) See Table 01, CLK.ST
CLK Alignment, Phase 3	0C-0D	(2)	Clock alignment word, phase 3
CLK Alignment, Phase 0	0E-0F	(2)	Clock alignment word, phase 0
CLK Alignment, Phase 1	10-11	(2)	Clock alignment word, phase 1
CLK Alignment, Phase 2	12-13	(2)	Clock alignment word, phase 2
uPC Trace, Step 1	14-1D	(10)	uPC Trace, Step 1
uPC Trace, Step 2	1E-27	(10)	uPC Trace, Step 2
uPC Trace, Step 3	28-31	(10)	uPC Trace, Step 3
uPC Trace, Step 4	32-3B	(10)	uPC Trace, Step 4
uPC Trace, Step 5	3C-45	(10)	uPC Trace, Step 5
uPC Trace, Step 6	46-4F	(10)	uPC Trace, Step 6
uPC Trace, Step 7	50-59	(10)	uPC Trace, Step 7
uPC Trace, Step 8	5A-63	(10)	uPC Trace, Step 8
uPC Trace, Step 9	64-6D	(10)	uPC Trace, Step 9
uPC Trace, Step 10	6E-77	(10)	uPC Trace, Step 10
uPC Trace, Step 11	78-81	(10)	uPC Trace, Step 11
uPC Trace, Step 12	82-8B	(10)	uPC Trace, Step 12
uPC Trace, Step 13	8C-95	(10)	uPC Trace, Step 13
uPC Trace, Step 14	96-9F	(10)	uPC Trace, Step 14
uPC Trace, Step 15	A0-A9	(10)	uPC Trace, Step 15
uPC Trace, Step 16	AA-B3	(10)	uPC Trace, Step 16
uPC Trace, Step 17	B4-BD	(10)	uPC Trace, Step 17
uPC Trace, Step 18	BE-C7	(10)	uPC Trace, Step 18
uPC Trace, Step 19	C8-D1	(10)	uPC Trace, Step 19
uPC Trace, Step 20	D2-DB	(10)	uPC Trace, Step 20
uPC Trace, Step 21	DC-E5	(10)	uPC Trace, Step 21
uPC Trace, Step 22	E6-EF	(10)	uPC Trace, Step 22
uPC Trace, Step 23	F0-F9	(10)	uPC Trace, Step 23
uPC Trace, Step 24	FA-103	(10)	uPC Trace, Step 24
uPC Trace, Step 25	104-10D	(10)	uPC Trace, Step 25

Total size = 270 bytes

NOTE

1. The CPU clock status word is the same as CLK.ST in the master header.

2. The clock signals in Table 13 are checked during each of the four clock phases. If the signal is incorrect, a bit is set in the clock alignment word for that phase. The bit that will be set can be determined from Table 13.

3. The console will read the uPCs, then generate 24 micro steps, reading the uPC after each micro step. The uPCs are read in order; EBox, IBox, MBox, FBA, then FBM. Therefore, there will be 5 words (10 bytes) for each uPC. For instance, part of line 030 in block 9 (Example 2) reads: 0006 0004 0018 005F 1480. The uPCs, from left to right are: FMB, FBA, MBox, IBox, and EBox.

Table 13 Clock State Table

Signal Name	Expected State in Phase 3 0 1 2	Bit Set
FA14 CLK8 T13 DLY B H	0 1 0 1	0
FA14 CLK8 T13 DLY PW B H	0 1 0 1	0
FM13 CLK8 T13 DLY B H	0 1 0 1	1
FM13 CLK8 T13 DLY PW B H	0 1 0 1	1
-MCDT CLK3 T1A B H	1 1 0 1	2
-MCDT CLK3 T1B B H	1 1 0 1	2
-MCDT CLK3 T1C B H	1 1 0 1	2
-MCDT CLK3 T1D B H	1 1 0 1	2
IBDF CLK6 PHASE T0A H	0 1 0 0	3
IBDF CLK6 PHASE T0B H	0 1 0 0	3
IBDF CLK6 PHASE T0C H	0 1 0 0	3
IBDF CLK6 PHASE T0D H	0 1 0 0	3
-IDPD WBUS C CLK3 T0A H	1 0 1 1	4
IDPD CLK6 PHASE T0B H	0 1 0 0	4
-IDPD ENWBUS CLK3 T0C H	1 0 1 1	4
-IDPD EVPAR CLK3 T0D H	1 0 1 1	4
-ICAJ CLK6 PHASE T0A H	1 0 1 1	5
-ICAJ CLK3 T0B A H	1 0 1 1	5
-ICAJ CLK3 T0C B H	1 0 1 1	5
-ICAJ CLK6 PHASE T0D H	1 0 1 1	5
-ICBD CLK3 T0A B H	1 0 1 1	6
-ICBD CLK6 PHASE T0B H	1 0 1 1	6
-ICBD CLK3 T0C C H	1 0 1 1	6
-ICBD CLK6 PHASE T0D H	1 0 1 1	6
CLK3 CPU CLK STOP H	1 1 1 1	7
EDPG CLK6 PHASE T0A H	0 1 0 0	8
EDPG CLK6 PHASE T0B H	0 1 0 0	8
EDPH PHASE T0C DLY H	0 1 0 0	8
EDPG CLK6 PHASE T0D H	0 1 0 0	8

Table 13 Clock State Table (Cont)

EBEG CLK6 PHASE T0A H	0 1 0 0	9
EBEG CLK6 PHASE T0B H	0 1 0 0	9
EBEG CLK6 PHASE T0C H	0 1 0 0	9
EBEG CLK6 PHASE T0D H	0 1 0 0	9
-MCCK CLK3 T1A A H	1 1 0 1	10
-MCCK CLK3 T1B B H	1 1 0 1	10
MCCK CLK3 T1C A H	0 0 1 0	10
-MCCK CLK3 T1D C H	1 1 0 1	10
MAPN LD CLK3 T1A C H	0 0 1 0	11
-MAPN LD CLK3 T1B A H	1 1 0 1	11
-MAPN LD CLK3 T1C B H	1 1 0 1	11
-MAPN CLK6 PHASE T1D H	1 1 0 1	11
EBDF CLK6 PHASE T0A H	0 1 0 0	12
EBDF CLK6 PHASE T0B H	0 1 0 0	12
EBDF CLK6 PHASE T0C H	0 1 0 0	12
EBDF CLK6 PHASE T0D H	0 1 0 0	12
EBCG CLK6 PHASE T0A H	0 1 0 0	13
EBCG CLK6 PHASE T0B H	0 1 0 0	13
EBCG CLK6 PHASE T0C H	0 1 0 0	13
EBCG CLK6 PHASE T0D H	0 1 0 0	13
-CSBT CLK6 PHASE T0A H	1 0 1 1	14
-CSBT CLK6 PHASE T0B H	1 0 1 1	14
-CSBT CLK6 PHASE T0D H	1 0 1 1	14
-CSBT CLK6 PHASE T0C H	1 0 1 1	14
-CSAT CLK6 PHASE T0A H	1 0 1 1	15
-CSAT CLK6 PHASE T0B H	1 0 1 1	15
-CSAT CLK6 PHASE T0C H	1 0 1 1	15
-CSAT CLK6 PHASE T0D H	1 0 1 1	15

Example 1 - This is what a typical ERRSNAP.LOG looks like after it has been processed by VSR. In contrast, this is the same file illustrated in Example 2 (SHOW SNAP).

```
    -- Master Header --

CSM status - 5 = Non-EBox double error
KAF reason - 1B = CPU ERROR HALT
Time stamp  -  29-JAN-1987 11:48:20.59
Console PROM revision  - 37
EMM PROM revision      - 66
Minimum HW    revision - 0.4
Ebox Ucode    revision - 2.10
Ibox Ucode    revision - 3.73
Mbox Ucode    revision - 3.14
FboxA Ucode   revision - 2.74
FboxM Ucode   revision - 1.13
ACCESS Ucode  revision - 1.0
CPR Ucode     revision - 1.0
MCF Ucode     revision - 1.0
CONTEXT Ucode revision - 1.0
Ucode ReLease revision - 9.2
Ebox Ucode    verification - 0
MCF Ucode     verification - 0
CONTEXT Ucode verification - 0
FBOXA Ucode   verification - 0
FBOXM Ucode   verification - 0
FDRAM Ucode   verification - 0
IBOX Ucode    verification - 0
IDRAM Ucode   verification - 0
Mbox Ucode    verification - 0
CPR Ucode     verification - 0
ACCESS Ucode  verification - 0
PAMM          verification - 0
MCC shift channel (below)
    84 02 05 20 AA 00 52 A8 80 61 10


-- CSL in bytes --

MCSR0           1F
MCSR1           06
MCSR2           1E
MCSR3           FF
ERSR            C0
LRSR            C0
RRSR            00
spare           FF
QCSR0           28
QCSR1           00
QCSR2           00
QCSR3           42
SID0            97
SID1            F0
SID2            83
SID3            04
RL CTLR STATUS  0085
RL DRIVE STATU  009D
```

-- EMM data --

POWREG	3F
MARGEN	00
MARHILO	00
MODOK	8FFF
MISREG	40
SWREG	02
PROM revision	42
REGULATOR_A VOLTAGE	5.051
REGULATOR_B VOLTAGE	5.078
REGULATOR_C VOLTAGE	5.051
REGULATOR_D VOLTAGE	-2.015
REGULATOR_E VOLTAGE	-2.015
REGULATOR_F VOLTAGE	-5.244
REGULATOR_H VOLTAGE	-5.244
GND CURRENT VALUE	70.000
REGULATOR_L + VOLTAGE	12.192
REGULATOR_L - VOLTAGE	-11.954
REGULATOR_K + VOLTAGE	15.316
REGULATOR_K - VOLTAGE	-15.409
T1 TEMPERATURE VOLTAGE	24.991
T2 TEMPERATURE VOLTAGE	26.323
T3 TEMPERATURE VOLTAGE	25.324
T4 TEMPERATURE VOLTAGE	28.321

-- IPRs in Longwords --

KSP	80000C40
ESP	00000000
SSP	00000000
USP	00000000
ISP	8056956C
P0BR	80000000
P0LR	00000000
P1BR	7F802000
P1LR	00200000
SBR	027DC000
SLR	00009000
PCBB	0270CFA0
SCBB	027D9400
IPL	0000001F
ASTLVL	00000004
SISR	00000000
ICCS	800000C1
ICR	FFFFE3BE
TODR	1EAC436A
RXCS	000E0040
RXDB	0001008D
TXCS	00000FC0
ACCS	00008001
MAPEN	00000001
PME	00000000
SID	0483F097
PAMACC	26000000
PAMLOC	26000000
CSWP	00000003

-- IPRs in Longwords (Cont) --

MDECC	00060000
MENA	00F80FF9
MDCTL	00000000
MCCTL	00000000
MERG	00000100
EHSR	00180060
STXCS	010000C4
STXDB	000000DD
VPCBITS	00000004
EDPSR	00000000
EBCS	00000004
CSLINT	040C0116
EDMC	041F4404
IBESR	00000000
EMD	FFFFFF01
IVASAV	80569598
VIBASAV	80328B08
ESASAV	80328AF5
ISASAV	80328B00
CPC	80328B00
MSTAT1	84000004
MSTAT2	00000202
MEDR	FFFFFF01
MEAR	0000007C
CSHCTL	00000003
CSES	FFFFFFFF
IBGPR	0000000C
PSL	041F0004
SPADR	00000006
STATE	00000004
EVMQSAV	8056956C

-- E box Scratchpad --
General Purpose Registers

R0:	00000000	R1:	00000000	R2:	00000000	R3:	00000000
R4:	00000000	R5:	00000000	R6:	00000000	R7:	00000000
R8:	00000000	R9:	00000000	R10:	00000000	R11:	00000000
AP:	80569594	FP:	00000000	SP:	8056956C	PC:	00000000

Temporaries:

10:	00200000	
11:	00000000	
12:	0000400C	
13:	00000001	
14:	00000000	
15:	00FC00F9	
16:	00000200	
17:	00000058	SFBYCT
18:	60001800	EHMSTS
19:	00000000	EVMQSAV
1A:	00202000	EBCS

```
 1B:  00000000   EDPSR
 1C:  04180000   CSLINT
 1D:  02006403   IBESR
 1E:  00000000   EBXWD1
 1F:  00000000   EBXWD2
 20:  80569598   IVASAV
 21:  80328B08   VIBASAV
 22:  80328AF5   ESASAV
 23:  80328B00   ISASAV
 24:  80328B00   CPC
 25:  84003004   MSTAT1
 26:  00000F00   MSTAT2
 27:  00060000   MDECC
 28:  00000100   MERG
 29:  00000003   CSHCTL
 2A:  0000007C   MEAR
 2B:  0000001F   MEDR
 2C:  FFFFFFFF   FBXERR
 2D:  FFFFFFFF   CSES
 2E:  80328AF5   PC
 2F:  041F0004   PSL
 30:  80329544
 31:  00000000
 32:  00000008
 33:  00000000
 34:  00000000
 35:  00000075
 36:  00000116
 37:  00000000
 38:  00000000
 39:  00000000
 3A:  0000000F
 3B:  00000010
 3C:  FFFFFFFF
 3D:  00000000
 3E:  AAAAAAAA
 3F:  66666666
 40:  00000000
 ""      ""
 ""      ""
 FF:  00000000
```

-- PAM --

```
00  00000000 00000000 00000000 00000000 02020202 02020202 02020202 02020202
20  03030303 04040404 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F
40  1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F
60  1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F
80  1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F
A0  1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F
C0  1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F
E0  1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F
100 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F
120 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F
140 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F
160 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F
180 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F
1A0 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F
```

-- PAM (Cont) --

```
1C0 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F
1E0 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F
200 18181818 18181818 18181818 18181818 18181818 18181818 18181818 18181818
220 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F
(Rest of PAM is all 1F)
```

-- Interrupt Stack --

Longword	0	00000000
Longword	1	00000000
Longword	2	00000000
Longword	3	00000000
Longword	4	00000000
"		"
"		"
Longword	95	FFFFFFFF
Longword	96	FFFFFFFF
Longword	97	FFFFFFFF
Longword	98	FFFFFFFF

-- SBI0 Longwords --

Longword	0	02800010
Longword	1	EE000000
Longword	2	1C000008
Longword	3	00000000
Longword	4	409C9BFC
Longword	5	0000000E
Longword	6	28007246
Longword	7	00000010
Longword	8	B09E27EA
Longword	9	0000000E
Longword	A	B08E7864
Longword	B	0000000E
Longword	C	00000000
Longword	D	00000000
Longword	E	00000000
Longword	F	00000000
Longword	10	00000000
Longword	11	00000000
Longword	12	00000000
Longword	13	00000000
Longword	14	00000000
Longword	15	00000000
Longword	16	00000000
Longword	17	00000000

-- SBI0 Longwords (Cont) --

Longword	18	00000000	
Longword	19	00000002	
Longword	1A	1C000000	
Longword	1B	00000000	
Longword	1C	00000000	
Longword	1D	0802000E	
Longword	1E	040F0000	
Longword	1F	00000000	
Longword	20	00000000	
Longword	21	FFFFFFFF	TR1
Longword	22	1C800000	
Longword	23	00001000	
Longword	24	00000000	
Longword	25	FFFFFFFF	TR2
Longword	26	1C800000	
Longword	27	00001000	
Longword	28	00000000	
Longword	29	00000028	TR3
Longword	2A	1C000000	
Longword	2B	00001000	
Longword	2C	00000000	
"		"	
"		"	
Longword	59	FFFFFFFF	TR15
Longword	5A	1C800000	
Longword	5B	00001000	
Longword	5C	00000000	

-- SBI1 Longwords --

Longword	0	FFFFFFFF	
Longword	1	FFFFFFFF	
Longword	2	FFFFFFFF	
Longword	3	FFFFFFFF	
Longword	4	FFFFFFFF	
"		"	
"		"	
Longword	59	FFFFFFFF	TR15
Longword	5A	FFFFFFFF	
Longword	5B	FFFFFFFF	
Longword	5C	FFFFFFFF	

-- Dumping unknown record (3C) in longwords.

Longword	0	00004810	
Longword	1	00000000	This is in reality the snapshot
Longword	2	14800000	record for clock alignment and
Longword	3	00180000	uPC trace. See Table 12 and 13.
Longword	4	00060004	VSR doesn't know about it at the
Longword	5	005F1481	present time.
"		"	
"		"	
Longword	3E	08E20006	
Longword	3F	0018005F	
Longword	40	00060004	

Example 2 - This example illustrates what a typical snap file will look like when displayed on the console terminal via the "SHOW SNAP" command. The data is first translated into hexadecimal and output by virtual blocks (0 thru 9).

The gibberish to the right is an ASCII translation of the data, to be used as a reference point if manual analysis is necessary. Every once in a while, mixed in with the gibberish, you will see a string of six ASCII characters that make sense. For example, on line number 090 of block 0, you will see "FBMC01". This is the CDF file or ASCII record name described in Table 2 (byte position 04 thru 09). Since this is the last piece of information in the data header record, it can be used to identify the beginning of the data that corresponds to that record. In this case the data for the FBM SDB visability channel.

Again using line 090 as an example, you will see the following string of characters in the data field:

3130 434D 4246

When read right to left (standard PDP-11 convention), these characters are the Hex equivalent of the ASCII string:

FBMC01

The bytes to the right of this field correspond to the record length (004A) which equals 74 bytes decimal, the status code (01) which indicates that the record is valid, and the header ID (21) which indicates the FBA SDB visibility channel. See Table 2.

This example was also designed to serve as an aid should manual snap file analysis become necessary. The individual records in the virtual blocks have been boxed in with a solid line. The headers for each record have also been boxed in. Thus, since the "SHOW SNAP" format is standard, if you need to analyze a snap file manually you can use this example as a map to locate the header and beginning of each record. In addition, the following EBox scratch pad register locations have been boxed in with a dotted line:

ESC Location	Virtual Block #	Contents
17-2F	3	Possible Partial Machine Check Stack Frame
0C0	4	CSM.STATUS
0D8	5	EHM.SP
0DA	5	EHSR
0E0	5	KSP
0E1	5	ESP
0E2	5	SSP
0E3	5	USP
0E4	5	ISP

NOTE

If the Error Handling Microcode (EHM) was in the process of building a Machine Check Stack Frame when the KAF occurred then you might find a partial stack frame in ESC Locations 17-2F. If the EHM built the stack frame, pushed it on the interrupt stack and called the VMS Machine Check Handler then you might find a partial Machine Check Stack Frame in in ESC 17-2F and a full stack frame on the Interrupt Stack (Virtual Block 6).

SNAP1.DAT block 0.

41CB	1B05	132A	5845	2E42	4D56	004A	0120	[..J.VMB.EX*....A]	000	Master
01D0	024A	030E	0349	020A	0004	4225	1EAC	[..%B....I...J...]	010	Header
0000	0000	0000	0902	0100	0100	0100	0100	[................]	020	
0205	20AA	0052	A880	6110	0000	0000	0000	[.......a..R.....]	030	
4246	004A	0121	FFFF	FFFF	4810	0709	8584	[.....H......J.FB]	040	
0E96	09D6	0BF3	0FF3	0516	0D9E	3130	4241	[AB01............]	050	FBA
0E05	0F0C	0E9D	0E33	0F03	0E4F	0FB6	0D87	[....O...3.......]	060	
0707	039F	038F	030F	030F	030A	0D4D	0D1D	[..M.............]	070	
0000	0000	0000	0000	0000	0000	06C1	03C7	[................]	080	
0228	3130	434D	4246	004A	0122	0000	0000	[....".J.FBMC01(.]	090	FBM
0B15	0408	0010	0818	0828	0618	0028	0418	[..(...(.........]	0A0	
0168	0983	0943	0941	0B55	0A51	0147	0AC7	[..G.Q.U.A.C...h.]	0B0	
0000	052B	042E	04EA	018B	00EC	048A	0429	[)...........+...]	0C0	
0123	0000	0000	0000	0000	0000	0000	0000	[..............#.]	0D0	
8000	0010	2880	2920	3430	4444	434D	002A	[*.MCDD04.).(....]	0E0	MCD
0000	0000	0000	0000	0004	C204	C1CA	4C8C	[.L..............]	0F0	
4644	4249	002A	0124	0000	0000	0000	0000	[........$.*.IBDF]	100	IBD
E166	D171	4321	0003	040D	2082	B9AD	3530	[05.........Cq.g.]	110	
6240	8720	0214	A99C	D4D6	CD60	CFA0	4545	[EE..`.........@b]	120	
6C74	CFE6	3230	4650	4449	002A	0125	030D	[..%.*.IDPF02..tl]	130	IDP
8EF1	AFC3	FEF1	EF67	EEEF	EC2F	EDE7	E18D	[..../...g.......]	140	
002A	0126	F7F7	E7FD	FBF1	1F5A	EFCD	CD2B	[+...z.......&.*.]	150	
CEBE	C089	34E8	B4F2	133C	3230	4841	4349	[ICAH02<....4....]	160	ICA
A000	24F0	8080	4223	8E32	99B0	8C1D	1B1A	[......2.#B...$..]	170	
3130	4642	4349	002A	0127	F060	C030	20A0	[..0.`.'.*.ICBF01]	180	ICB
1791	4F45	7292	15A2	B8CC	1D37	9DF2	B8C3	[....7......rEO..]	190	
0000	0000	0000	0000	0202	030D	080A	0204	[................]	1A0	
88CC	4343	6363	3130	454B	4C43	002A	0128	[(.*.CLKE01ccCC..]	1B0	CLK
0000	0000	0000	0000	0000	0000	0000	0300	[................]	1C0	
4445	002A	0129	0000	0000	0000	0000	0000	[..........).*.ED]	1D0	
C0D0	C070	C4A0	2028	4049	210D	3230	4350	[PC02..I@(...p...]	1E0	EDP
0000	0000	0000	0000	0000	0000	10D0	A010	[@...............]	1F0	

Example 2 SHOW SNAP Command - Sample Printout (Virtual Block 0)

SNAP1.DAT block 1.

4200	3230	4245	4245	002A	012A	0000	0000	[....*.*.EBEB02.B]	000	EBE
0000	0000	0400	0000	0001	4404	09C0	A012	[.....D.........]	010	
012B	0000	0000	0000	0000	0000	0000	0000	[..............+.]	020	
4921	4000	1E05	1E26	3130	4B43	434D	002A	[*.MCCK01&....@.I]	030	MCC
4111	BAA0	4A00	0083	024C	000C	0214	0202	[......L....J...A]	040	
4450	414D	002A	012C	0000	0000	0000	0000	[.........,.*.MAPD]	050	MAP
4003	0508	41C1	0040	008A	1006	2622	3230	[02"&....@..A...@]	060	
0000	0000	0000	0002	0002	0000	0002	C100	[................]	070	
43E1	70A4	3230	4444	4245	002A	012D	0000	[..-.*.EBDD02.p.C]	080	EBD
0002	0100	F068	8E03	F113	70C2	7571	3000	[.0qu.p....h.....]	090	
002A	012E	0000	0000	0000	0000	0101	0101	[..............*.]	0A0	
0A00	281C	4844	196C	0D60	3530	4343	4245	[EBCC05`.1.DH.(..]	0B0	EBC
0000	0000	0000	0000	0000	070E	0E00	0705	[................]	0C0	
3230	4242	5343	002A	012F	0000	0000	0000	[....../.*.CSBB02]	0D0	CSB
0804	001D	0912	090D	1010	09D0	A801	8248	[H...............]	0E0	
0000	0000	0000	0000	0000	0000	0000	0000	[................]	0F0	
0206	0409	0005	3230	4241	5343	002A	0130	[0.*.CSAB02......]	100	CSA
0000	0000	0000	0000	0000	0000	0000	0802	[................]	110	
544D	002A	0131	0000	0000	0000	0000	0000	[.........1.*.MT]	120	
0000	0000	D800	2044	3C08	A000	3130	424D	[MB01...<D.......]	130	MTM
0000	0000	0000	0000	0000	0000	0002	0000	[................]	140	
061F	6365	724C	5343	001E	0132	0000	0000	[....2...CSLrec..]	150	CSL
0085	0483	F097	4200	0028	FF00	C0C0	FF1E	[......(..B......]	160	
FF00	003F	6365	724D	4D45	0032	0133	009D	[..3.2.EMMrec?...]	170	EMM
85BE	8449	8349	02B7	01B8	00B7	4202	408F	[.@.B.....I.I...]	180	
8D1F	8C1B	8BDF	0AC9	89AD	08A0	0700	86BE	[................]	190	
0C40	6365	7252	5049	00FA	0134	8F25	8E1C	[..%.4...IPRrec@.]	1A0	IPR
956C	0000	0000	0000	0000	0000	0000	8000	[..............l.]	1B0	
0000	7F80	2000	0000	0000	8000	0000	8056	[V...............]	1C0	
9400	0270	CFA0	0000	9000	02D7	C000	0020	[....}.......p...]	1D0	
00C1	0000	0000	0000	0004	0000	001F	027D	[}...............]	1E0	
008D	000E	0040	1EAC	436A	FFFF	E3EB	8000	[......jC..@.....]	1F0	

Example 2 SHOW SNAP Command - Sample Printout (Virtual Block 1)

SNAP1.DAT block 2.

```
0000 0000 0001 0000 8001 0000 0FC0 0001  [................]  000
0003 2600 0000 2600 001F 0483 F097 0000  [.........&...&..]  010
0000 0000 0000 00F8 0FF9 0006 0000 0000  [................]  020
00DD 0100 00C4 0018 0060 0000 0100 0000  [......`.........]  030
0116 0000 0004 0000 0000 0000 0004 0000  [................]  040
9598 FFFF FF01 0000 0000 041F 4404 040C  [...D............]  050
8B00 8032 8B00 8032 8AF5 8032 8B08 8056  [V...2...2...2...]  060
007C FFFF FF01 0000 0202 8400 0004 8032  [2.............|.]  070
0004 0000 000C FFFF FFFF 0000 0003 0000  [................]  080
0135 8056 956C 0000 0004 0000 0006 041F  [.........1.V.5.]  090
0000 0000 0000 0000 6365 7243 5345 040A  [.ESCrec.........]  0A0  ESC
0000 0000 0000 0000 0000 0000 0000 0000  [................]  0B0
0000 0000 0000 0000 0000 0000 0000 0000  [................]  0C0
0000 0000 8056 9594 0000 0000 0000 0000  [.........V.....]  0D0
0000 0000 0020 0000 0000 0000 8056 956C  [1.V.............]  0E0
00FC 00F9 0000 0000 0000 0001 0000 400C  [.@..............]  0F0
0000 0000 6000 1800 0000 0058 0000 0200  [....X......`....]  100  MCK
0200 6403 0418 0000 0000 0000 0020 2000  [.............d..]  110  Stack
8032 8B08 8056 9598 0000 0000 0000 0000  [.........V...2.]  120  Frame
8400 3004 8032 8B00 8032 8B00 8032 8AF5  [..2...2...2..0..]  130
0000 0003 0000 0100 0006 0000 0000 0F00  [................]  140
FFFF FFFF FFFF FFFF 0000 001F 0000 007C  [|...............]  150
0000 0000 8032 9544 041F 0004 8032 8AF5  [..2.....D.2.....]  160
0000 0075 0000 0000 0000 0000 0000 0008  [............u...]  170
0000 0000 0000 0000 0000 0000 0000 0116  [................]  180
0000 0000 FFFF FFFF 0000 0010 0000 000F  [................]  190
0000 0001 0000 0000 6666 6666 AAAA AAAA  [....ffff........]  1A0
0000 0005 0000 0004 0000 0003 0000 0002  [................]  1B0
0000 0009 0000 0008 0000 0007 0000 0006  [................]  1C0
0000 000D 0000 000C 0000 000B 0000 000A  [................]  1D0
0000 FFFF 0000 00FF 0000 000F 0000 000E  [................]  1E0
4000 0000 4000 0003 0000 8000 0000 0080  [...........@...@]  1F0
```

Example 2 SHOW SNAP Command - Sample Printout (Virtual Block 2)

SNAP1.DAT block 3.

```
0000 0056 0000 0016 001F 0000 0000 0400  [.@.........V...]  000
0000 001D 0000 7FFF 0000 7FF0 0000 7F80  [................]  010
0000 0400 0000 00F0 0000 0040 0000 0020  [....@...........]  020
0000 00A0 0000 002D 0000 002B 0000 0017  [....+...-.......]  030
0F0F 0F0F 0000 0070 0000 0050 0000 004A  [J...P...p.......]  040
0000 0013 FFFF FFF0 0000 0FFF 0000 01FF  [................]  050
0000 0060 2008 0080 2000 0000 0000 0023  [#...........`...]  060
3030 3030 0300 0000 0000 007F 0000 00A1  [............0000]  070
00FF FFFF 0000 00EF 0000 2000 0000 3000  [.0..............]  080
3020 FF00 FC1F FFFF C000 01FF 0000 0015  [................0]  090
0000 0420 0000 0035 0000 FFF0 FFFF FFE0  [........5.......]  0A0
0000 003F 0400 0000 0000 0041 0000 4020  [.@..A.......?...]  0B0
0000 07FF C000 0000 0000 FF00 2000 FF00  [................]  0C0
0000 3C00 0001 0000 0000 0038 0000 004D  [M...8........<..]  0D0
0000 0021 0000 3F80 0000 0100 0000 0380  [.........?......]  0E0
0000 001F 000F 0000 0000 0085 0000 0055  [U...............]  0F0
0800 0000 0000 0033 0000 001C 03C0 0000  [.........3......]  100
0000 0090 0000 0047 0000 2020 0000 2D20  [.-.......G.......]  110
8000 FE00 0000 00E0 0000 0030 0000 00B0  [....0...........]  120
43DE 1D01 F0F0 3E8F 6666 C051 0003 3003  [.0..Q.ff.>.....C]  130
2002 3EC1 00FF 4180 3FC3 C200 8000 0000  [.......?.A...>..]  140
FFFF 8000 FFFF 0000 0000 407E 0000 4080  [[.@...@.........]  150
0000 0000 0000 0000 0000 3FFF 0000 0062  [b....?..........]  160
0000 0000 0000 0000 0000 0000 0000 0000  [................]  170
0000 0000 0000 0000 0000 0000 0000 0000  [................]  180
0000 0000 0000 0000 0000 0000 0000 0000  [................]  190
8056 956C 0000 0005 0000 0000 0000 0000  [............1.V.]  1A0  CSM.STS
0000 00C5 FFFF FF01 041F 0004 0000 0201  [................]  1B0
0000 0000 0000 0080 0000 0007 0000 0006  [................]  1C0
0000 0000 0000 0000 0000 0000 0000 0000  [................]  1D0
0000 0000 027F FE02 2600 0000 0000 0000  [.......&........]  1E0
0000 0000 0000 0000 0000 0000 0000 0000  [................]  1F0
```

Example 2 SHOW SNAP Command - Sample Printout (Virtual Block 3)

SNAP1.DAT block 4.

7FFA	1840	8056	956C	0000	0000	0000	0000	[.....l.V.@...]	000	EHM.SP
0000	0000	0000	8001	0000	0000	0018	0060	[`.............]	010	EHSR
0000	0000	8000	0C40	0483	F097	0000	0058	[X.....@.....]	020	
0270	CFA0	8056	9600	0000	0000	0000	0000	[.........V...p.]	030	Stack
0000	0000	8000	0000	0000	0000	027D	9400	[..}.....]	040	Pointers
0002	4000	027D	C000	0080	0000	7F80	2000	[.....}..@..]	050	
0000	0005	0000	0000	0000	0000	0000	0004	[.....]	060	
8056	95F8	0020	2000	0000	0000	0000	0000	[.....V.]	070	
8056	95F0	0000	0003	0000	0058	4000	1800	[...@X.....V.]	080	
0000	0000	0000	0000	0000	0000	0000	0000	[.....]	090	
724D	4150	040A	0136	0000	0000	0000	0000	[.....6...PAMr]	0A0	PAMM
0000	0000	0000	0000	0000	0000	0000	6365	[ec.....]	0B0	
0202	0202	0202	0202	0202	0202	0202	0000	[.....]	0C0	
1F1F	1F1F	1F1F	0404	0404	0303	0303	0202	[.....]	0D0	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	0E0	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	0F0	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	100	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	110	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	120	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	130	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	140	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	150	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	160	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	170	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	180	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	190	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	1A0	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	1B0	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	1C0	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	1D0	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	1E0	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	1F0	

Example 2 SHOW SNAP Command - Sample Printout (Virtual Block 4)

SNAP1.DAT block 5.

1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	000	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	010	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	020	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	030	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	040	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	050	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	060	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	070	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	080	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	090	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	0A0	
1818	1818	1818	1818	1818	1818	1818	1F1F	[.....]	0B0	SBIA0
1818	1818	1818	1818	1818	1818	1818	1818	[.....]	0C0	ADDRESS
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1818	[.....]	0D0	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	0E0	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	0F0	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	100	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	110	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	120	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	130	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	140	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	150	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	160	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	170	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	180	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	190	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	1A0	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	1B0	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	1C0	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	1D0	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	1E0	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	1F0	

Example 2 SHOW SNAP Command - Sample Printout (Virtual Block 5)

SNAP1.DAT block 6.

1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	000	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	010	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	020	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	030	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	040	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	050	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	060	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	070	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	080	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	090	
1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	1F1F	[.....]	0A0	
0000	0000	6365	7250	5349	026E	0137	1F1F	]..7.n.ISPrec....]	0B0	ISP
0000	0000	0000	0000	0000	0000	0000	0000	[.....]	0C0	
8056	95F0	0000	0000	0000	0000	0000	0000	[.....V.]	0D0	
8056	95F8	4000	1800	0000	0058	0000	0003	[....X.....@..V.]	0E0	
0200	6003	0438	0000	0000	0000	0020	2000	[.....8..`..]	0F0	
8000	A0D4	8000	A0CC	0000	0000	8000	8B1F	[.....]	100	
8400	6004	8000	A0C8	8000	A0C8	8000	8B1F	[.....`..]	110	
0000	0003	0000	0100	0006	0000	0000	0000	[.....]	120	
FFFF	FFFF	FFFF	FFFF	0000	001F	0000	007C	[|.....]	130	
0000	0000	8000	8B1F	0418	0000	8000	A0C8	[.....]	140	
FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	[.....]	150	
FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	[.....]	160	
FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	[.....]	170	
FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	[.....]	180	
FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	[.....]	190	
FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	[.....]	1A0	
FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	[.....]	1B0	
FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	[.....]	1C0	
FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	[.....]	1D0	
FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	[.....]	1E0	
FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	[.....]	1F0	

Example 2 SHOW SNAP Command - Sample Printout (Virtual Block 6)

SNAP1.DAT block 7.

```
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  000
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  010
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  020
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  030
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  040
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  050
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  060
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  070
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  080
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  090
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  0A0
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  0B0
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  0C0
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  0D0
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  0E0
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  0F0
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  100
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  110
0000 0280 0010 6365 7230 4253 017E 0138  [8...SB0rec.....]  120  SBIA0
000E 409C 9BFC 0000 0000 1C00 0008 EE00  [............@...]  130
000E B09E 27EA 0000 0010 2800 7246 0000  [..Fr.(.....'....]  140
0000 0000 0000 0000 000E B08E 7864 0000  [..dx............]  150
0000 0000 0000 0000 0000 0000 0000 0000  [................]  160
0000 0000 0000 0000 0000 0000 0000 0000  [................]  170
0002 0000 0000 0000 0000 0000 0000 0000  [................]  180
000E 0000 0000 0000 0000 1C00 0000 0000  [................]  190
FFFF 0000 0000 0000 0000 040F 0000 0802  [................]  1A0
FFFF 0000 0000 0000 1000 1C80 0000 FFFF  [................]  1B0
0028 0000 0000 0000 1000 1C80 0000 FFFF  [................]  1C0
FFFF 0000 0000 0000 0000 1C00 0000 0000  [................]  1D0
002A 0000 0000 0000 1000 1C80 0000 FFFF  [..............*.]  1E0
FFFF 0000 0000 0000 0000 1C00 0000 0000  [................]  1F0
```

Example 2 SHOW SNAP Command - Sample Printout (Virtual Block 7)

SNAP1.DAT block 8.

```
FFFF 0000 0000 0000 1000 1C80 0000 FFFF  [................]  000
FFFF 0000 0000 0000 1000 1C80 0000 FFFF  [................]  010
FFFF 0000 0000 0000 1000 1C80 0000 FFFF  [................]  020
FFFF 0000 0000 0000 1000 1C80 0000 FFFF  [................]  030
FFFF 0000 0000 0000 1000 1C80 0000 FFFF  [................]  040
FFFF 0000 0000 0000 1000 1C80 0000 FFFF  [................]  050
FFFF 0000 0000 0000 1000 1C80 0000 FFFF  [................]  060
0038 0000 0000 0000 1000 1C80 0000 FFFF  [..............8.]  070
FFFF 0000 0000 0000 0000 1C00 0000 0018  [................]  080
FF39 0000 0000 0000 1000 1C80 0000 FFFF  [..............9.]  090
FFFF FFFF FFFF FFFF 6365 7231 4253 017E  [..SBlrec........]  0A0  SBIA1
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  0B0
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  0C0
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  0D0
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  0E0
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  0F0
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  100
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  110
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  120
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  130
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  140
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  150
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  160
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  170
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  180
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  190
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  1A0
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  1B0
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  1C0
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  1D0
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  1E0
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  [................]  1F0
```

Example 2 SHOW SNAP Command - Sample Printout (Virtual Block 8)

SNAP1.DAT block 9.

FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	[.....]	000	
010E	013C	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	[.....<...]	010	uPC
0000	0000	0000	0000	4810	6365	7243	5055	[UPCrec.H.....]	020	Trace
0018	005F	1481	0006	0004	0018	005F	1480	[.._.....]_...]	030	
08E2	0006	0004	0018	005F	1482	0006	0004	[.....]_.....]	040	
0004	0018	005F	1488	0006	0004	0018	005F	[_.....]_.....]	050	
005F	0D27	0006	0004	0018	005F	0E67	0006	[..g._.....'._.]	060	
0006	0004	0018	005F	08D8	0006	0004	0018	[.....]_.....]	070	
0007	005F	1478	0006	0004	0018	005F	04C2	[.._.....x._...]	080	
147C	0006	0004	00D8	005F	147A	0006	0004	[....z._.....|.]	090	
0004	0064	005F	1480	0006	0004	004A	005F	[_.J.....]_.d...]	0A0	
005F	1482	0006	0004	0018	005F	1481	0006	[....]_.....]_.]	0B0	
0006	0004	0017	005F	08DA	0006	0004	0018	[.....]_.....]	0C0	
00A8	005F	04C2	0006	0004	00EC	005F	08E0	[.._.....]_...]	0D0	
147A	0006	0004	00F3	005F	1478	0006	0004	[....x._.....z.]	0E0	
0004	00AF	005F	147C	0006	0004	00AD	005F	[_.....|._.....]	0F0	
005F	1481	0006	0004	009D	005F	1480	0006	[....]_.....]_.]	100	
0006	0004	0018	005F	1482	0006	0004	0018	[.....]_.....]	110	
0000	0000	0000	0006	0004	0018	005F	08E2	[.._.....]	120	
0000	0000	0000	0000	0000	0000	0000	0000	[.....]	130	
0000	0000	0000	0000	0000	0000	0000	0000	[.....]	140	
0000	0000	0000	0000	0000	0000	0000	0000	[.....]	150	
0000	0000	0000	0000	0000	0000	0000	0000	[.....]	160	
0000	0000	0000	0000	0000	0000	0000	0000	[.....]	170	
0000	0000	0000	0000	0000	0000	0000	0000	[.....]	180	
0000	0000	0000	0000	0000	0000	0000	0000	[.....]	190	
0000	0000	0000	0000	0000	0000	0000	0000	[.....]	1A0	
0000	0000	0000	0000	0000	0000	0000	0000	[.....]	1B0	
0000	0000	0000	0000	0000	0000	0000	0000	[.....]	1C0	
0000	0000	0000	0000	0000	0000	0000	0000	[.....]	1D0	
0000	0000	0000	0000	0000	0000	0000	0000	[.....]	1E0	
0000	0000	0000	0000	0000	0000	0000	0000	[.....]	1F0	

Example 2 SHOW SNAP Command - Sample Printout (Virtual Block 9)

Key SDB Signals

The following is a list of key SDB signals that are used to analyze the SDB signals captured by the Console and put in a Snap File. The list is used by VSABLD (Venus snap file analysis) to evaluate the contents of Snap files up-line loaded to a remote host.

Although the list is primarily intended to aid automated analysis it can be very useful in analyzing the state of a hung machine. The list includes:

1. The SDB Visability Channel Number

2. A Note column that provides some information about the signal
   c = constant (normal or console stimulus)
   g = goodstate (no error conditions)
   t = testable (dependent on system state)

3. The expected state of the signal (given a good machine)

4. The Signal Name as it appears in the print set. Signals that are identifed with an astrisk () indicate an error condition. Because these are the signals that you will most likely want to look at first thay have been extracted and put in Table 15

Table 14 Key SDB Signals

	SDB VC#	Note	Expected State	Signal Name
*	12xxx	g	0	ABUS CPU BUF ERROR H
	12xxx	g	0	ABUS MEMORY LOCK H
	12xxx	t	0	ARRAY RD BUSY H
	24xxx	t	0	ARRAY RD BUSY H
*	15xxx	g	0	CL CPU PF INTR H
	15xxx	c	0	CL MASTER RESET H
	12xxx	c	1	CL SDB CNTRL S1 A H
	12xxx	c	1	CL SDB CNTRL S2 A H
	15xxx	c	0	CL UNHANG RESET H
	16xxx	c	0	CL UNHANG RESET H
	7xxx	c	0	CLC3 MARK STOP COND H
	7xxx	g	1	CLC4 CPU PHS 1 NXT H
	7xxx	g	1	CLC4 CPU PHS 2 NXT H
	7xxx	c	0	CLK2 START H
	7xxx	c	1	CLK3 CPU CLK STOP H
	7xxx	c	0	CLK3 EN EBOX MARK BIT H
	7xxx	c	0	CLK3 EN FIELD STOP H
	7xxx	c	0	CLK3 EN IBOX MARK BIT H
	7xxx	c	0	CLK3 EN MBOX MARK BIT H
	7xxx	c	0	CLK3 FORCE MARK BIT H

Table 14 Key SDB Signals (Cont)

	SDB VC#	Note	Expected State	Signal Name
	7xxx	c	0	CLK3 SET VCO FREQ H
	7xxx	c	0	CLK3 STOP FBOX IN T0 H
	7xxx	c	0	CLK3 STOP FBOX IN T1 H
	7xxx	c	1	CLK9 FBOX T3 CLK H
	7xxx	c	0	CLK9 STOP F IN PHS 0 H
	7xxx	c	0	CLK9 STOP F IN PHS 1 H
	7xxx	g	1	CLK9 WBUS T3 CLK H
	7xxx	c	0	CLKB VCO INIT H
	16xxx	c	0	CSA CSB CLK DATA CHNL H
	7xxx	c	0	CSA EBOX UMARK H
*	16xxx	g	0	CSA PAR ERR H
	17xxx	c	0	CSB CLK DATA CHNL H
	14xxx	c	0	CSB CNSL OP H
	14xxx	c	0	CSB CSPE RESET A H
	17xxx	c	0	CSB CSPE RESET B H
	17xxx	c	0	CSB FCT SEL 3A H
	17xxx	g	0	CSB HOLD ERR LTH A H
	14xxx	g	0	CSB USTK PAR ERR H
	17xxx	c	0	CSB WRITE PULSE A H
	16xxx	c	0	CSBR CNSL OP1 FLAG H
	16xxx	c	0	CSBR CNSL OP2 FLAG H
	16xxx	c	0	CSBR LOAD DIAG CNTR H
*	11xxx	g	0	EBC CPU PF INTR LVL3 H
	14xxx	c	0	EBC EBD MAST RST DLY H
	14xxx	c	0	EBC EBD UNHG RST DLY H
	11xxx	c	0	EBC EBE MAST RST DLY H
	11xxx	c	0	EBC FLIP WBUS PAR B0 H
	11xxx	c	0	EBC FLIP WBUS PAR B1 H
	11xxx	c	0	EBC FLIP WBUS PAR B2 H
	11xxx	c	0	EBC FLIP WBUS PAR B3 H
	5xxx	c	0	EBC IBOX MASTER RST H
	14xxx	c	0	EBC INSERT DIAG ERR H
*	11xxx	g	0	EBC MBOX INTR LVL3 H
	12xxx	c	0	EBC MBOX MASTER RST H
	12xxx	c	0	EBC MBOX UNHANG RST H
	15xxx	g	0	EBCG CLK6 PHASE T0A H
	15xxx	g	0	EBCG CLK6 PHASE T0B H
	15xxx	g	0	EBCG CLK6 PHASE T0C H
	15xxx	g	0	EBCG CLK6 PHASE T0D H
	15xxx	c	0	EBCH FLIP MCF RAM PAR H
	15xxx	c	0	EBD DIAG RST IBOX H
*	15xxx	g	0	EBD EBOX ERR LST CYC H
*	4xxx	g	0	EBD EBOX ERR TO IDP H
*	11xxx	g	0	EBD ECS PE FLAG H
*	11xxx	g	0	EBD ECS PE LST CYC H

Table 14 Key SDB Signals (Cont)

	SDB VC#	Note	Expected State	Signal Name
*	15xxx	g	0	EBD EDP PE FLAG A H
*	11xxx	g	0	EBD EDP PE FLAG H
*	11xxx	g	0	EBD EMCR PE FLAG H
*	11xxx	g	0	EBD MBOX FE FLAG H
*	11xxx	g	0	EBD USTK PE FLAG H
*	11xxx	g	0	EBD WBUS PE FLAG H
	14xxx	g	0	EBDF CLK6 PHASE T0A H
	14xxx	g	0	EBDF CLK6 PHASE T0B H
	14xxx	g	0	EBDF CLK6 PHASE T0C H
	14xxx	g	0	EBDF CLK6 PHASE T0D H
*	5xxx	g	0	EBE IBOX ERR LTH A H
*	6xxx	g	0	EBE IBOX ERR LTH B H
*	4xxx	g	0	EBE IBOX ERR LTH C H
*	10xxx	g	0	EBE IBOX ERR LTH D H
	11xxx	g	0	EBEG CLK6 PHASE T0A H
	11xxx	g	0	EBEG CLK6 PHASE T0B H
	11xxx	g	0	EBEG CLK6 PHASE T0C H
	11xxx	g	0	EBEG CLK6 PHASE T0D H
*	14xxx	g	0	EDP OPR PAR ERR H
	10xxx	g	0	EDPG CLK6 PHASE T0A H
	10xxx	g	0	EDPG CLK6 PHASE T0B H
	10xxx	g	0	EDPG CLK6 PHASE T0D H
	10xxx	g	0	EDPH PHASE T0C DLY H
	10xxx	c	0	EDPI DISA ALU AR BUS H
	10xxx	c	0	EDPI DISA BYTE 10 PAR H
	10xxx	c	0	EDPI DISA BYTE 32 PAR H
	10xxx	c	0	EDPI DISA SCE AR BUS H
	10xxx	c	0	EDPI FLIP WREG PAR H
	xxx	g	0	FA14 CLK8 T13 DLY B H
	xxx	g	0	FA14 CLK8 T13 DLY PW B H
*	14xxx	g	0	FBA FBOX PROBLEM B H
*	12xxx	g	0	FBA FBOX WRITE PROB H
*	10xxx	g	0	FBA FWBUS ABORT H
	xxx	c	0	FBM CLK 141 RESET H
*	xxx	g	0	FBM CS PAR ERROR H
*	xxx	g	0	FBM FDRAM PAR ERROR H
	1xxx	g	0	FM13 CLK8 T13 DLY B H
	1xxx	g	0	FM13 CLK8 T13 DLY PW B H
*	6xxx	g	0	IBD BUF DRAM PE H
*	6xxx	g	0	IBD BUF IBUF PE H
	5xxx	g	0	IBD DECODE ERROR H
*	3xxx	g	0	IBD4 DRAM PE H
*	3xxx	g	0	IBD5 IBUF PE H
*	3xxx	g	0	IBDA ERROR SAV H
	3xxx	g	0	IBDF CLK6 PHASE T0A H

Table 14 Key SDB Signals (Cont)

	SDB VC#	Note	Expected State	Signal Name
	3xxx	g	0	IBDF CLK6 PHASE T0B H
	3xxx	g	0	IBDF CLK6 PHASE T0C H
	3xxx	g	0	IBDF CLK6 PHASE T0D H
	4xxx	c	0	ICA FORCE AMUX OPAR H
	4xxx	c	0	ICA FORCE BMUX OPAR H
	4xxx	c	0	ICA FORCE GPR PE H
	6xxx	c	0	ICA FORCE RLOG PE H
*	11xxx	g	0	ICA ICS PE H
	6xxx	c	0	ICA MAS RES H
	5xxx	c	0	ICA1 SDB LD H
	5xxx	c	0	ICA5 DIAG UNSTALL H
	5xxx	c	0	ICA5 IBOX UMARK H
	5xxx	c	0	ICA7 DIAG RES 1LAT H
	5xxx	c	0	ICA7 FLUSH MAS RES 1 H
	5xxx	c	0	ICA7 FLUSH MAS RES 3 H
*	5xxx	g	0	ICA7 ICS PAR ERR H
	5xxx	c	0	ICA7 UPC MAS RES H
	5xxx	g	0	ICB ERR STALL H
*	11xxx	g	0	ICB IBUF PE H
*	11xxx	g	0	ICB IDRAM PE H
*	11xxx	g	0	ICB RLOG PE H
	6xxx	c	0	ICB6 FRC RLOG PE LAT H
*	6xxx	g	0	ICB6 RLOG PE 3LAT H
	6xxx	c	0	ICB8 FLUSH MAS RES 1 H
	6xxx	c	0	ICB8 FLUSH MAS RES 3 B H
	6xxx	c	0	ICB8 MAS RES 1 H
*	11xxx	g	0	IDP IAMUX ERR CODE 0 H
*	11xxx	g	0	IDP IAMUX ERR CODE 1 H
*	11xxx	g	0	IDP IAMUX PE H
*	11xxx	g	0	IDP IBMUX PE H
	4xxx	g	0	IDPD CLK6 PHASE T0B H
*	12xxx	g	0	MAPL TAG PERR H
*	12xxx	g	0	MAPL TAG W PERR H
	13xxx	g	0	MAPN LD CLK3 T1A C H
*	11xxx	g	0	MCC MBOX CS PE H
*	15xxx	g	0	MCC MBOX INTR H
	2xxx	c	0	MCC1 MBOX MAST RESET H
*	12xxx	g	0	MCC1 STACK ERR H
*	12xxx	g	0	MCC7 MBOX FTL ERR H
	12xxx	t	0	MCCJ ARRY TIMER BSY H
*	12xxx	g	0	MCCM CYC ERR SUM H
	2xxx	g	0	MCCM DIS MD DRIVERS H
*	13xxx	g	0	MCCM HLD ERR ADR REG H
*	2xxx	g	0	MCCM HLD ERR DAT REG H
*	12xxx	g	0	MCD3 ABUS DAT PERR H

Table 14 Key SDB Signals (Cont)

	SDB VC#	Note	Expected State	Signal Name
*	13xxx	g	0	MCDM ECC CORR ERROR H
*	12xxx	g	0	MCDM ECC ERROR ANY A H
*	12xxx	g	0	MCDM ECC ERROR FATAL H
	7xxx	c	0	TEST CLR A H
	7xxx	c	0	TEST CLR B H
	7xxx	c	0	VCO INIT H
	12xxx	c	1	-CLK RESET 141 A H
	3xxx	c	1	-CLK RESET 141 B H
	14xxx	c	1	-CLK RESET 141 C H
	11xxx	c	1	-CLK RESET 141 C H
	15xxx	c	1	-CLK RESET 141 D H
	17xxx	c	1	-CLK RESET 141 D H
	1xxx	c	1	-CLK RESET 141 E H
	17xxx	g	1	-CSAT CLK6 PHASE T0A H
	17xxx	g	1	-CSAT CLK6 PHASE T0B H
	17xxx	g	1	-CSAT CLK6 PHASE T0C H
	17xxx	g	1	-CSAT CLK6 PHASE T0D H
	17xxx	g	0	-CSB CS PAR OK A H
*	14xxx	g	1	-CSB ECS PAR ERR H
	16xxx	c	1	-CSBR FLIP USTK PAR H
	16xxx	g	1	-CSBT CLK6 PHASE T0A H
	16xxx	g	1	-CSBT CLK6 PHASE T0B H
	16xxx	g	1	-CSBT CLK6 PHASE T0C H
	16xxx	g	1	-CSBT CLK6 PHASE T0D H
*	14xxx	g	1	-EBC MCF RAM PAR ERR H
*	14xxx	g	1	-EDP RESULT PAR ERR H
	10xxx	c	1	-EDPI FLIP GPRA H
	10xxx	c	1	-EDPI FLIP GPRB H
*	6xxx	g	1	-FBA FBOX PROBLEM A H
	5xxx	c	1	-ICA1 ENA AMUX OPAR H
	5xxx	c	1	-ICA1 ENA BMUX OPAR H
	5xxx	c	1	-ICA1 ENA GPR PE H
	5xxx	c	1	-ICA1 ENA RLOG PE H
	5xxx	c	1	-ICA1 INJ ERR 1LAT H
	5xxx	c	1	-ICA7 FLUSH MAS RES 1 H
	5xxx	c	1	-ICA7 MAS RES 1 H
	5xxx	g	1	-ICAJ CLK3 T0B A H
	5xxx	g	1	-ICAJ CLK3 T0C B H
	5xxx	g	1	-ICAJ CLK6 PHASE T0A H
	5xxx	g	1	-ICAJ CLK6 PHASE T0D H
	6xxx	g	1	-ICBD CLK3 T0A B H
	6xxx	g	1	-ICBD CLK3 T0C C H
	6xxx	g	1	-ICBD CLK6 PHASE T0B H
	6xxx	g	1	-ICBD CLK6 PHASE T0D H
	4xxx	g	1	-IDPD ENWBUS CLK3 T0C H

Table 14 Key SDB Signals (Cont)

	SDB VC#	Note	Expected State	Signal Name
	4xxx	g	1	-IDPD EVPAR CLK3 T0D H
	4xxx	g	1	-IDPD WBUS C CLK3 T0A H
	13xxx	g	1	-MAPN CLK6 PHASE T1D H
	13xxx	g	1	-MAPN LD CLK3 T1B A H
	13xxx	g	1	-MAPN LD CLK3 T1C B H
*	12xxx	g	1	-MAPR TB PERR H
*	2xxx	g	1	-MCCJ TOT CYC ERR H
	12xxx	g	1	-MCCK CLK3 T1A A H
	12xxx	g	1	-MCCK CLK3 T1B B H
	12xxx	g	1	-MCCK CLK3 T1D C H
*	12xxx	g	1	-MCCM BYTWR CACH PERR H
*	12xxx	g	1	-MCCM DMA ERR H
*	2xxxx	g	1	-MCCM SET BAD DAT CHK H
	2xxx	g	1	-MCDT CLK3 T1A B H
	2xxx	g	1	-MCDT CLK3 T1B B H
	2xxx	g	1	-MCDT CLK3 T1C B H
	2xxx	g	1	-MCDT CLK3 T1D B H
	7xxx	c	1	-STOP ON PHS 3

Table 15 Key SDB Error Signals

	SDB VC#	Note	Expected State	Signal Name
*	12xxx	g	0	ABUS CPU BUF ERROR H
*	15xxx	g	0	CL CPU PF INTR H
*	16xxx	g	0	CSA PAR ERR H
*	17xxx	g	0	CSB HOLD ERR LTH A H
*	14xxx	g	0	CSB USTK PAR ERR H
*	11xxx	g	0	EBC CPU PF INTR LVL3 H
*	11xxx	g	0	EBC MBOX INTR LVL3 H
*	15xxx	g	0	EBD EBOX ERR LST CYC H
*	4xxx	g	0	EBD EBOX ERR TO IDP H
*	11xxx	g	0	EBD ECS PE FLAG H
*	11xxx	g	0	EBD ECS PE LST CYC H
*	15xxx	g	0	EBD EDP PE FLAG A H
*	11xxx	g	0	EBD EDP PE FLAG H
*	11xxx	g	0	EBD EMCR PE FLAG H
*	11xxx	g	0	EBD MBOX FE FLAG H
*	11xxx	g	0	EBD USTK PE FLAG H
*	11xxx	g	0	EBD WBUS PE FLAG H
*	5xxx	g	0	EBE IBOX ERR LTH A H
*	6xxx	g	0	EBE IBOX ERR LTH B H
*	4xxx	g	0	EBE IBOX ERR LTH C H
*	10xxx	g	0	EBE IBOX ERR LTH D H
*	14xxx	g	0	EDP OPR PAR ERR H
*	14xxx	g	0	FBA FBOX PROBLEM B H
*	12xxx	g	0	FBA FBOX WRITE PROB H
*	10xxx	g	0	FBA FWBUS ABORT H
*	xxx	g	0	FBM CS PAR ERROR H
*	xxx	g	0	FBM FDRAM PAR ERROR H
*	6xxx	g	0	IBD BUF DRAM PE H
*	6xxx	g	0	IBD BUF IBUF PE H
*	3xxx	g	0	IBD4 DRAM PE H
*	3xxx	g	0	IBD5 IBUF PE H
*	3xxx	g	0	IBDA ERROR SAV H
*	11xxx	g	0	ICA ICS PE H
*	5xxx	g	0	ICA7 ICS PAR ERR H
*	11xxx	g	0	ICB IBUF PE H
*	11xxx	g	0	ICB IDRAM PE H
*	11xxx	g	0	ICB RLOG PE H
*	6xxx	g	0	ICB6 RLOG PE 3LAT H
*	11xxx	g	0	IDP IAMUX ERR CODE 0 H
*	11xxx	g	0	IDP IAMUX ERR CODE 1 H
*	11xxx	g	0	IDP IAMUX PE H
*	11xxx	g	0	IDP IBMUX PE H
*	12xxx	g	0	MAPL TAG PERR H
*	12xxx	g	0	MAPL TAG W PERR H
*	11xxx	g	0	MCC MBOX CS PE H

Table 15 Key SDB Error Signals (cont)

	SDB VC#	Note	Expected State	Signal Name
*	15xxx	g	0	MCC MBOX INTR H
*	12xxx	g	0	MCC1 STACK ERR H
*	12xxx	g	0	MCC7 MBOX FTL ERR H
*	12xxx	g	0	MCCM CYC ERR SUM H
*	13xxx	g	0	MCCM HLD ERR ADR REG H
*	2xxx	g	0	MCCM HLD ERR DAT REG H
*	12xxx	g	0	MCD3 ABUS DAT PERR H
*	13xxx	g	0	MCDM ECC CORR ERROR H
*	12xxx	g	0	MCDM ECC ERROR ANY A H
*	12xxx	g	0	MCDM ECC ERROR FATAL H
*	14xxx	g	1	-CSB ECS PAR ERR H
*	14xxx	g	1	-EBC MCF RAM PAR ERR H
*	14xxx	g	1	-EDP RESULT PAR ERR H
*	6xxx	g	1	-FBA FBOX PROBLEM A H
*	12xxx	g	1	-MAPR TB PERR H
*	2xxx	g	1	-MCCJ TOT CYC ERR H
*	12xxx	g	1	-MCCM BYTWR CACH PERR H
*	12xxx	g	1	-MCCM DMA ERR H
*	2xxx	g	1	-MCCM SET BAD DAT CHK H

APPENDIX E

EBOX ERROR ARBITRATION NETWORK

This Appendix contains a Table and a Functional Block Diagram. The Table lists the Microcode Vector Addresses and Relative Priorities of all Interrupt and Exception Traps. The Functional Block Diagram illustrates the logic in the EBox that arbitrates, prioritizes, and generates the Microcode Vectors for VAX8600/8650 error conditions. This is a key block diagram because, with the exception of EBox and MBox Control Store Parity Errors, every Error Detection Network in the VAX8600/8650 (listed below) converge here as input.

Appendix F - EBox Error Detection Networks
Appendix G - FBox Error Detection Networks
Appendix H - IBox Error Detection Networks
Appendix I - MBox Error Detection Networks
Appendix J - SBIA Error Detection Networks

Thus, using these Appendices as a set you can, figure out the exact conditions that cause each type of error in the VAX8600/8650; determine how that type of error is reported to the EBox; and see how the EBox arbitrates the error and generates a Microtrap Vector address. From there, you can go to the Error Handling Microcode (EHM) Flows (Appendix A) and see how it handles that type of error. Then from the EHM Flows you can go to the VMS Machine Check Flows (Appendix C) and see how VMS responds to the error. And finally, you can go to Chapter 5 and see what the error looks like after it has been translated into an ASCII Report.

Table E-1 EHM Trap Vector Addresses And Relative Priorities

Exception Priorities and Microcode Vectors

Priority Main	Priority Sub	Micro Vector	Exception Type
0	0	08*	MBox Fatal Error
0	0	08*	EBox Fatal Error
			EBox Mem Req in Prog and EB Port Stat <3:0> =
-	-	--	0 - Normal (no problem)
0	1	08*	8 - TB PE
-	-	09	9 - Reserved
0	1	0A	A - TB Miss
0	1	0B	B - Access Violation
0	1	0C	C - Modify Bit Not Set
-	-	0D	D - Reserved
0	1	0E	E - Write Across Page Boundaries
0	1	0F	F - Unaligned Reference
1	0	10*	IB OP-PORT-WRT-Stall and IBox Error
			IB OP-PORT-WRT-Stall and OP Port Stat <3:0> =
-	-	--	0 - Normal (no problem)
1	1	10*	8 - TB PE
1	1	11	9 - FBox Write Problem
1	1	12	A - TB Miss
1	1	13	B - Access Violation
1	1	14	C - M Bit Not Set
1	1	15	D - I/O Physical Address
1	1	16	E - Write Access Page Boundaries
1	1	17	F - Unaligned Reference
2	0	1E*	IBox Error (during IBox CPC Sync)
2	0	1F*	IBox Error (during RLog Unwind)
			Miscellaneous Exceptions (handled at IRD time)
-	-	--	Reserved
3	1	01	Interger Overflow
3	2	02*	FBox Problem
-	-	--	Reserved
3	4	04*	MBox (MEAR) Full
3	5	05	Console Halt Pending
3	6A	06*	Internal (MBox) Interrupt
3	6B	06	External (I/O) Interrupt
3	7	07	Trace Pending
4	0	18*	E Fork Stall and IBox Error
			E Fork Stall and IB Port Stat <3:0> =
-	-	--	0 - Normal (no problem)
4	1	18*	8 - TB PE
-	-	--	9 - Reserved
4	1	1A	A - TB Miss
4	1	1B	B - Access Violation
4	1	1C	C - Reserved
4	0	1D	D - I/O Physical Address
-	-	n/a	E - Write Across Page Boundaries
4	1	1F	F - Unaligned Reference
5	0	10*	EBox Read IMD and IBox Error
			EBox Read IMD and OP Port Stat <3:0> =
-	-	--	0 - Normal (no problem)
5	1	10*	8 - TB PE
5	1	11	9 - FBox Write Problem
5	1	12	A - TB Miss
5	1	13	B - Access Violation
5	1	14	C - M Bit Not Set
5	1	15	D - I/O Physical Address
5	1	16	E - Write Across Page Boundaries
5	1	17	F - Unaligned Reference
6	0	18*	EBox ID Read and IBox Error
7	0	18*	EBox Read String and IBox Error
			EBox Read String and OP Port Stat <3:0> =
-	-	--	0 - Normal (no problem)
7	1	18*	8 - TB PE
7	1	19	9 - FBox Write Problem
7	1	1A	A - TB Miss
7	1	1B	B - Access Violation
7	1	1C	C - M Bit Not Set
7	1	1D	D - I/O Physical Address
7	1	1E	E - Write Across Page Boundaries
7	1	1F	F - Unaligned Reference

* Vectors marked with an asterisk call the Error Handling Microcode (EHM). See Appendix A. All other vectors call micro routines that are not discussed in this manual.

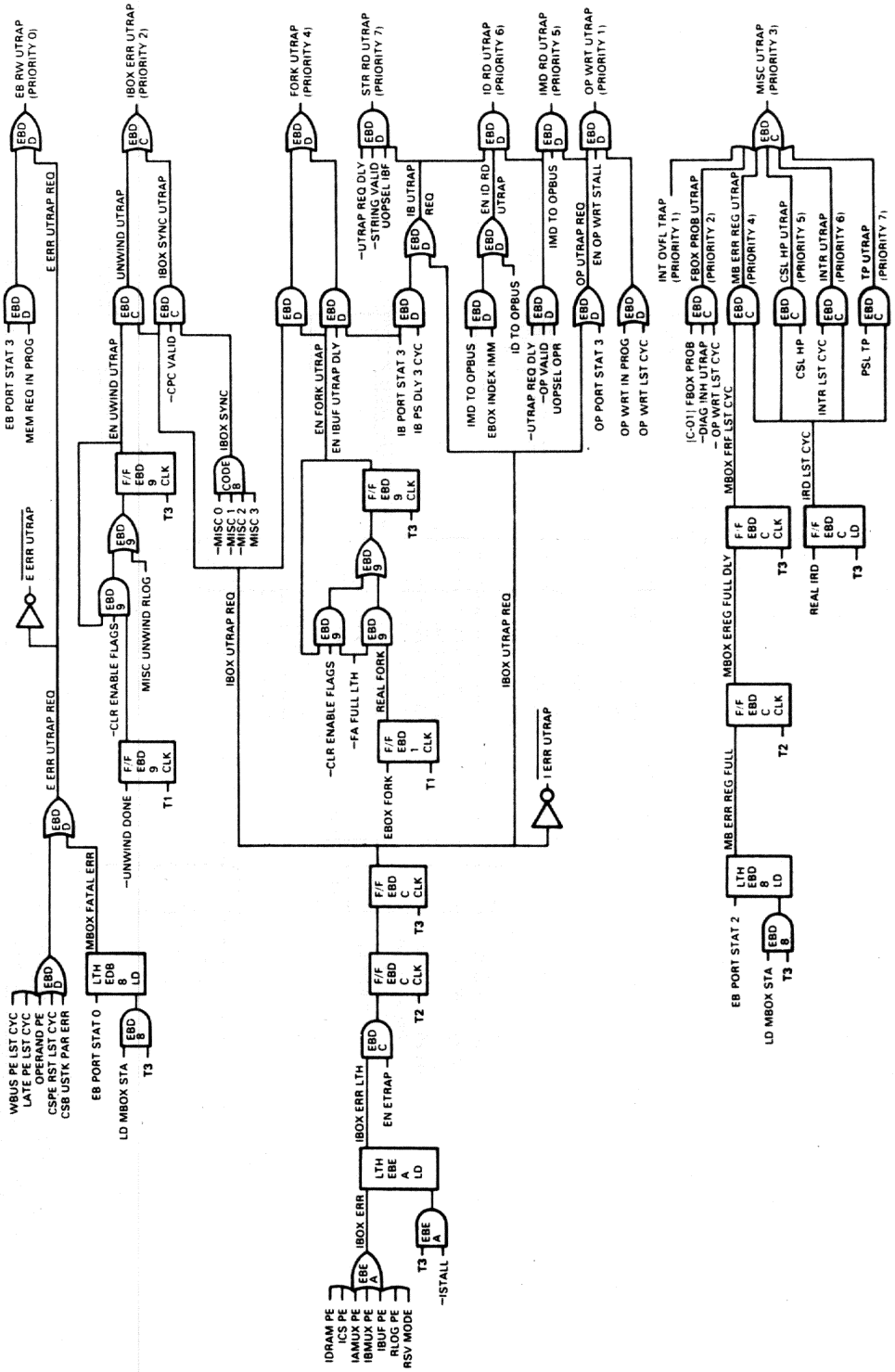


Figure E-1 EBox Error Arbitration Network (Part 1 of 2)

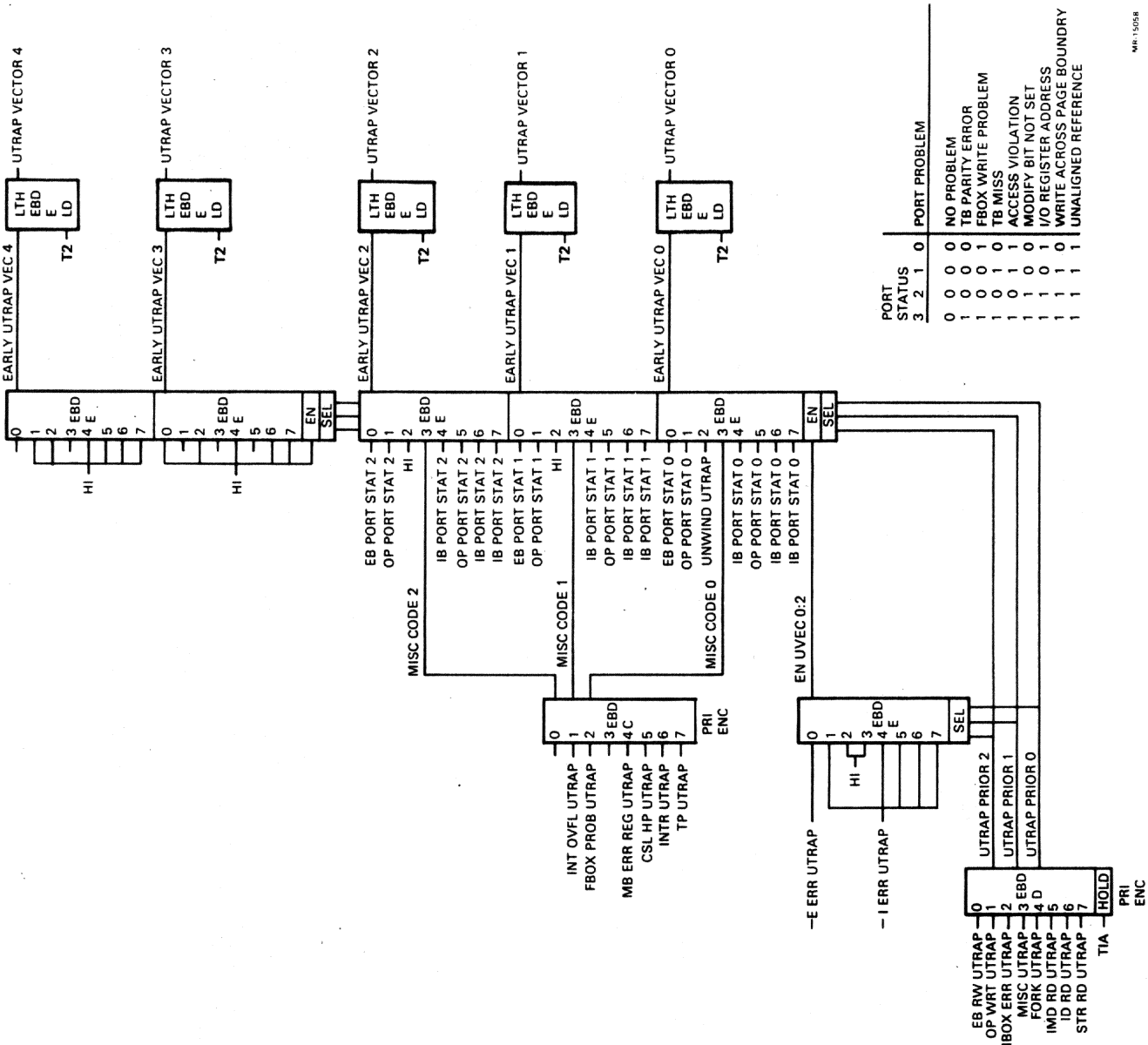


Figure E-1 EBox Error Arbitration Network (Part 2 of 2)

APPENDIX F

EBOX ERROR DETECTION NETWORKS

This Appendix contains a set Functional Block Diagrams that describe the EBox Error Detection Networks. The output of these networks go to the EBox Error Arbitration Network where they are prioritizd and used to generated an Error Handling Microcode Trap Vector Address. Table 1 (below) lists both the error conditions and the Figures that describes the networks that are used to detect the error conditions.

Table 1 EBox Error Detection Networks

Figure	Error Detection Networks
F-1	EBox AMux and BMux Parity Generation
F-2	EBox Operand Parity Error Detection Network
F-3	EBox Result Parity Error Detection Network and Late Parity Error Last Cycle
F-4	EBox WBus Parity Error Detection Network
F-5	EBox Micro-Stack Parity Error Detection Network

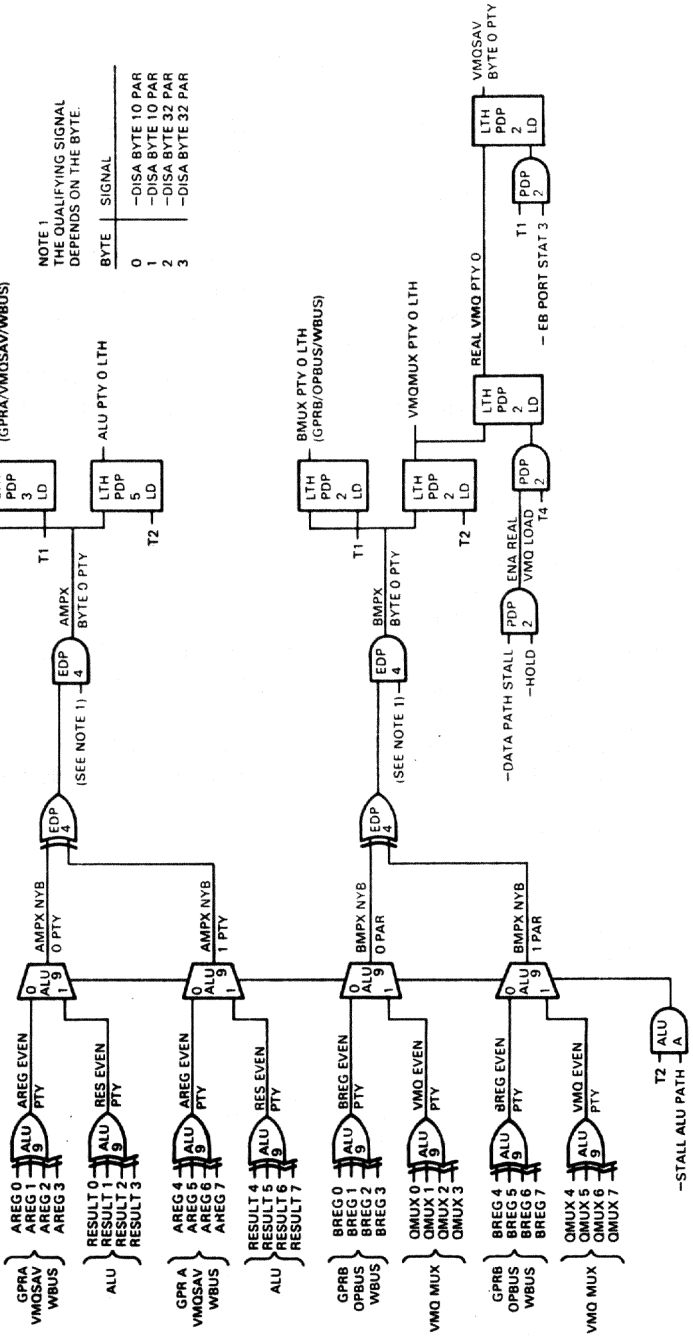


Figure F-1 EBox AMux and BMux Parity Generation

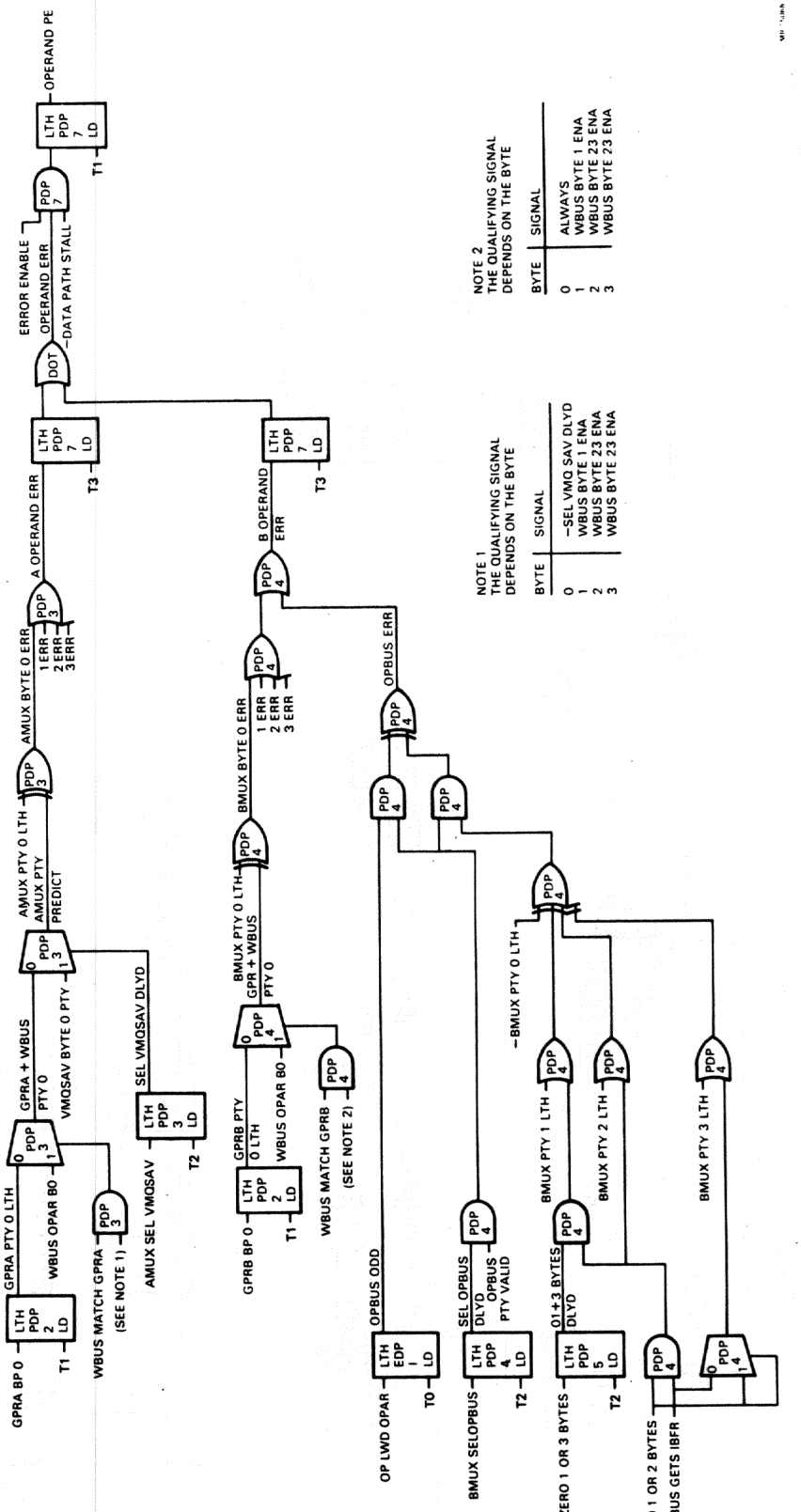


Figure F-2 EBox Operand Parity Error Detection Network

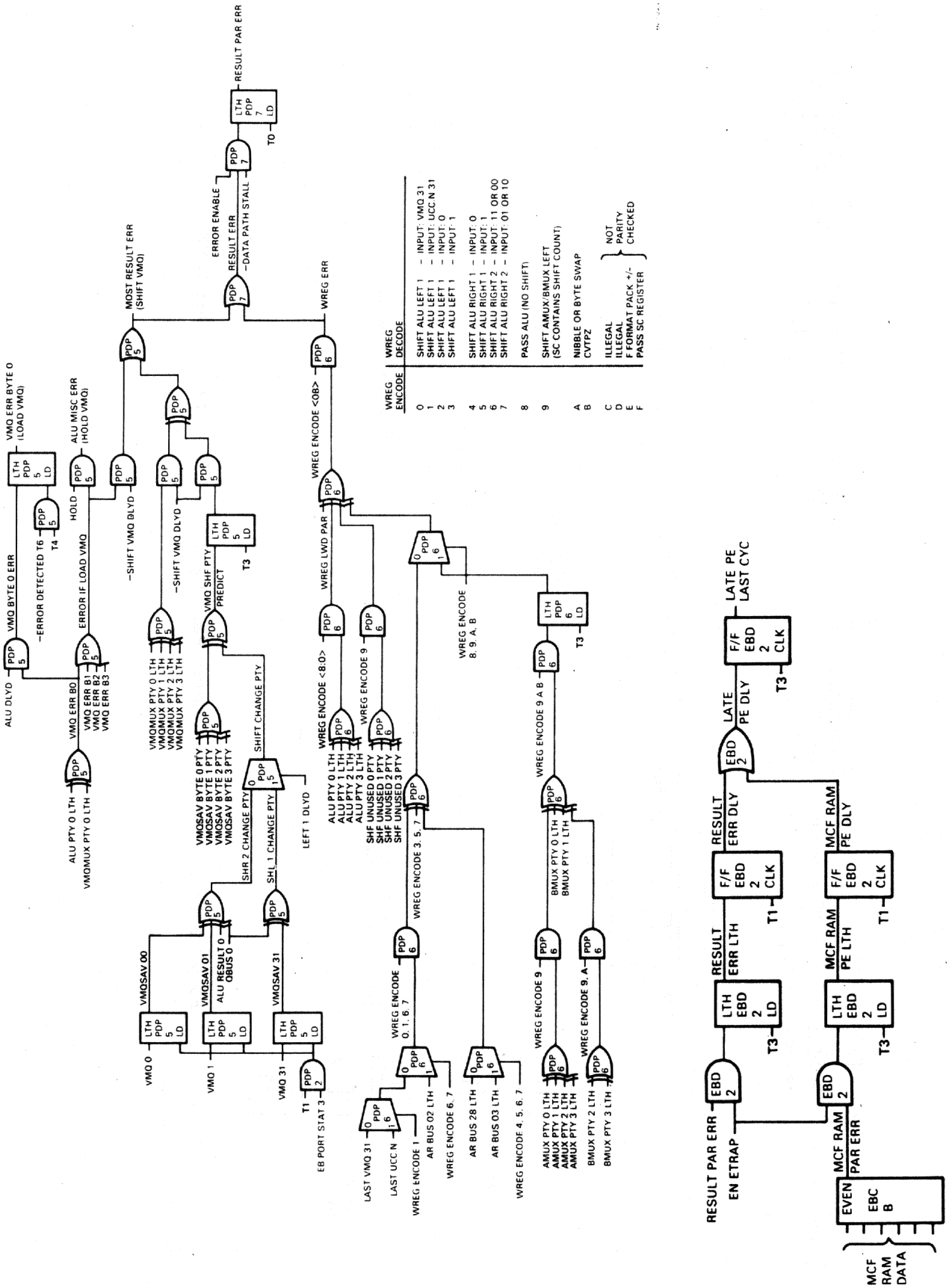


Figure F-3 EBox Result Parity Error Detection Network and Late Parity Error Last Cycle

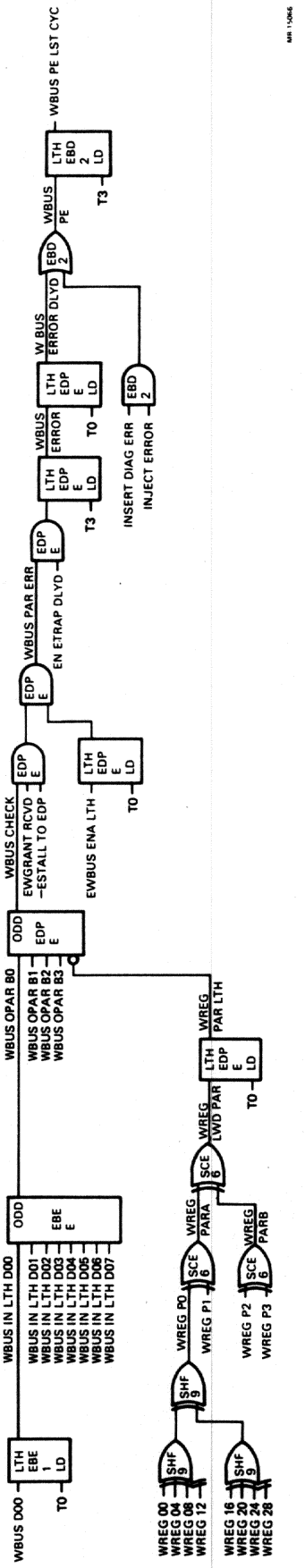


Figure F-4 EBox WBus Parity Error Detection Network

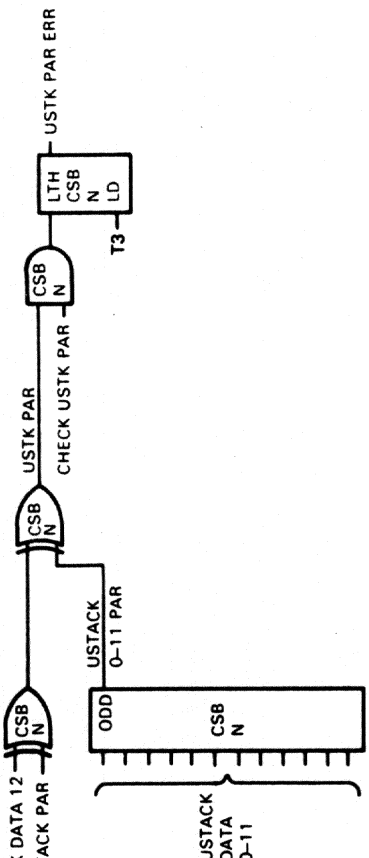


Figure F-5 EBox Micro-Stack Parity Error Detection Network

APPENDIX G

FBOX ERROR DETECTION NETWORKS

This Appendix contains a Functional Block Diagram that describe the FBox Error Detection Network. The network produces three outputs.

- FBox Problem - This output goes to the EBox Error Arbitration Network where it is prioritizd and used to generated an Error Handling Microcode Trap Vector Address.
- FBox Write Problem - This output goes to the MBox and prevents the MBox from writing the result into memory (Cache).
- FBox WBus Abort - This output goes to the EBox and prevents the EBox from writing the result in a GPR.

Table 1 FBox Error Detection Networks

Figure	Frror Detection Networks
G-1	FBox Problem Detection
G-1	FBox Write Problem
G-1	FBox WBus ABort

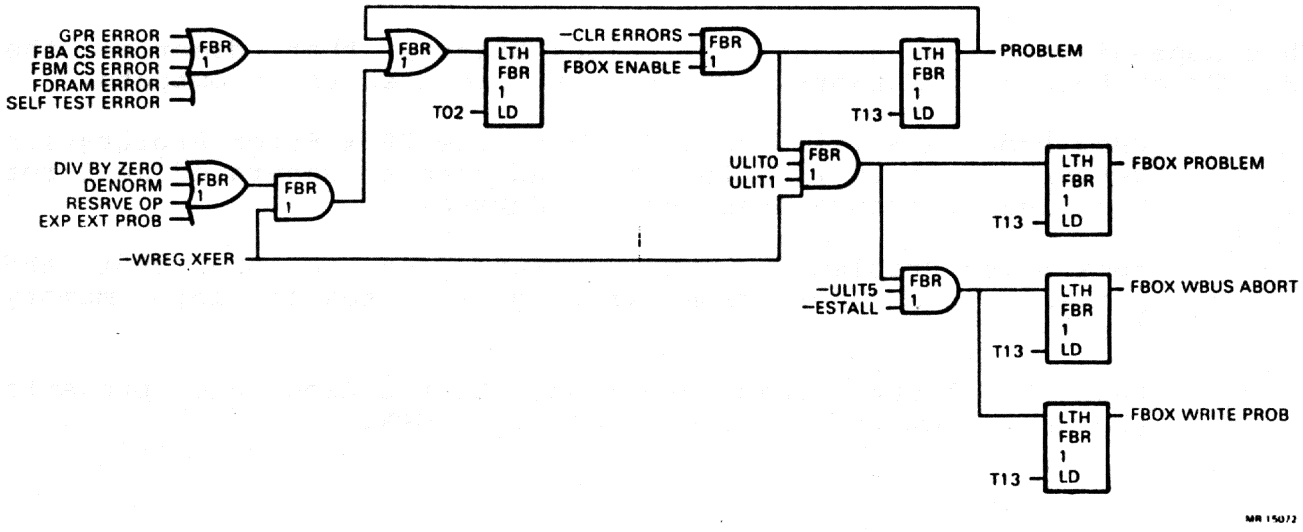


Figure G-1 FBox Problem Detection

APPENDIX H

IBOX ERROR DETECTION NETWORKS

This Appendix contains a set Functional Block Diagrams that describe the IBox Error Detection Networks. The output of these networks go to the EBox Error Arbitration Network where they are prioritizd and used to generated an Error Handling Microcode Trap Vector Address. Table 1 (below) lists both the error conditions and the Figures that describes the networks that are used to detect the error conditions.

Table 1 IBox Error Detection Networks

Figure	Error Detection Networks
H-1	IBox Instruction Buffer Parity Error Detection Network
H-2	IBox AMux Parity Error Detection Network
H-3	IBox BMux Parity Error Detection Network
H-4	IBox AMux Error Code Generation
H-5	IBox AMux WBus Data Generation
H-6	IBox Control Store Parity Error Detection Network
H-7	IBox Dispatch RAM Parity Error Detection Network
H-8	IBox OPBus Longword Parity Generation
H-9	IBox Rlog Parity Error Detection Network

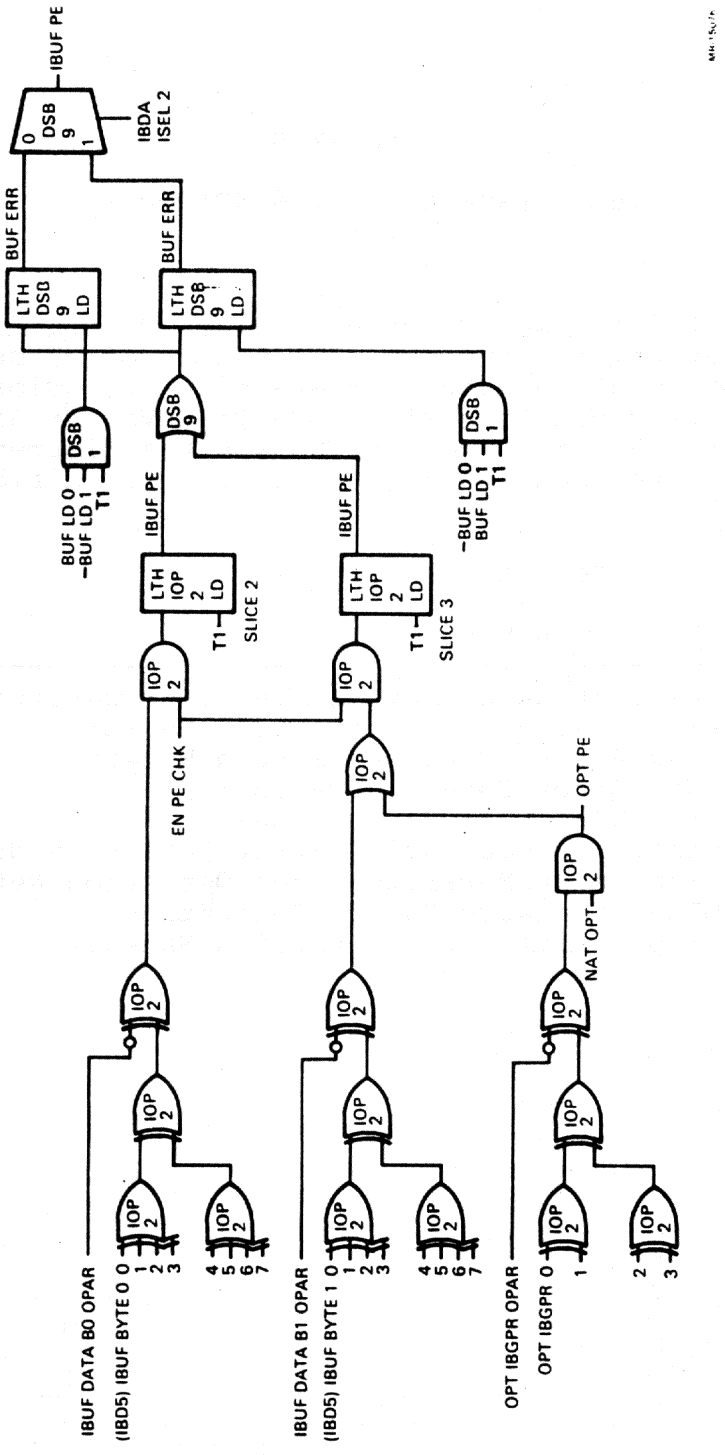


Figure H-1 IBox Instruction Buffer Parity Error Detection Network

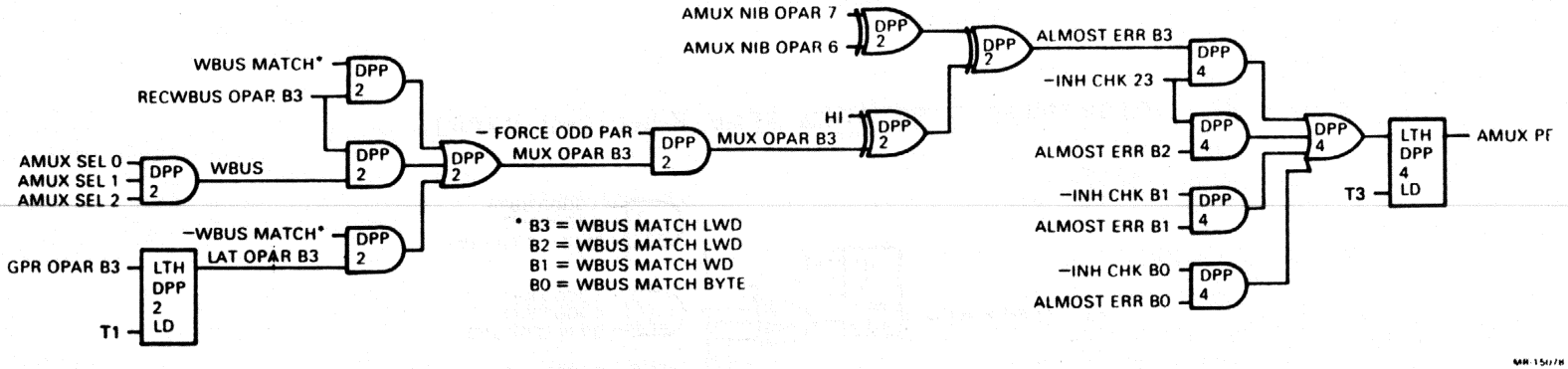


Figure H-2 IBox AMux Parity Error Detection Network

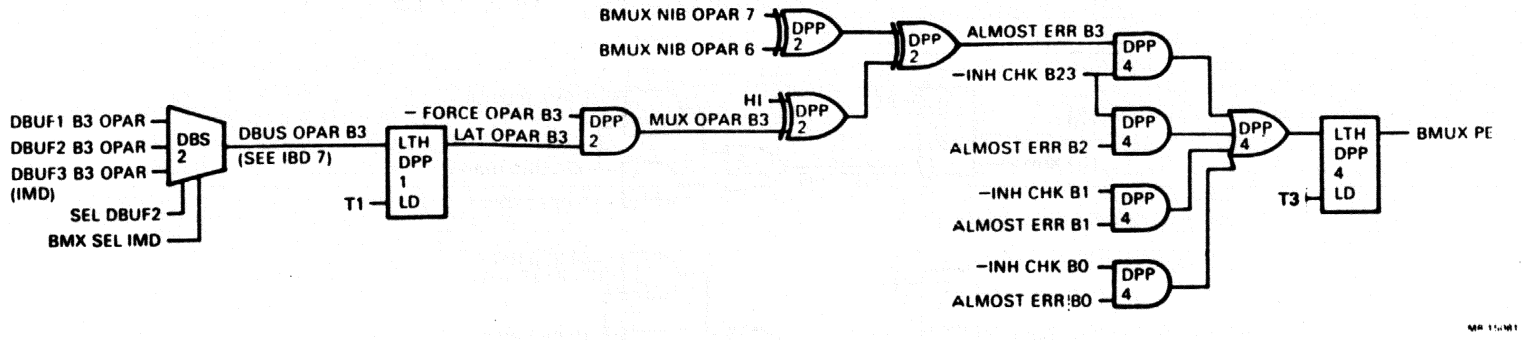


Figure H-3 IBox BMux Parity Error Detection Network

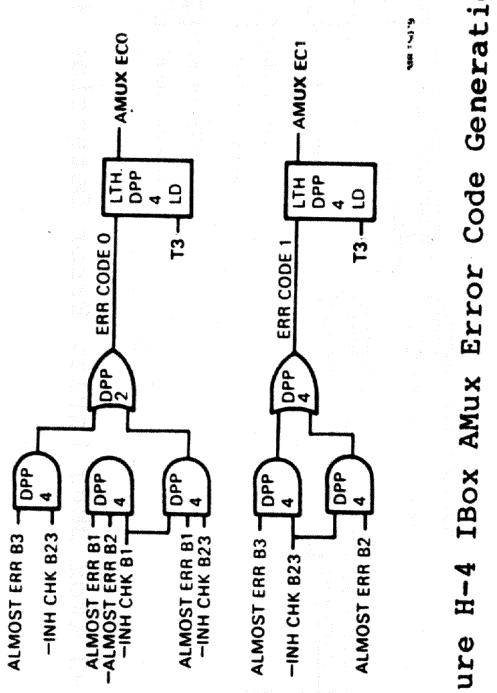


Figure H-4 IBox AMux Error Code Generation

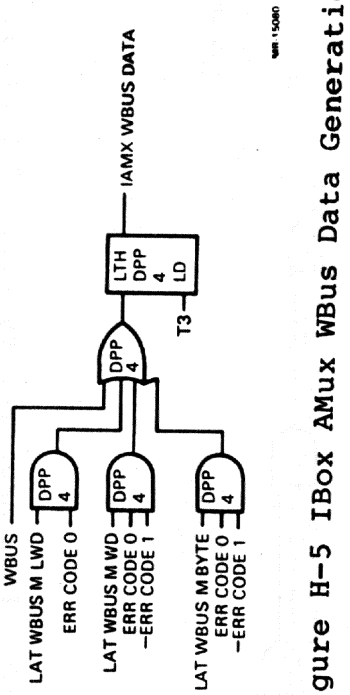


Figure H-5 IBox AMux WBus Data Generation

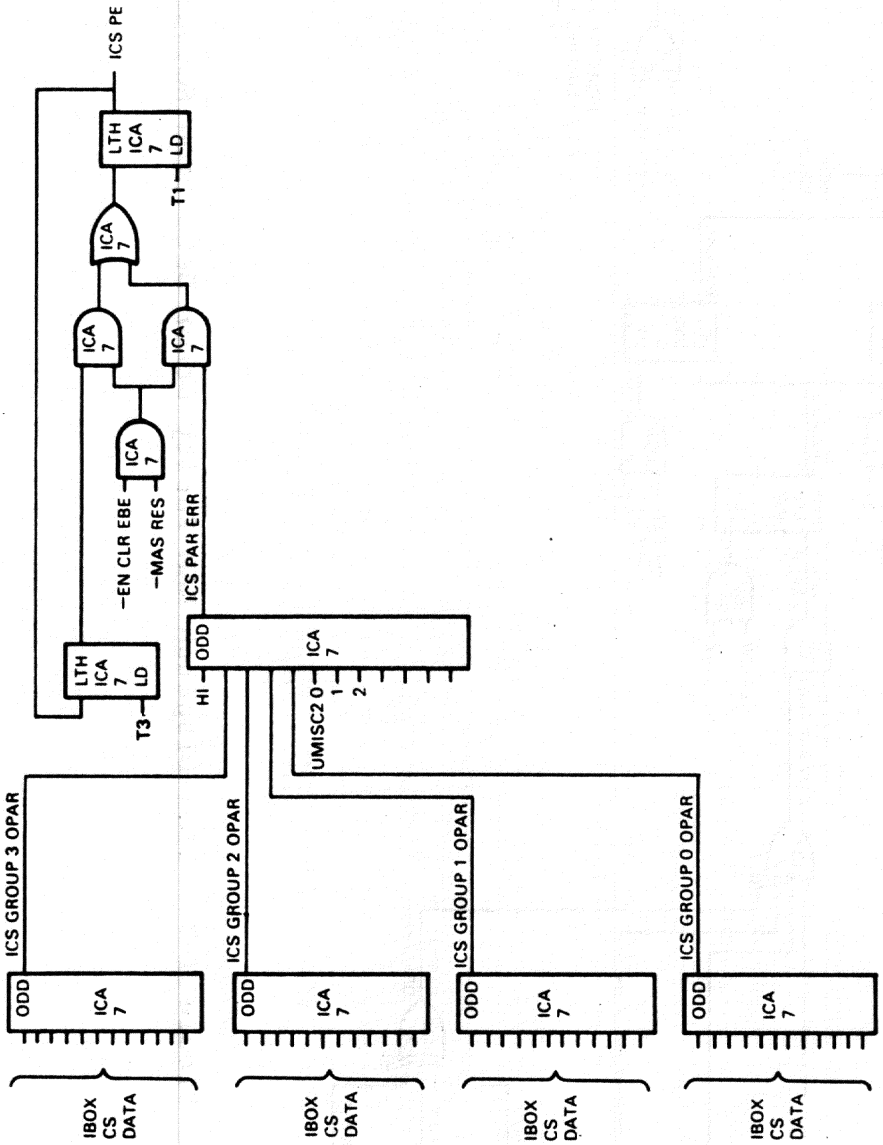


Figure H-6 IBox Control Store Parity Error Detection Network

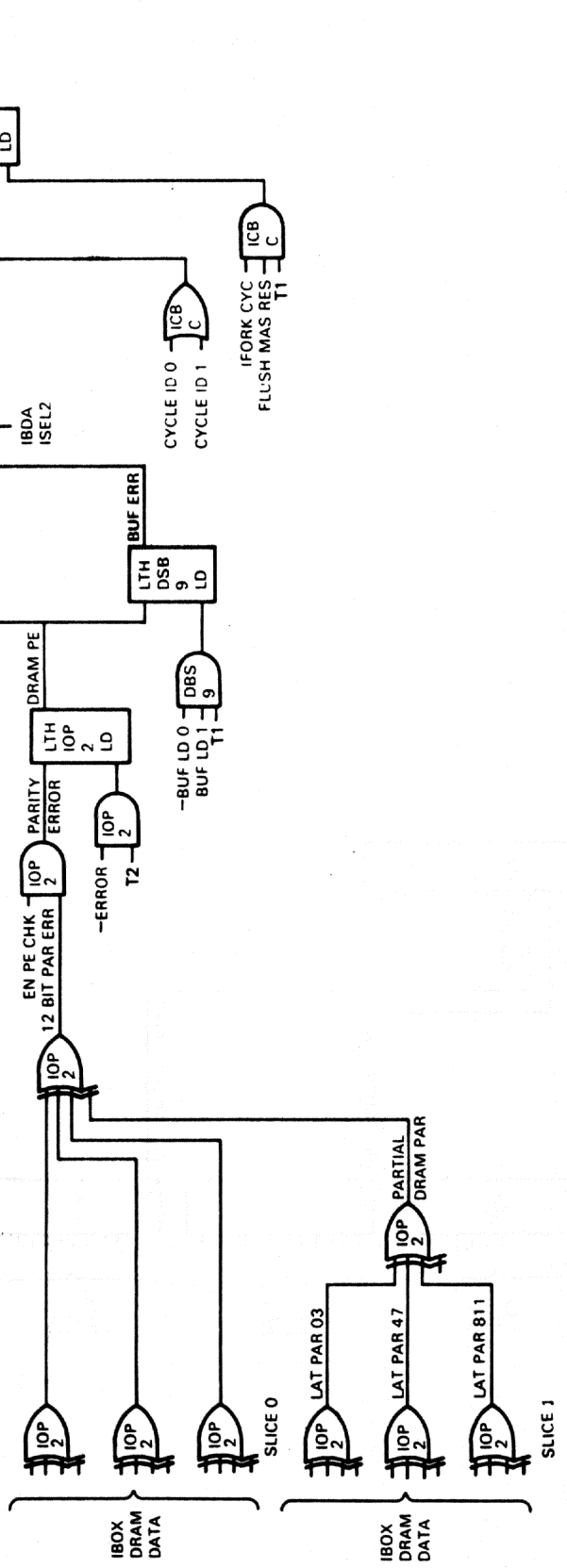


Figure H-7 IBox Dispatch RAM Parity Error Detection Network

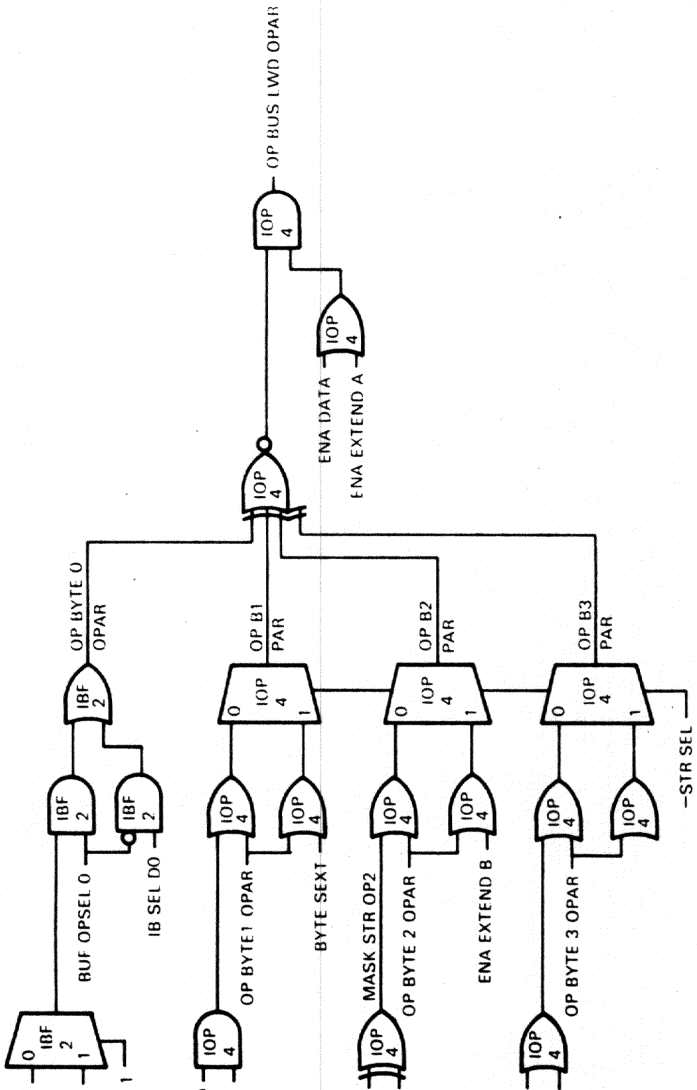


Figure H-8 IBox OPBus Longword Parity Generation

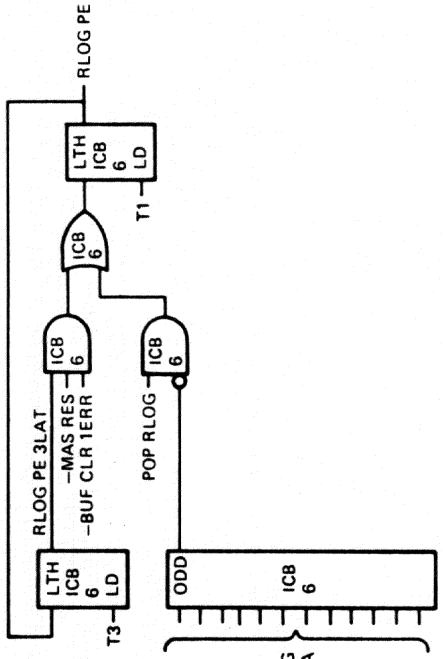


Figure H-9 IBox Rlog Parity Error Detection Network

APPENDIX I

MBOX ERROR DETECTION NETWORKS

This Appendix contains a set Functional Block Diagrams that describe the MBox Error Detection Networks. The output of these networks go to the EBox Error Arbitration Network where they are prioritizd and used to generated an Error Handling Microcode Trap Vector Address. Table 1 (below) lists both the error conditions and the Figures that describes the networks that are used to detect the error conditions.

Table 1 MBox Error Detection Networks

Figure	Error Detection Networks
I-1	MBox Fatal Error, Error Reg Full, and Interrupt Generation
I-2	MBox Cycle Error Sum Generation
I-3	MBox Internal Error Sum 28 Generation
I-4	MBox Internal Error Sum 58 Generation
I-5	MBox Multiple Error and Held Error Generation
I-6	MBox ECC Error Detection Network
I-7	MBox Cache Tag Parity Error Detection Network
I-8	MBox Cache Tag WBit Parity Error Detection Network
I-9	MBox Cache Data Correction, Cache Data Parity Error and Byte Write CP Error Detection Network
I-10	MBox CP Write and Write Data Parity Error Detection Network
I-11	MBox Write Byte and Byte Parity Error Detection Network
I-12	MBox CP NXM Error Detection Network
I-13	MBox CP Buffer Error Detection Network
I-14	MBox Control Store Error Parity Detection Network
I-15	MBox CPR Parity Error Detection Network
I-16	MBox ABus Address Parity Error Detection Network
I-17	MBox ABus Control and Mask Parity Error Detection Network
I-18	MBox ABus Data Parity Error Detection Network
I-19	MBox ABus Bad Data Error Detection Network

Table 2 MBox Error Handling Flow Charts

Figure	Flow Chart Title
I-20	MBox Array Read Data Error Handling Flow Chart
I-21	MBox Cache Read Data Error Handling Flow Chart
I-22	MBox Cache Writeback Error Handling Flow Chart

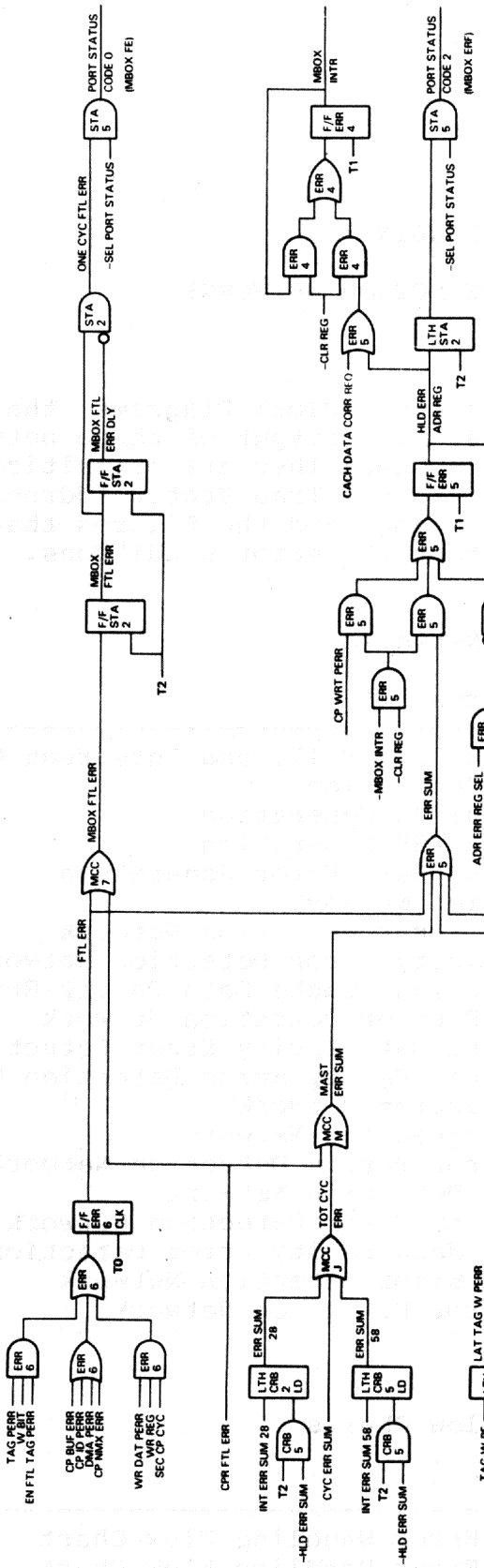


Figure I-1 MBox Fatal Error, Error Reg Full, and Interrupt Generation

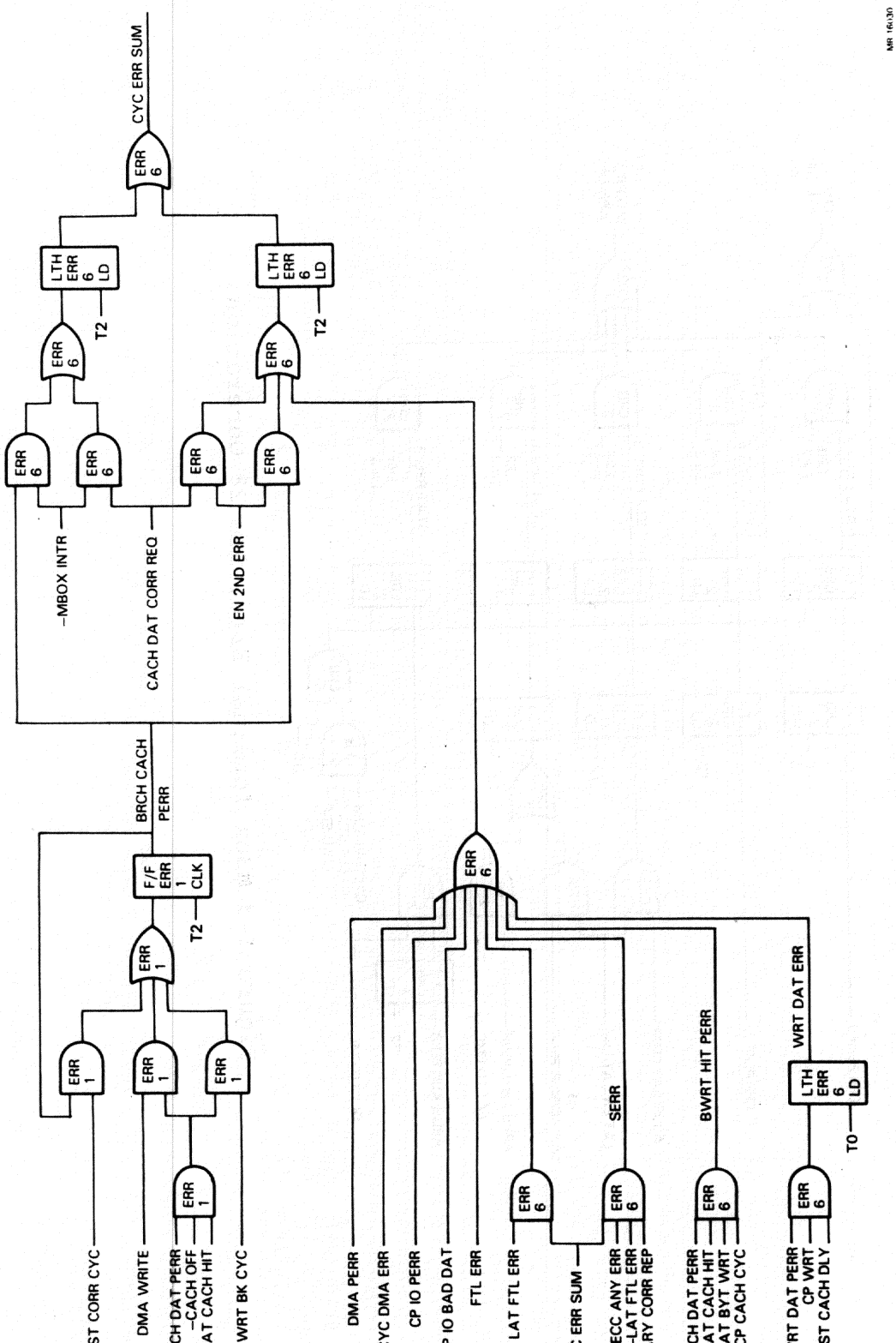


Figure I-2 MBox Cycle Error Sum Generation

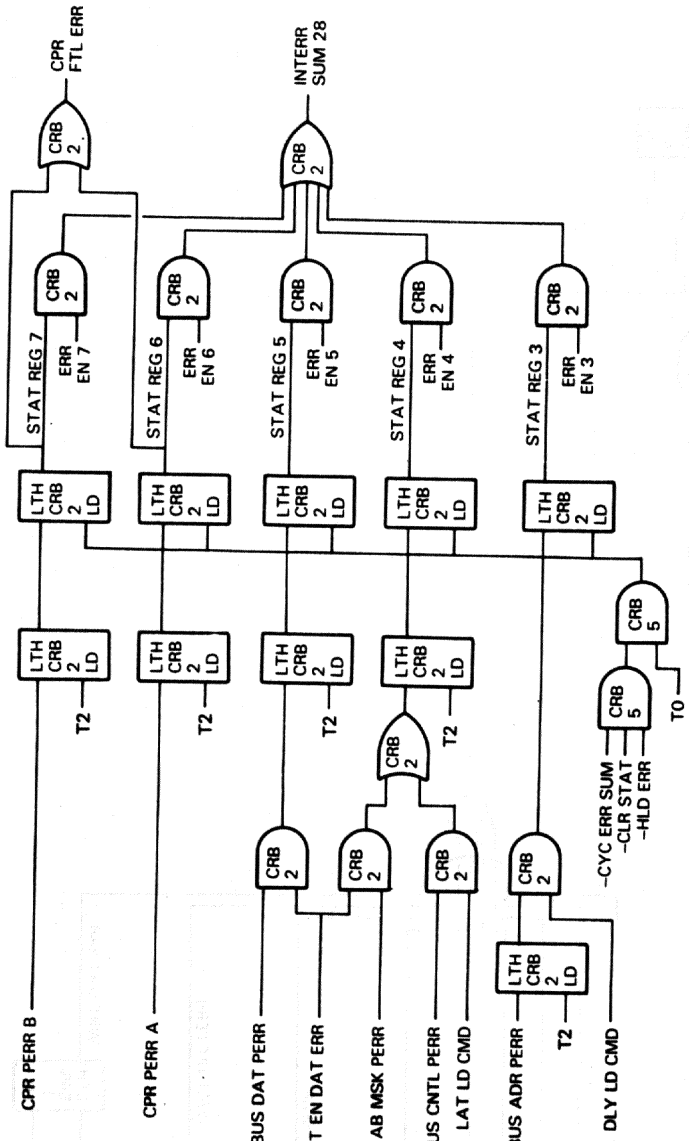


Figure I-3 MBox Internal Error Sum 28 Generation

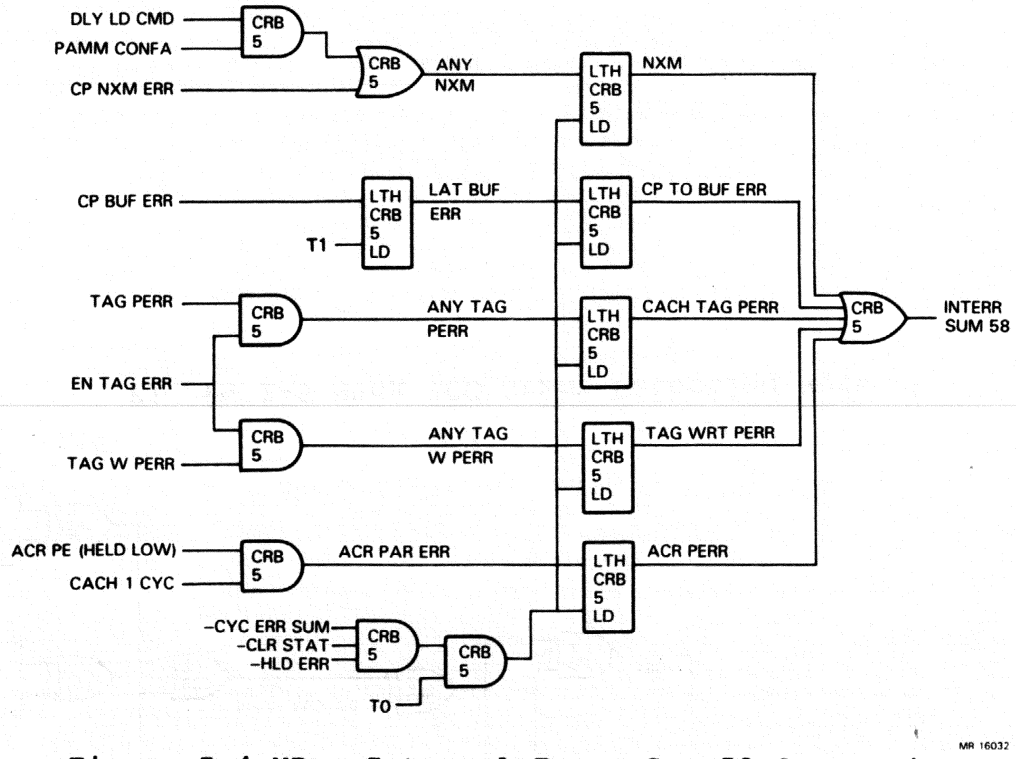


Figure I-4 MBox Internal Error Sum 58 Generation

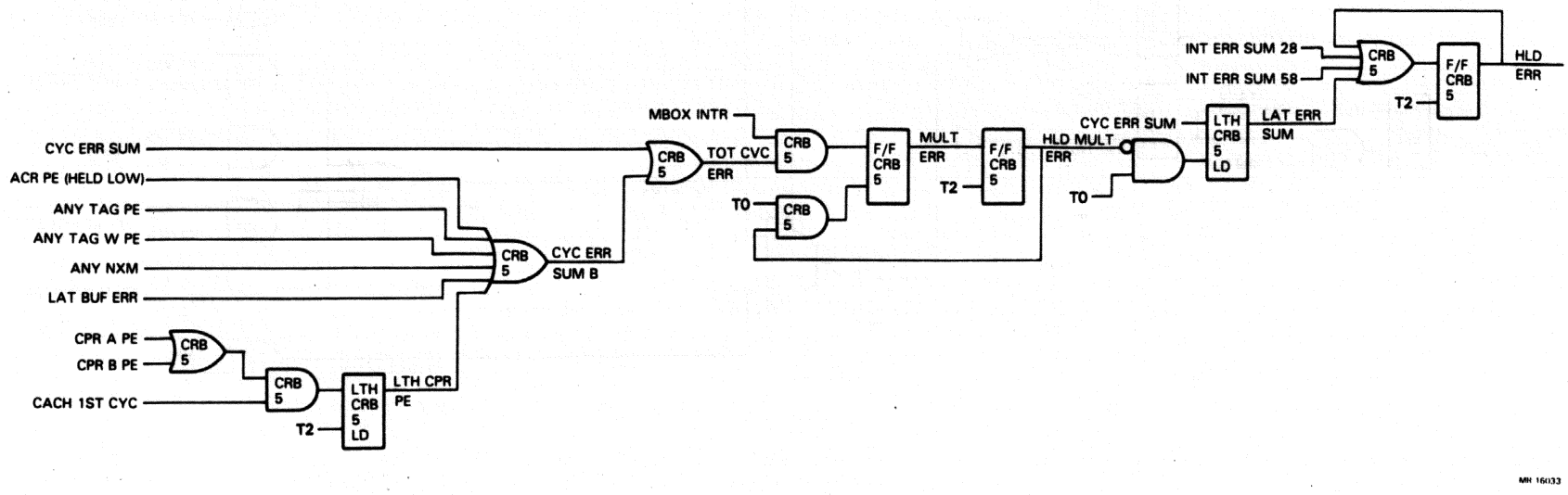


Figure I-5 MBox Multiple Error and Held Error Generation

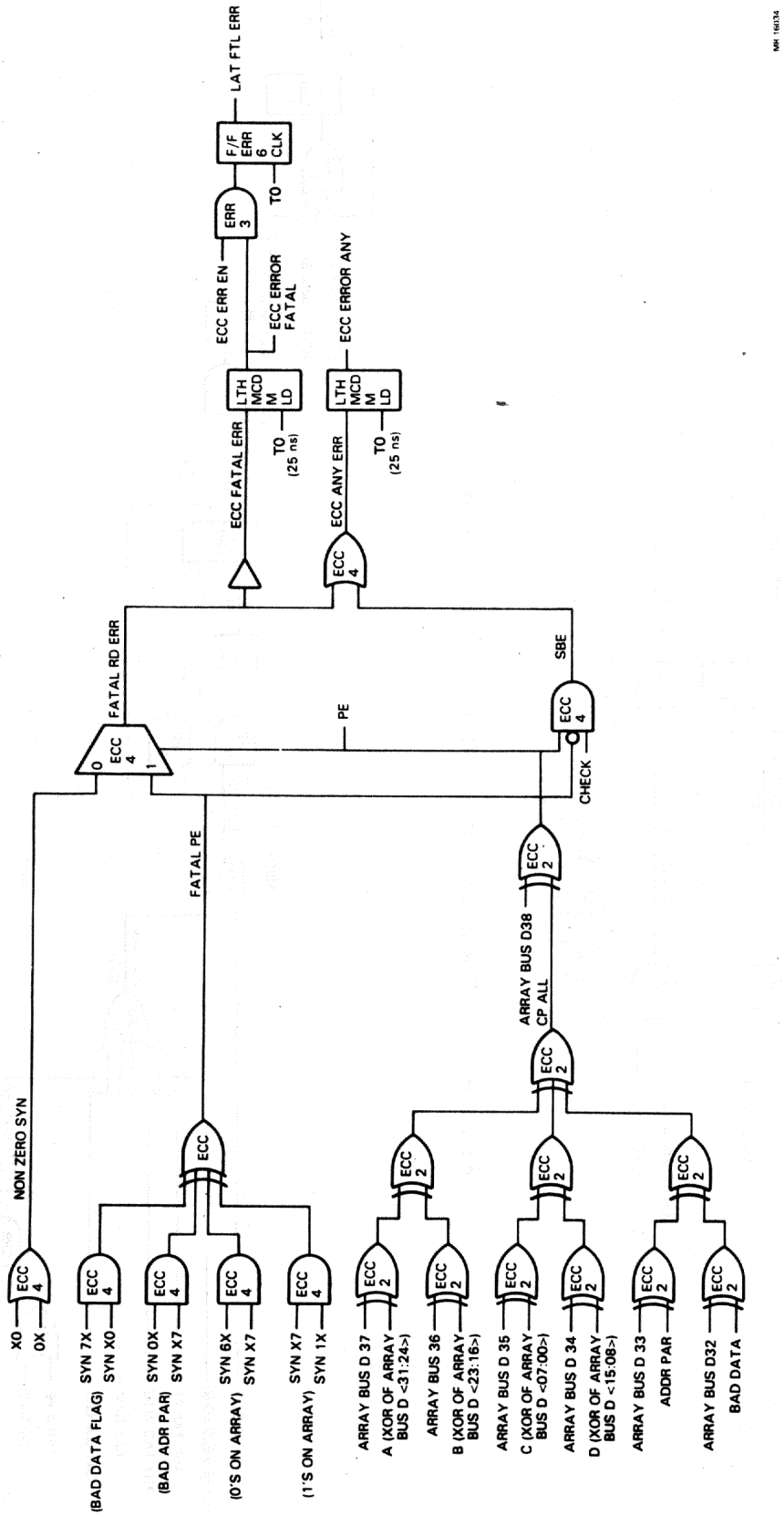


Figure I-6 MBox ECC Error Detection Network

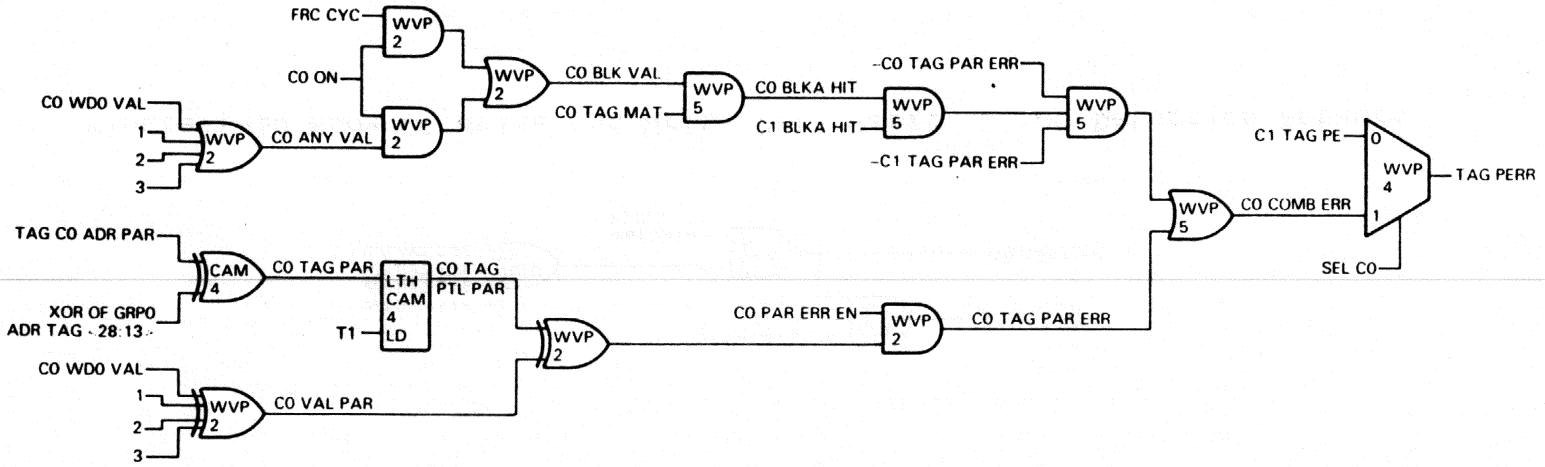


Figure I-7 MBox Cache Tag Parity Error Detection Network

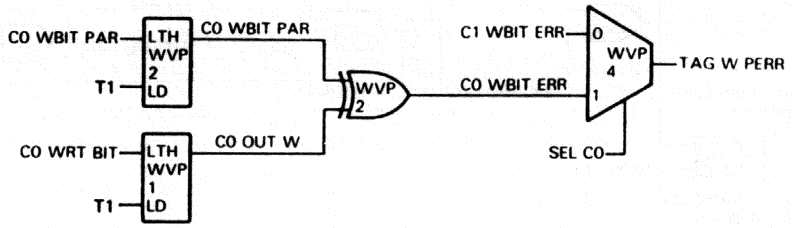


Figure I-8 MBox Cache Tag WBit Parity Error Detection Network

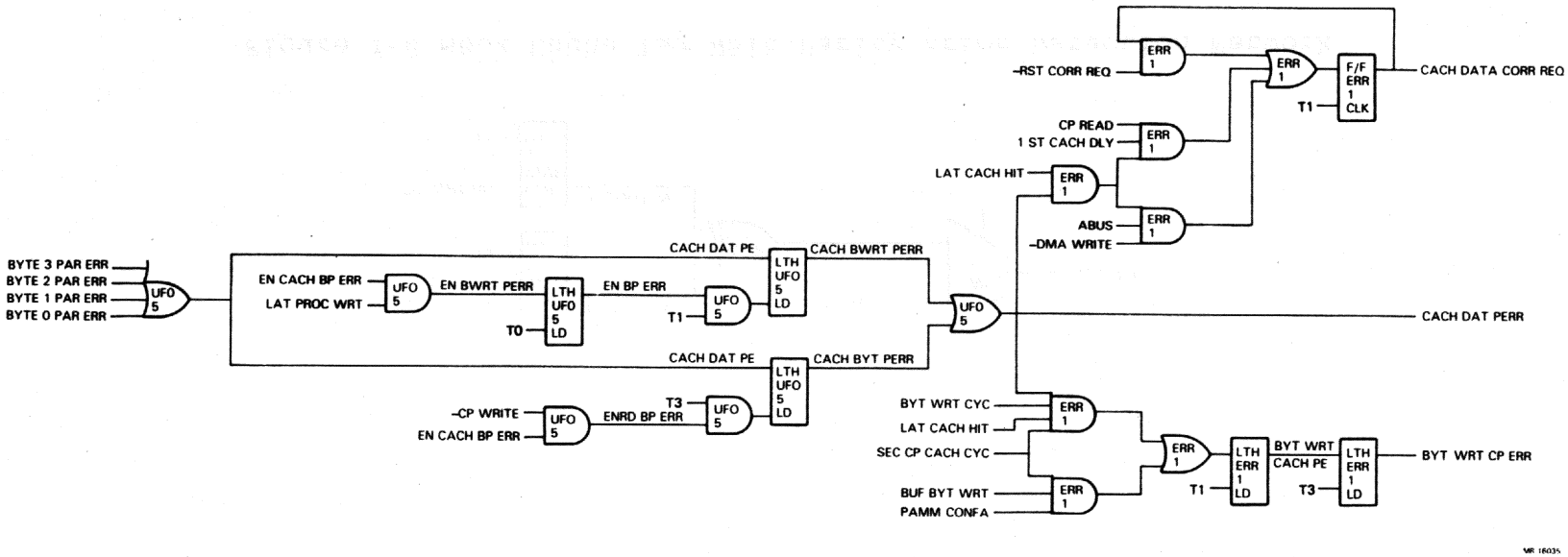


Figure I-9 MBox Cache Data Correction, Cache Data Parity Error and Byte Write CP Error Detection Network

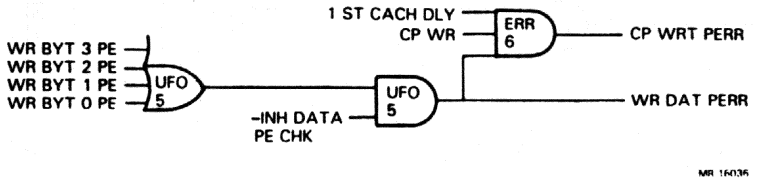


Figure I-10 MBox CP Write and Write Data Parity Error Detection Network

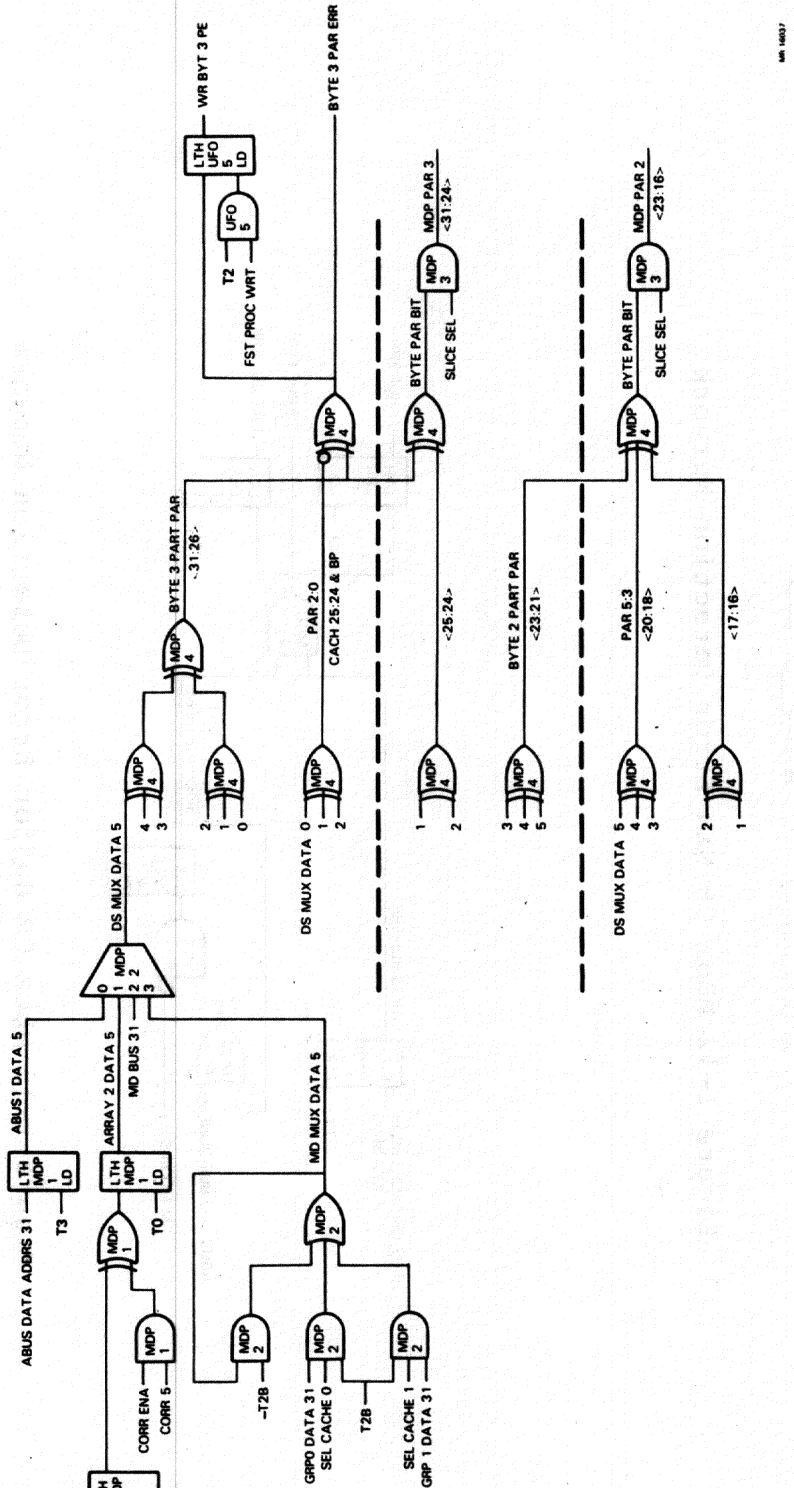


Figure I-11 MBox Write Byte and Byte Parity Error Detection Network

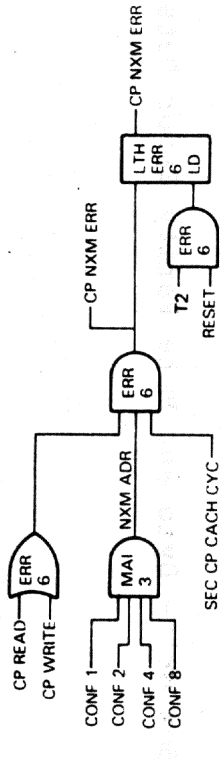


Figure I-12 MBox CP NXM Error Detection Network

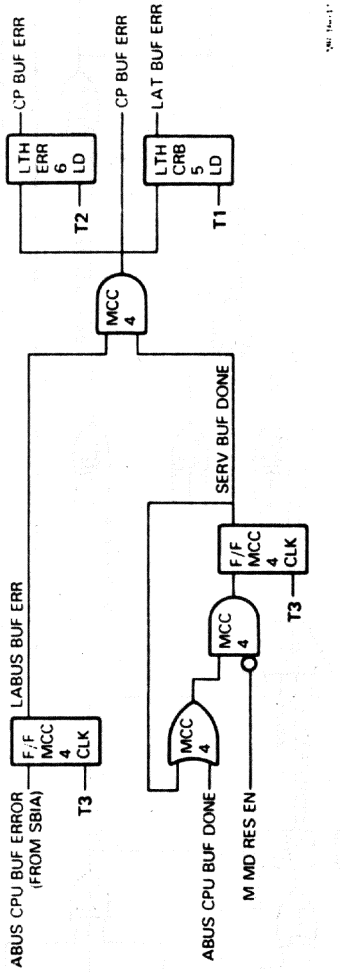


Figure I-13 MBox CP Buffer Error Detection Network

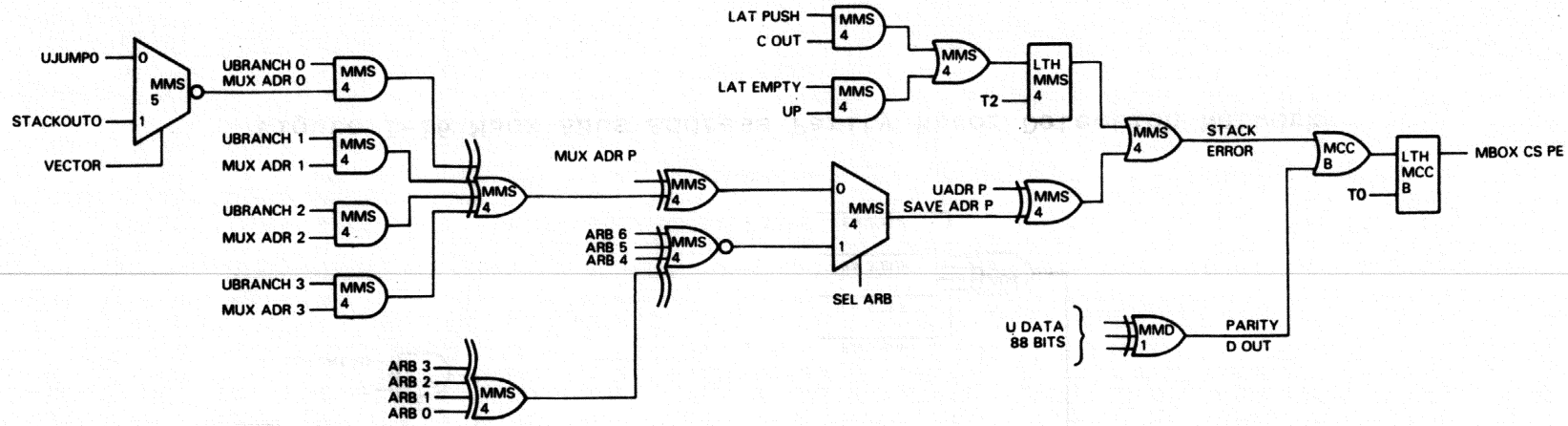


Figure I-14 MBox Control Store Error Parity Detection Network

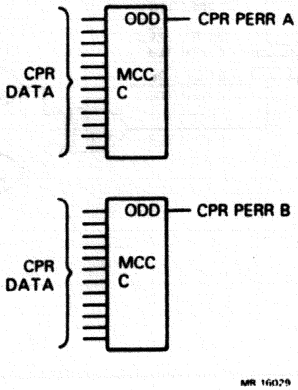


Figure I-15 MBox CPR Parity Error Detection Network

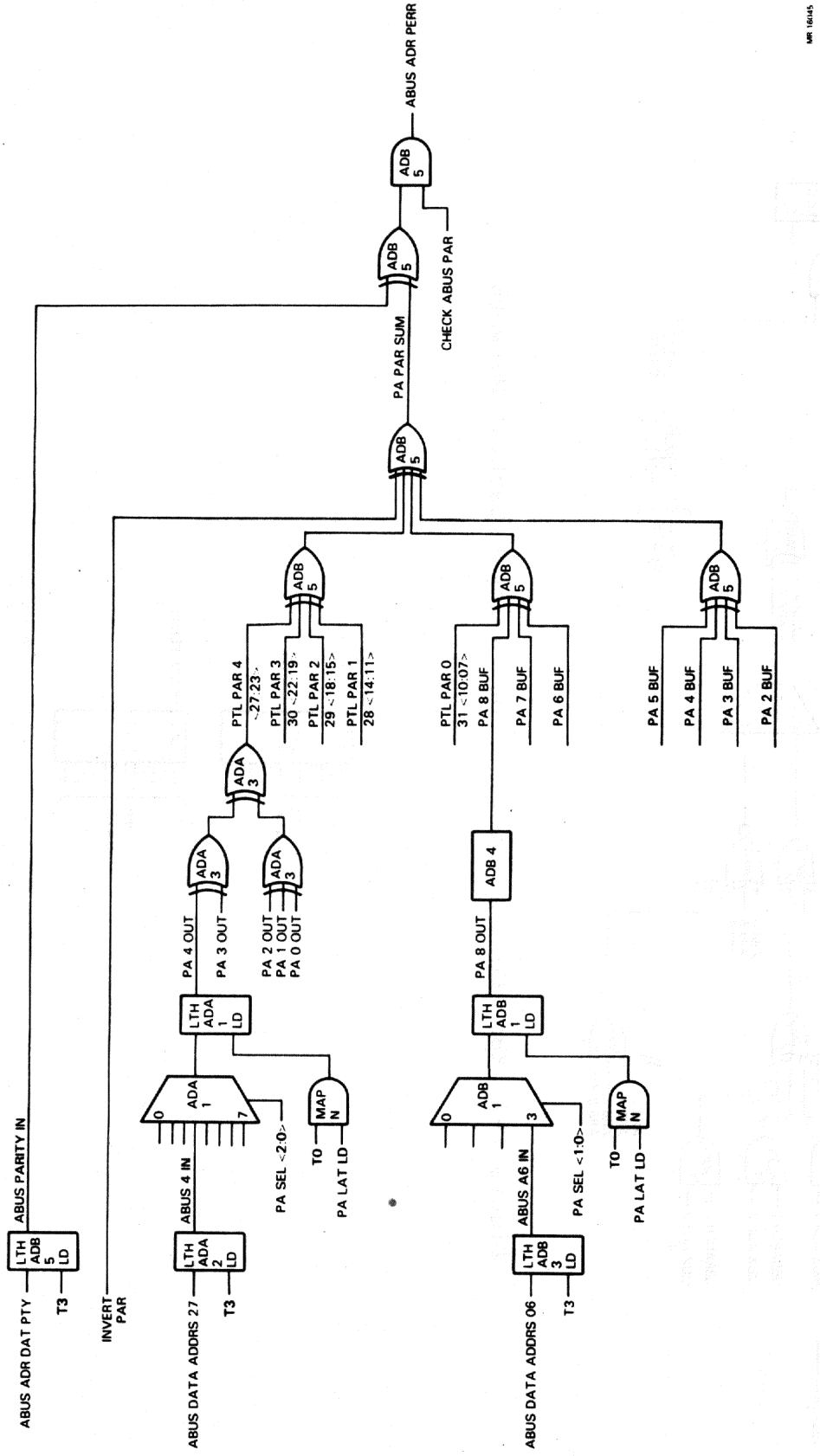


Figure I-16 MBox ABus Address Parity Error Detection Network

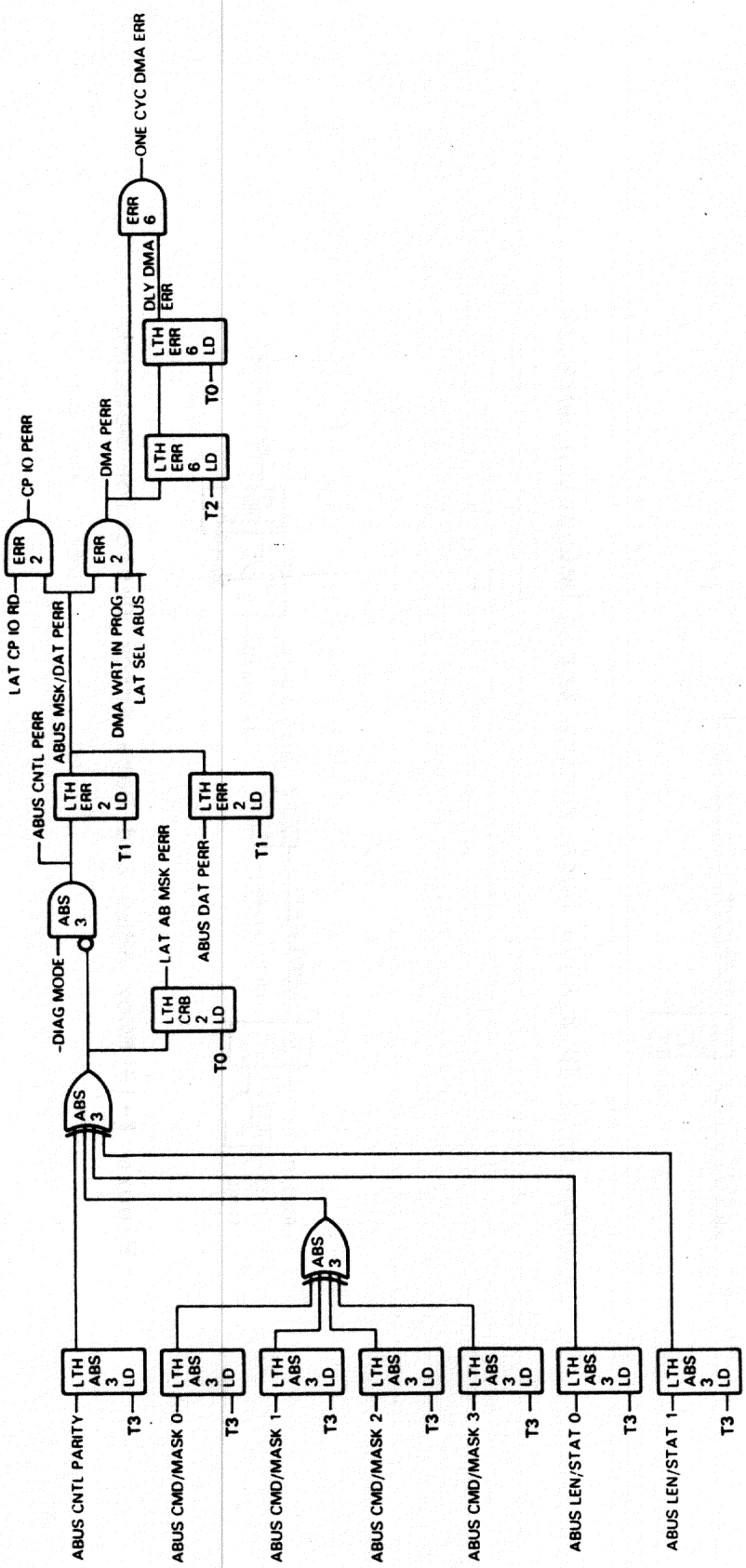


Figure I-17 MBox ABus Control and Mask Parity Error Detection Network

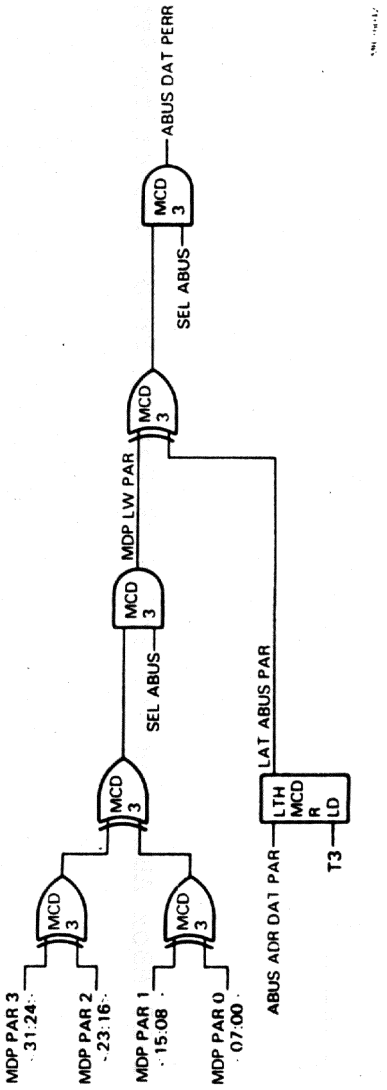


Figure I-18 MBox ABus Data Parity Error Detection Network

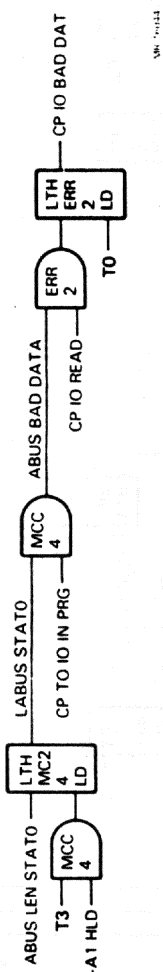


Figure I-19 MBox ABus Bad Data Error Detection Network

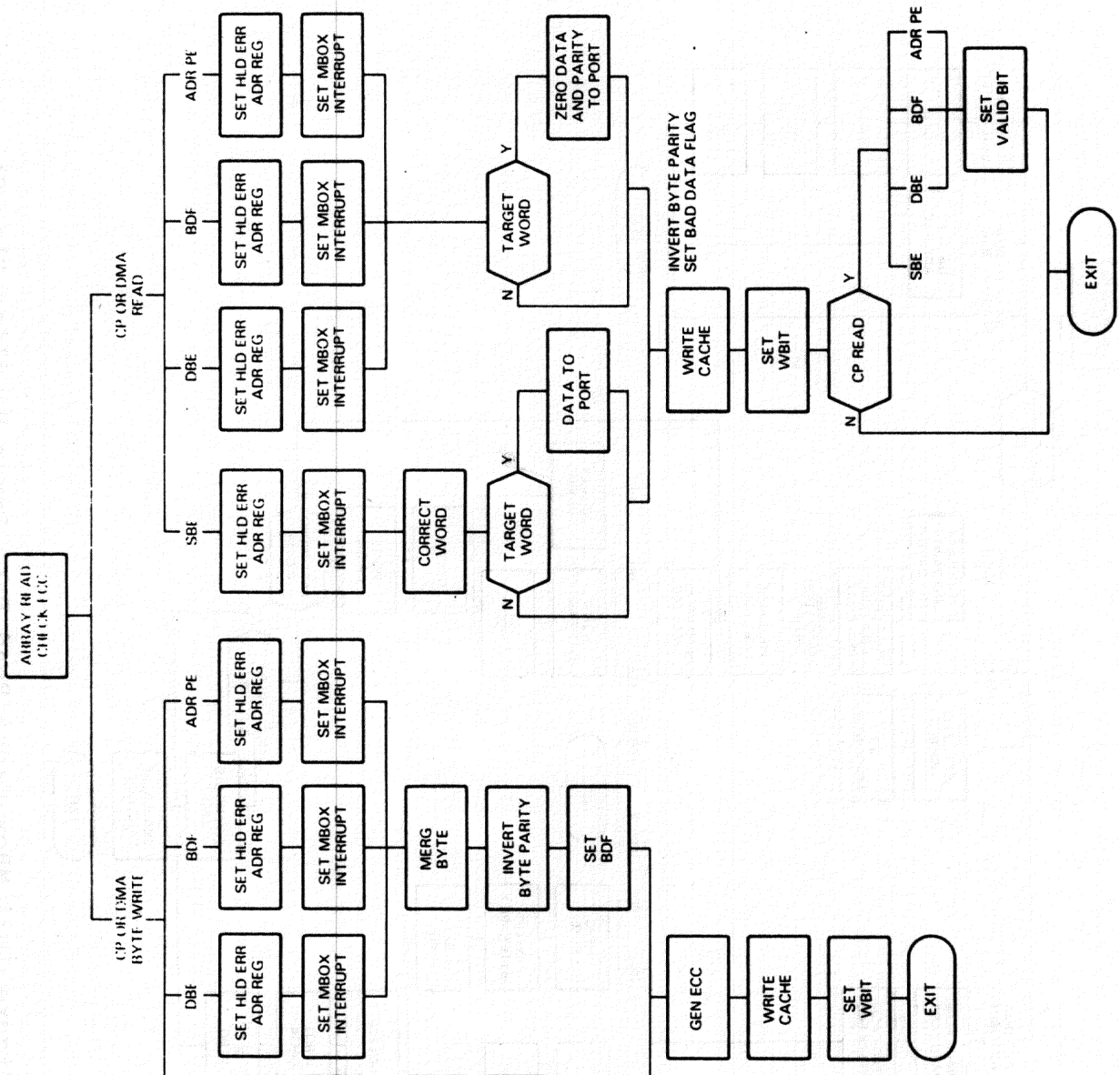


Figure I-20 MBox Array Read Data Error Handling Flow Chart

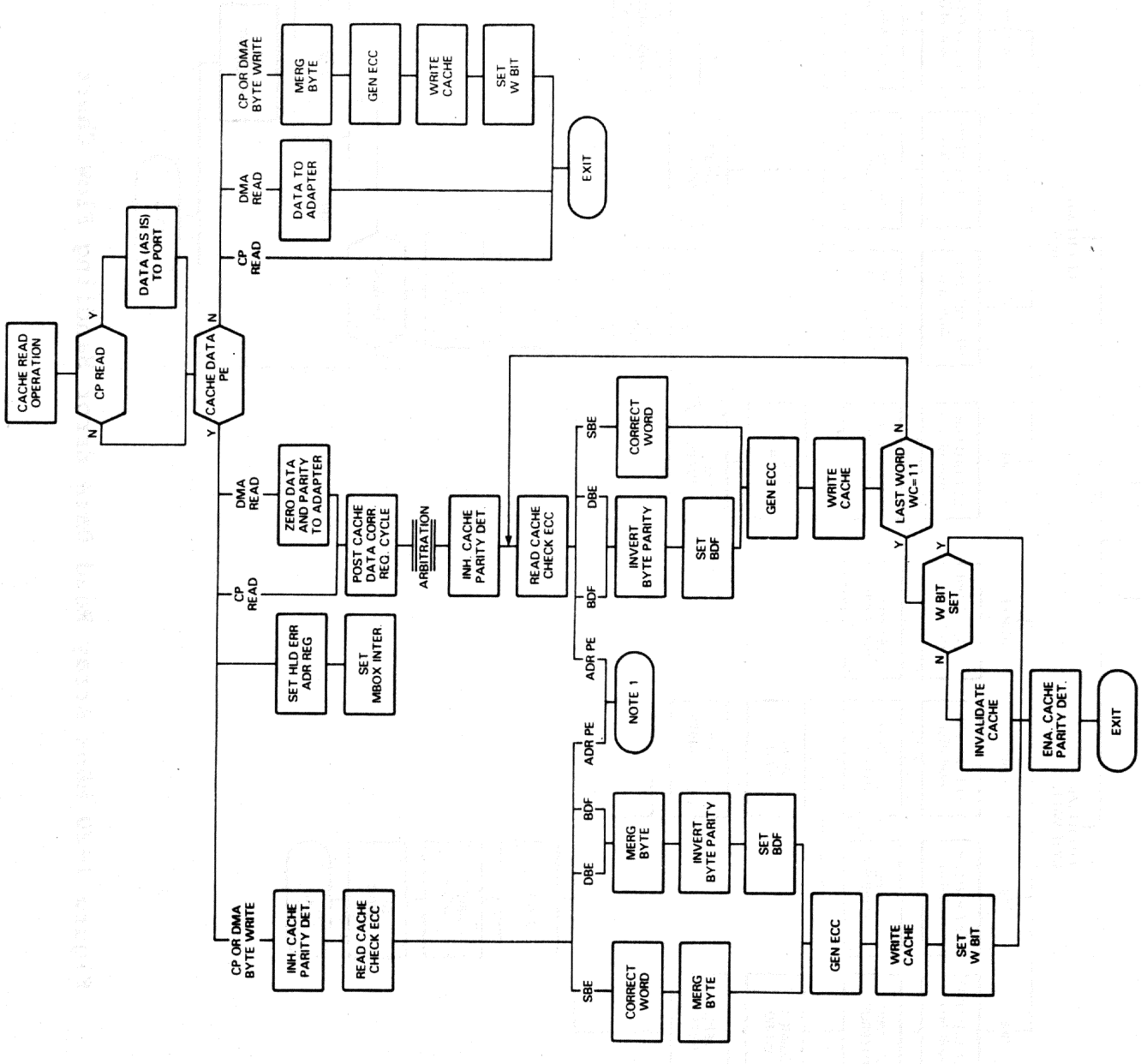


Figure I-21 MBox Cache Read Data Error Handling Flow Chart

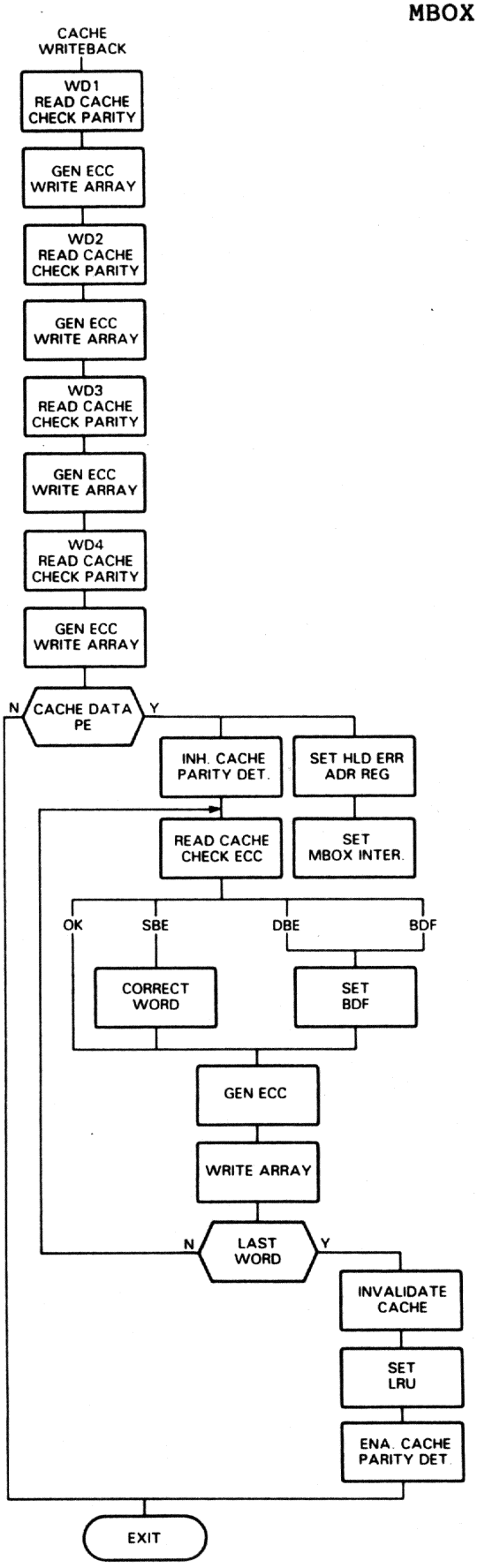


Figure I-22 MBox Cache Writeback Error Handling Flow Chart

APPENDIX J

SBIA ERROR DETECTION NETWORKS

This Appendix contains a set Functional Block Diagrams that describe the SBIA Error Detection Networks. The output of these networks go to the EBox Interrupt Arbitration Network where they are prioritizd and used to generated a System Control Block Vector Address. Other than handling the Interrupt, the EBox does not get involved with I/O Errors. I/O Errors are processed by the VMS Machin Check Handler and the VMS I/O Driver Routines. There is one exception, and that is CP IO BUF ERROR. CP IO BUF ERROR goes to the MBox which will, in turn, generate an MBox Fatal Error.

Table 1 (below) lists both the error conditions and the Figures that describes the networks that are used to detect the error conditions.

Table 1 SBIA Error Detection Networks

Figure	Error Detection Networks
J-1	SBIA DMAI Transaction Buffer Error Lock Circuit
J-2	SBIA DMAA Transaction Buffer Error Lock Circuit
J-3	SBIA DMAB Transaction Buffer Error Lock Circuit
J-4	SBIA DMAC Transaction Buffer Error Lock Circuit
J-5	SBIA Fault and Local Error Detection Network
J-6	SBIA Multiple CPU Error and CPU Buf Error Circuit
J-7	SBIA Control and Address Parity Error Detection Network

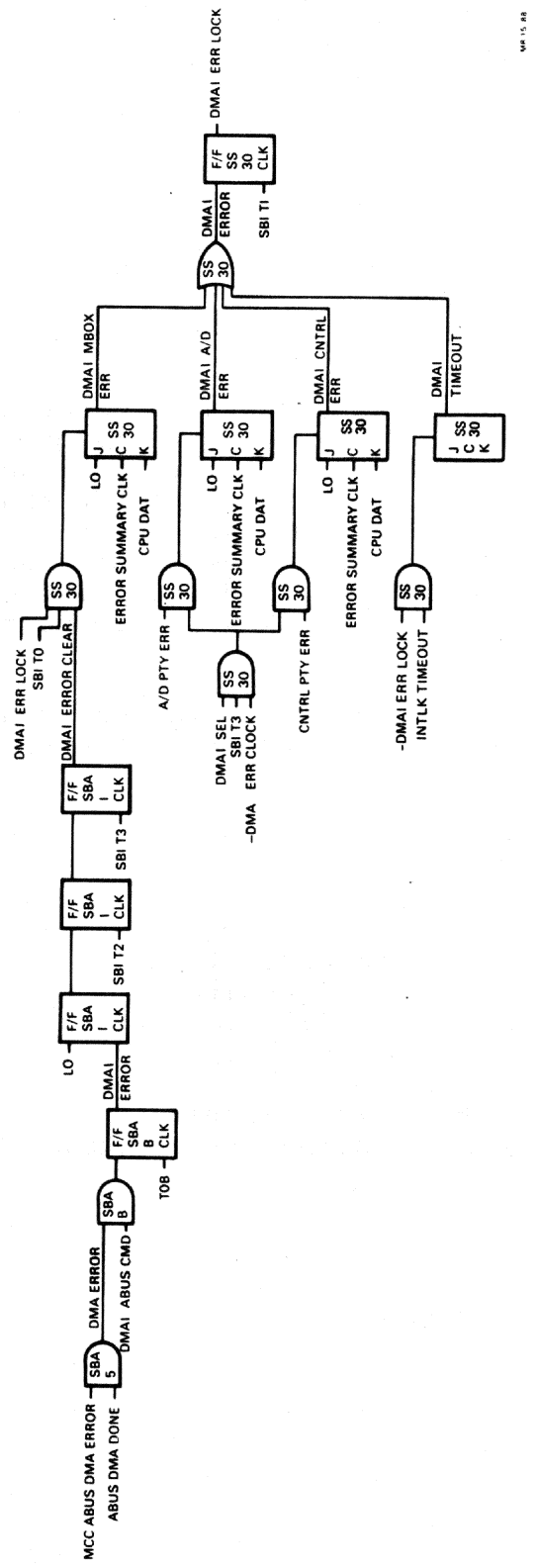


Figure J-1 SBIA DMAI Transaction Buffer Error Lock Circuit

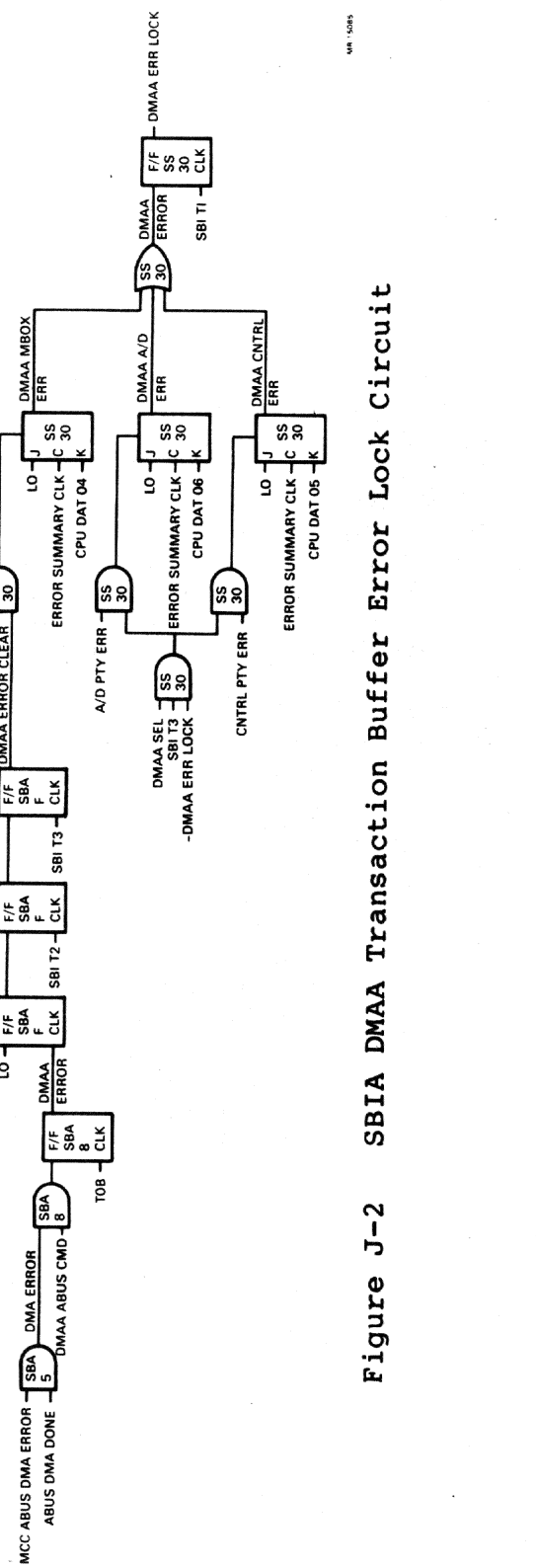


Figure J-2 SBIA DMAA Transaction Buffer Error Lock Circuit

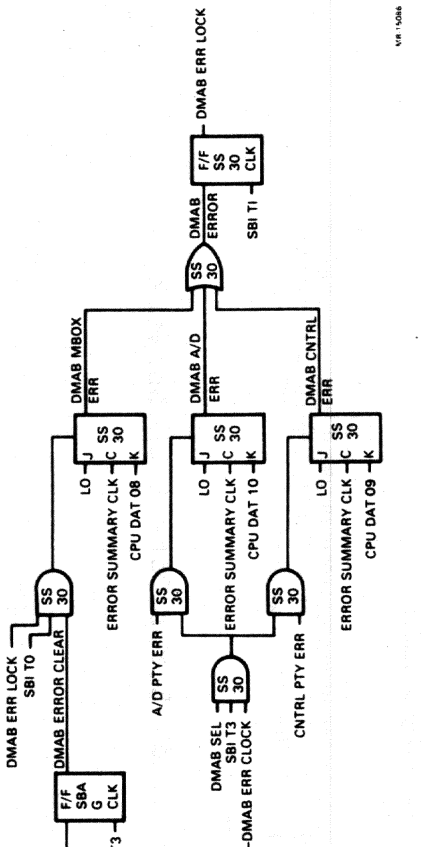


Figure J-3 SBIA DMAB Transaction Buffer Error Lock Circuit

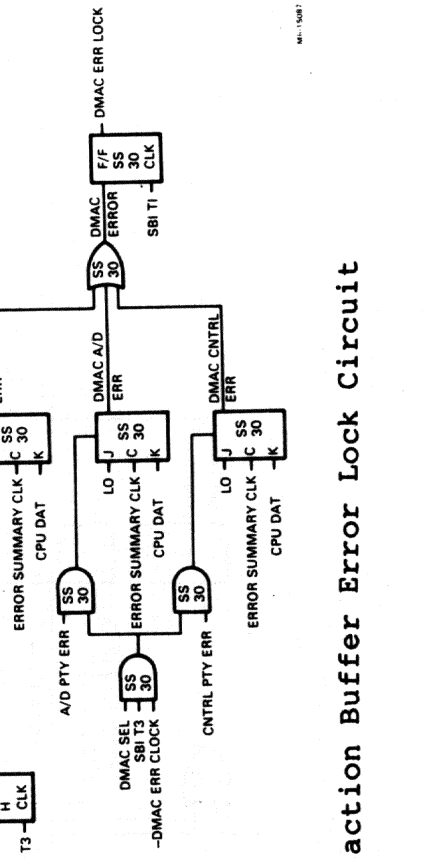


Figure J-4 SBIA DMAC Transaction Buffer Error Lock Circuit

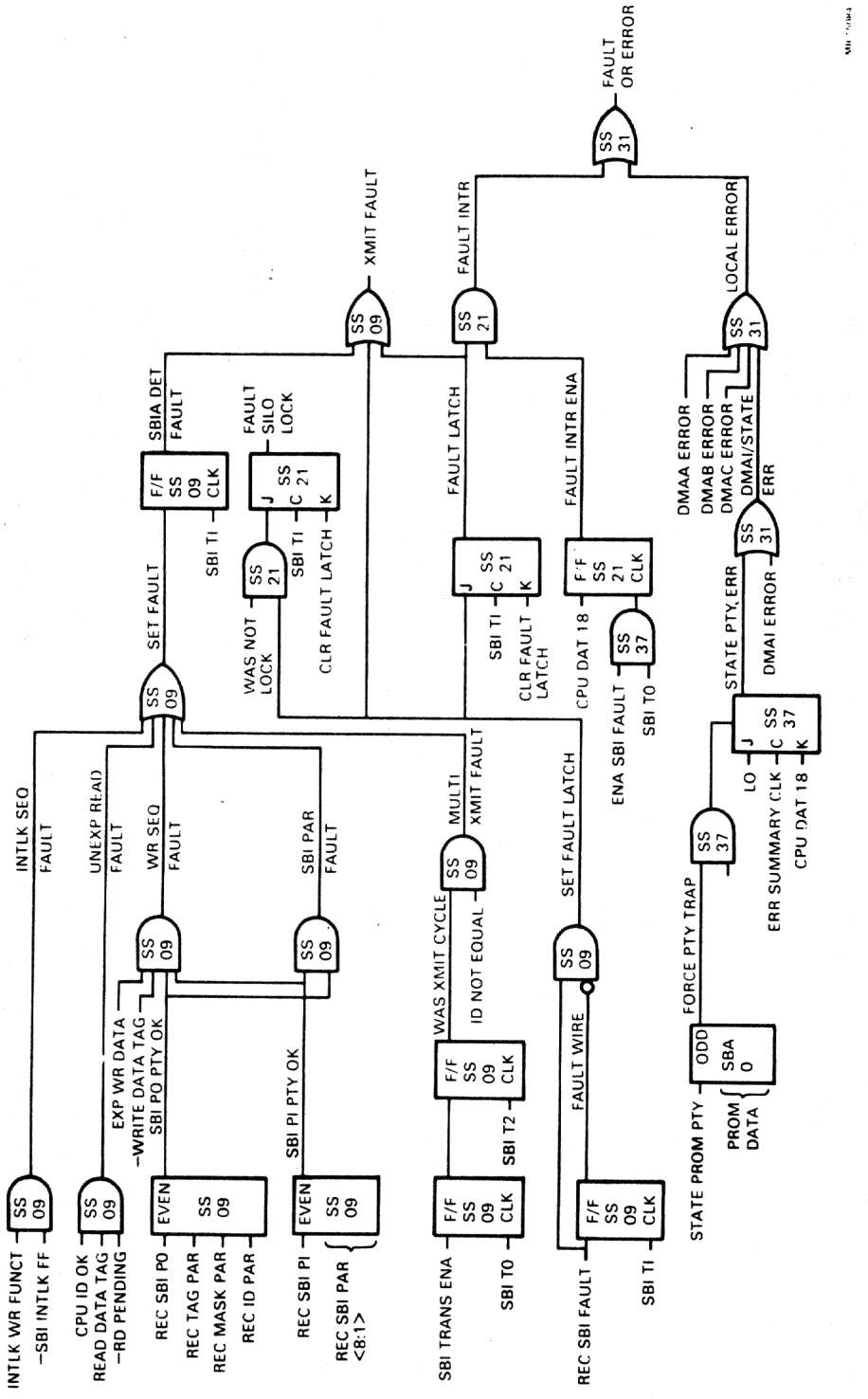


Figure J-5 SBIA Fault and Local Error Detection Network

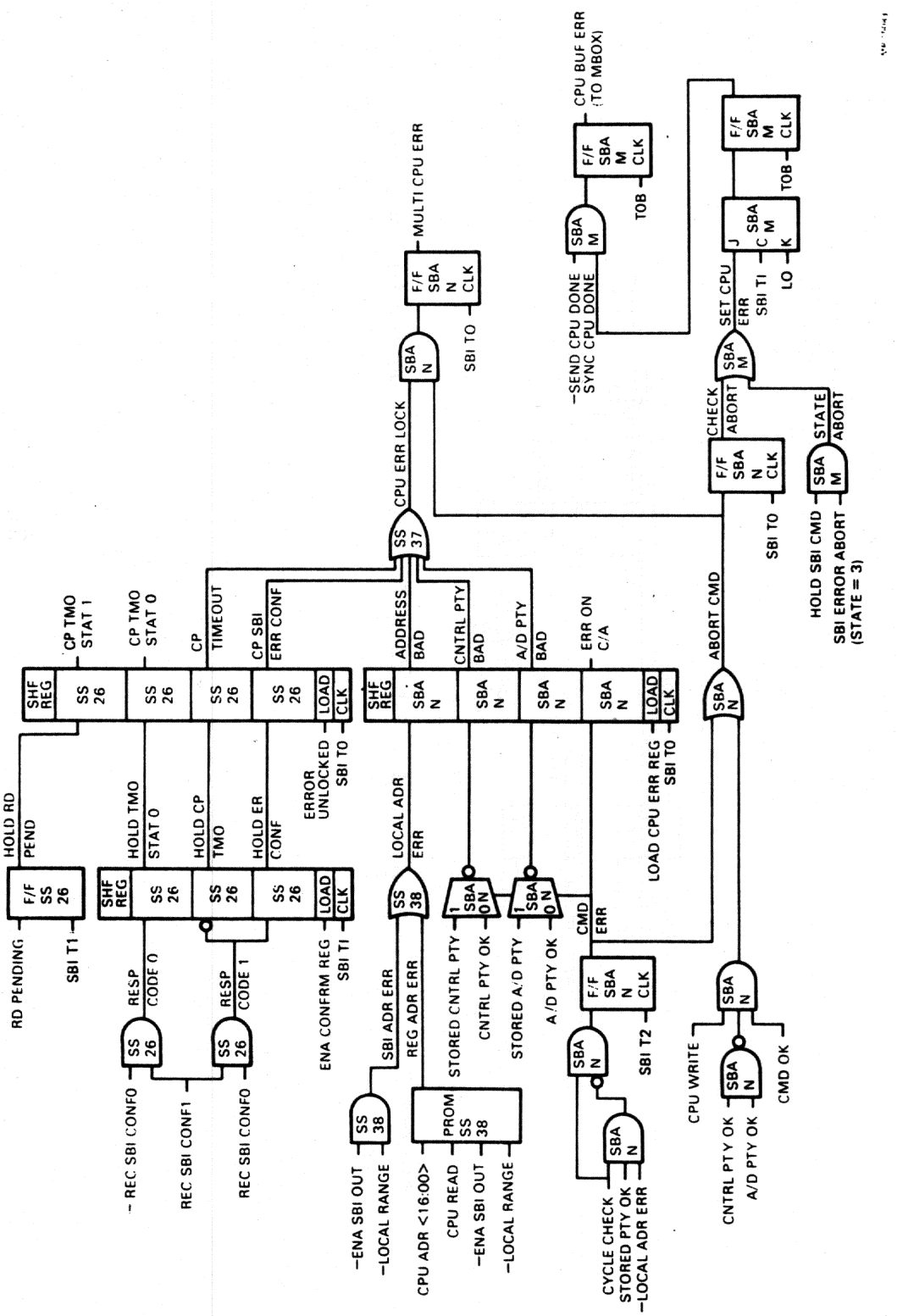


Figure J-6 SBIA Multiple CPU Error and CPU Buf Error Circuit

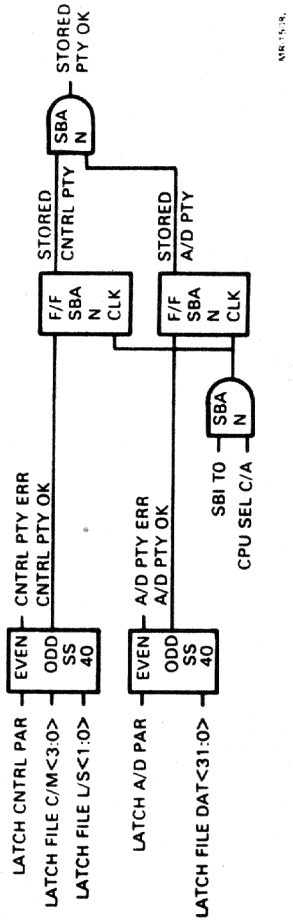
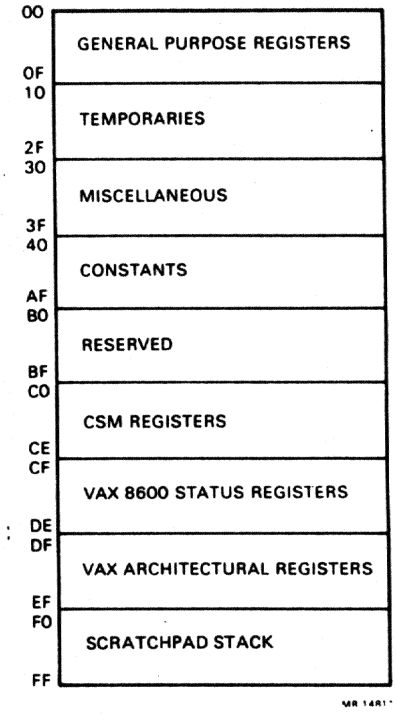


Figure J-7 SBIA Control and Address Parity Error Detection Network

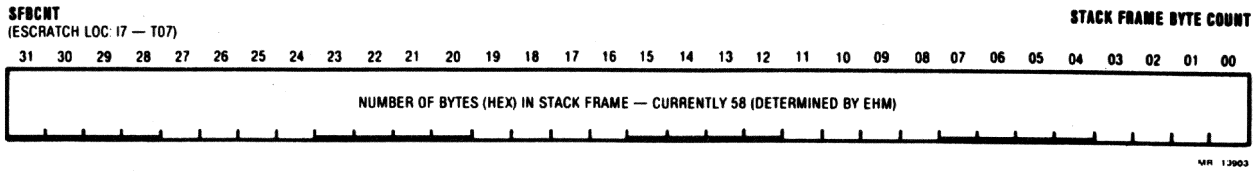
APPENDIX K

MACHINE CHECK STACK FRAME BIT DESCRIPTIONS

This appendix contains bit and field definitions for the registers that make up a machine check stack frame. It also contains bit and field definitions for the error handling status register (EHSR) and the console support microcode status register (CSM.STATUS).



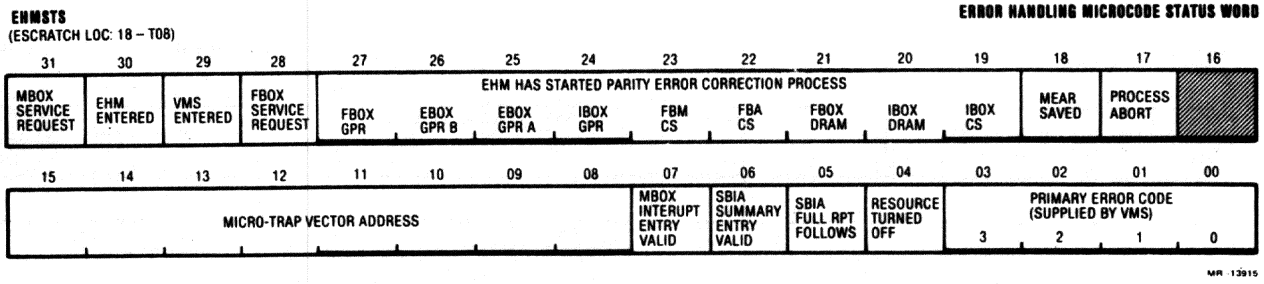
MACHINE CHECK STACK FRAME BIT DESCRIPTIONS
SFBCNT



This longword is built by the Error Handling Microcode (EHM). It contains the number of bytes (in HEX) that the EHM will assemble in the EBox Scratch Pad RAMS (location 18 through 1D). When the EHM is finished building the Machine Check Stack Frame, it calls the Interrupt Exception Microcode. The microcode pushes the contents of the Scratch Pad RAMS onto the Interrupt Stack and calls the VMS Machine Check (MCHK) Handler. The MCHK Handler will use the contents of this longword to process the stack but the SFBCNT will not appear in the Stack Frame Record written to ERRLOG.SYS.

<31:08> RESERVED

<07:00> BYTE COUNT
Indicates the number of bytes, currently 58 HEX-88 Decimal, that the EHM has pushed onto the Interrupt Stack.



This status word contains a modified copy of the Error Handling Status Register (EHSR) which is stored in Scratch Pad location 0DA. The Error Handling Microcode uses EHSR to keep track of status during the error handling process.

In addition, two other Microcode Routines use the EHSR to pass status to the EHM Routine. The Interrupt Handling Microcode Routine uses bit 31 to report MBox error interrupts; and the FBox Problem Handling Routine uses bit 28 to report FBox hardware errors.

The EHM copies the contents of EHSR into EBox Scratch Pad location 18 just before it sets VMS ENTERED and clears EHM ENTERED. The EHM then calls the Interrupt Exception Microcode to push the stack frame onto the interrupt stack and call the Machine Check Handler.

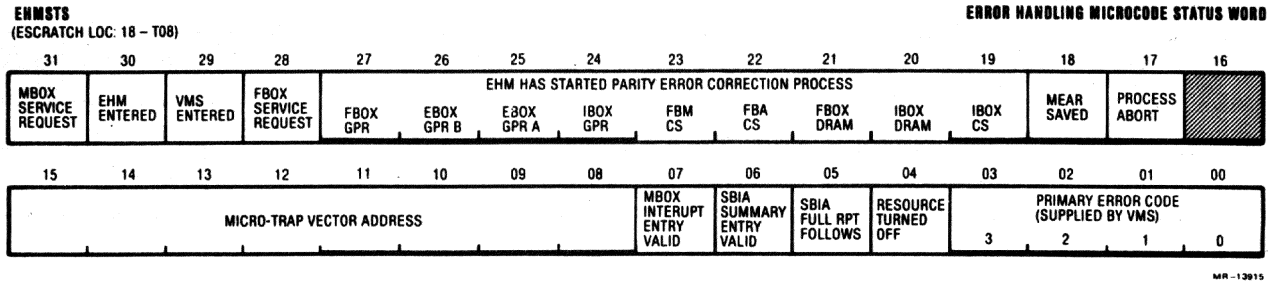
<31> **MBOX SERVICE REQUEST**
If the Interrupt Handling Microcode determines that it was called to handle an MBox Error Interrupt it will set this flag and call the EHM. The EHM will test this flag and, if set, the EHM will process the MBox Error.

<30> **EHM ENTERED**
This flag is used by the EHM to detect a double error trap condition. That is, those cases when a second error trap occurs before the EHM Routine is able to finish processing the first error and pass control to VMS.

This flag is checked each time the EHM routine is entered. If the flag is clear (which is the expected case) then the EHM Routine will set this flag and process the error in the normal manner. If the flag is set, however, indicating that the EHM Routine was in the process of handling an error when it was called to handle a second error, then one of two things will happen:

1. If the second error was detected by either the EBox or the MBox Fatal Error detection circuitry, then the EHM Routine will loop at UPC 21. This in turn will cause a Keep Alive Fail condition and the Console will Print the following message and Snap Shot the state of the system.

   "Attempting to save machine state due to" "MACHINE DOUBLE ERROR"



2. If the second error was detected by the IBox then the EHM will put a code of 5 in CSM.STATUS (Scratch Pad location: C0) and call the CSM.ENTRY.DE Routine. This will result in a Keep Alive Fail Condition and the Console will print the following message and Snap Shot the state of the system.

   "Attempting to save machine state due to" "CPU ERROR HALT"

NOTE

Because the state of EHSR is saved in EScratch 18 before the EHM Routine clears EHM ENTERED, this flag will always be set in the copy of the Stack Frame saved by the VMS MCHK Handler.

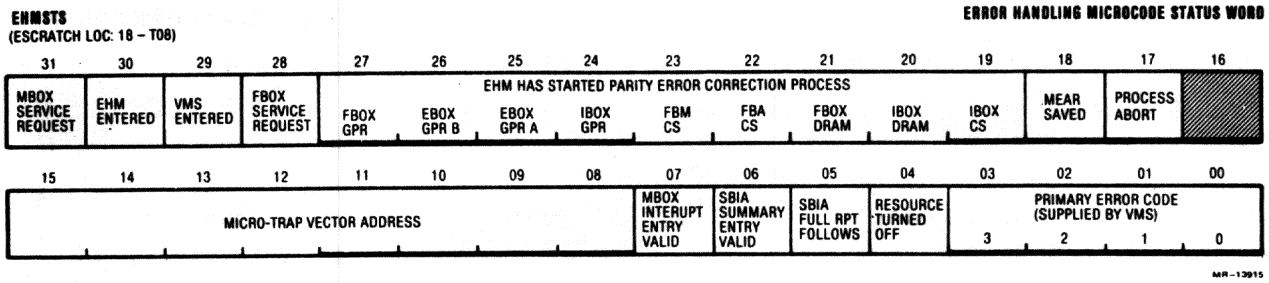
<29> VMS ENTERED

This flag is similar to the EHM ENTERED flag. It is used to detect the case where the VMS Machine Check Handler is in the process of handling an error when a second error is detected.

The EHM Routine sets this flag just before it calls the Machine Check Handler. The Machine Check Handler processes the errors and clears this flag just before it executes an REI to continue the operation (or a BUGCHECK to halt the operation).

If a second error trap occurs while the Machine Check Handler is still processing the first error, then the EHM Routine will process the error in the normal manner. That is, build a Stack Frame, clear the error condition, roll back the PCs and determine if it should call VMS.

However, since the VMS ENTERED flag is set (indicating that VMS was processing an error when a second error was detected), the EHM will not call VMS. Instead it will put a code of 5 in CSM.STATUS (EBox Scratch Pad Location: C0) and call the CSM.ENTRY.DE Routine. This in turn will result in a Keep Alive Fail Condition and the Console will print the following message and Snap Shot the state of the system.



"Attempting to save machine state due to" "CPU ERROR HALT"

<28> **FBOX SERVICE REQUEST**
The micro-routine that handles FBox Problems will set this flag if it determines that the problem was caused by an FBox hardware error. The routine will then call the EHM Routine to process the error. The EHM will test this flag, to determine if it was called to handle an FBox error.

<27> **FIX FBOX GENERAL PURPOSE REGISTER PARITY ERROR**
Set by the EHM when it starts to correct an FBox GPR parity error.

<26> **FIX EBOX GENERAL PURPOSE REGISTER B PARITY ERROR**
Set by the EHM when it starts to correct an EBox GPR B parity error.

<25> **FIX EBOX GENERAL PURPOSE REGISTER A PARITY ERROR**
Set by the EHM when it starts to correct an EBox GPR A parity error.

<24> **FIX IBOX GENERAL PURPOSE REGISTER PARITY ERROR**
Set by the EHM when it starts to correct an IBox GPR parity error.

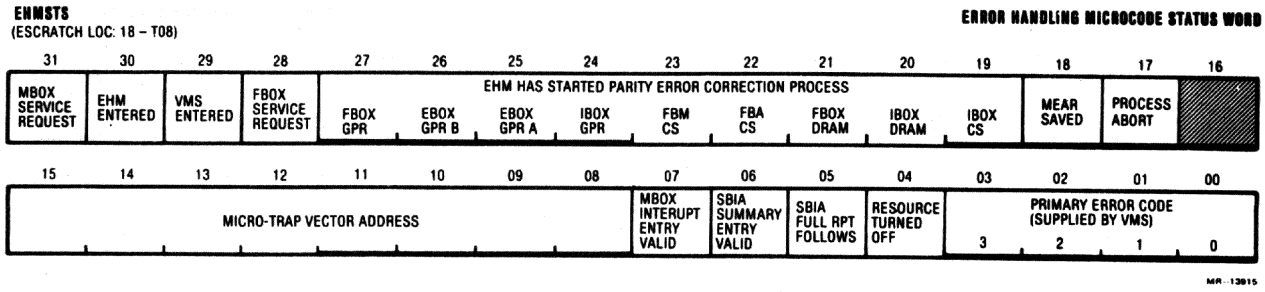
<23> **FIX FBM CONTROL STORE PARITY ERROR**
Set by the EHM when it starts to correct an FBM Control Store parity error.

<22> **FIX FBA CONTROL STORE PARITY ERROR**
Set by the EHM when it starts to correct an FBA Control Store parity error.

<21> **FIX FBOX DRAM PARITY ERROR**
Set by the EHM when it starts to correct an FBox Dispatch RAM parity error.

<20> **FIX IBOX DRAM PARITY ERROR**
Set by the EHM when it starts to correct an IBox Dispatch RAM parity error.

<19> **FIX IBOX CONTROL STORE PARITY ERROR**
Set by the EHM when it starts to correct an IBox Control Store parity error.



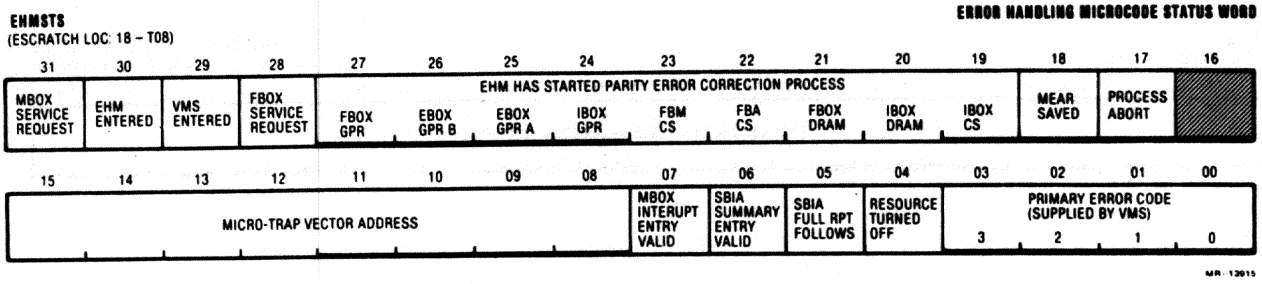
<18> **MEAR SAVED**
Indicates that the MBox Error Address Register (MEAR) was saved (in EScratch Location: DB) as a result of the last MBox Error Register Full trap.

<17> **PROCESS ABORT**
The EBox microcode detected a condition which prevents a retry of the faulted instruction. See EBCS <19:16> for the reason code.

<16> **RESERVED**

<15:08> **MICRO TRAP VECTOR ADDRESS**
Contains the vector address through which the EHM was entered.

VECTOR	REASON
2	FBox Error (Called by FBox Interrupt Handler)
4	EHM Detected a Process Abort condition during a MBox Error Register Full Micro-trap.
6	MBox Error (Called by MBox Interrupt Handler)
8	EBox Error
8	MBox Fatal Error.
8	TB PE Error (EBox Port Request only)
10	IBox Op-Port-Write and IBox Error
10	IBox Op-Port-Write and TB Parity Error
10	EBox IMD Read and IBox Error
10	EBox IMD Read and TB Parity Error
18	EBox Fork and IBox Error
18	EBox Fork and TB Parity Error
18	EBox ID Read and IBox Error
18	EBox String Read and IBox Error
18	EBox String Read and TB Parity Error
1E	IBox Sync Failure
1F	Rlog Unwind Failure



<07:00> **REASON CODE**
This field is supplied by the VMS Machine Check Handler. Therefore, this field will be valid only after the Machine Check has been processed by VMS. This field will not be valid for Stack Frames extracted directly from the EBox Scratch Pad RAMs or the Interrupt Stack. See <03:00> for the actual Reason Code.

<07> **MBOX INTERRUPT ENTRY VALID**
Indicates that the Stack Frame was generated as a result of an MBOX 1D interrupt.

<06> **SBI SUMMARY ENTRY VALID**
Indicates that a copy of the SBIA Error Summary Register has been appended to the end of the stack frame.

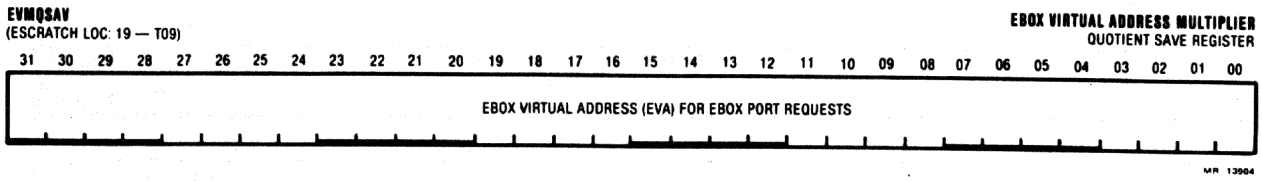
<05> **SBIA FULL REPORT FOLLOWS**
Indicates that a full SBIA Error Log entry follows this entry in the System Event File (ERRLOG.SYS).

<04> **RESOURCE TURNED OFF**
Indicates that VMS disabled either the FBox or one of the Caches.

<03:00> **PRIMARY ERROR CODE**
This field indicates which Box detected the error.

Code	Box Detecting Error
001	FBox
010	EBox
011	IBox
100	MBox (Fatal Error)

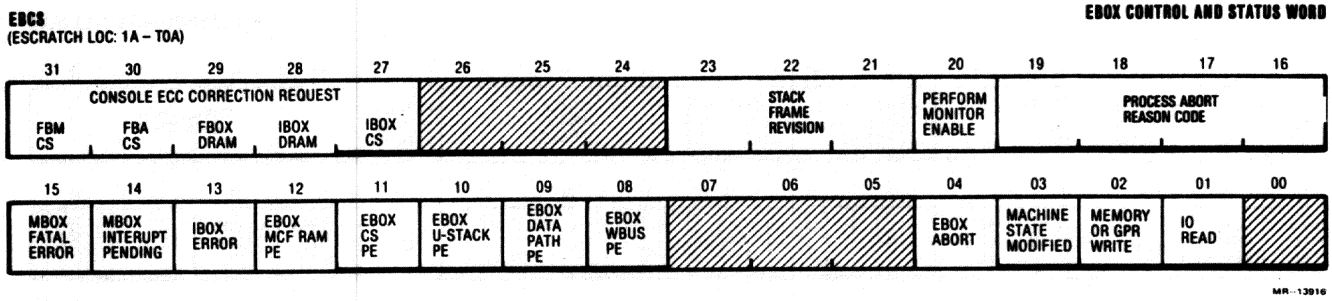
MACHINE CHECK STACK FRAME BIT DESCRIPTIONS
EVMQSAV



EVMQSAV (EBox Virtual Memory Address/Multiplier Quotient Save Register).

During EBox port requests this register contains the virtual address that was acknowledged by the MBox (PA ACK). During normal operation, this register is used to temporarily store partial EBox results.

<31:00> May contain an EBox Virtual Address or a partial calculated EBox result depending on the operation of the EBox.



This register is made up from the EBCS register (see EBD2, EBD3 and EBE5) and other EBox status bits (see EBEB and EBEC).

<31:27> CONSOLE ECC CORRECTION REQUESTS
The bits in this field are set by the EHM in order to interrupt the Console for Control Store or Dispatch RAM correction.

When set, these bits will cause an immediate Console interrupt. The EHM will loop waiting for the Console to set "DONE" in RBUFC. When correction is complete, control is returned to the EBox by setting "DONE". The EHM will then clear the correction request EBCS <31:27> and thus release the Console interrupt.

<31> FBM CONTROL STORE CORRECTION REQUEST
EBE5 FBOX FBM CS PE
The Console will attempt to correct the parity error and then return control to the EHM by setting the "DONE" bit <07> in RBUFC.

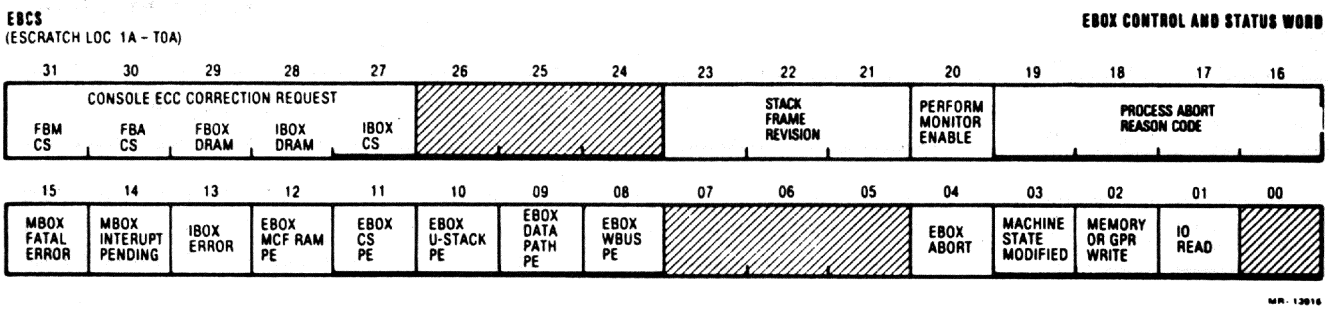
<30> FBA CS CORRECTION REQUEST
EBE5 FBOX FBA CS PE
The Console will attempt to correct the parity error and then return control to the EHM by setting the "DONE" bit <07> in RBUFC.

<29> FBOX DRAM CORRECTION REQUEST
EBE5 FBOX DRAM PE
The Console will reload the FBox Dispatch RAM and then return control to the EHM by setting the "DONE" bit <07> in RBUFC.

<28> IBOX DRAM CORRECTION REQUEST
EBE5 IBOX DRAM PE
The Console will attempt to correct the parity error and then return control to the EHM by setting the "DONE" bit <07> in RBUFC.

<27> IBOX CS CORRECTION REQUEST
EBE5 IBOX CS PE
The Console will attempt to correct the parity error and then return control to the EHM by setting the "DONE" bit <07> in RBUFC.

<26:24> RESERVED



<23:21> **STACK FRAME REVISION**
The stack frame revision is written by the error handling microcode to indicate how many times the stack frame has been revised. The previous stack frame was revision 0, and the current stack frame is revision 1.

<20> **PERFORM MONITOR ENABLE**
EBE5 PERFORMANCE MONITOR ENABLE
This is an architecturally defined bit which is controlled by System software. It is wired to a CPU backplane pin (Slot 09 Pin A07) and allows system performance to be monitored externally. See the PMR description for further information.

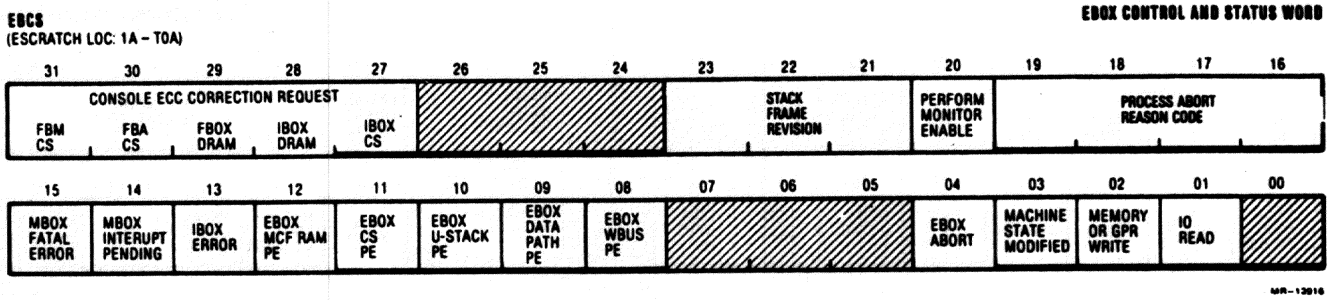
<19:16> **PROCESS ABORT REASON CODE**
If EHMSTS <17>, PROCESS ABORT, is set, EBCS <19:16> is the reason why the faulted instruction cannot be retried.

0001 Unrecoverable GPR parity error
0010 EBox WBus parity error
0011 All IBox PC's are invalid
0100 EBox failed to detect OPBus byte parity error
0101 CPR parity error that did not result in an MBox fatal error
0110 RLog parity error

<15> **MBOX FATAL ERROR**
ERR6 MBOX FATAL ERROR
Set via EBox Port Status Line 0. Indicates that the MBox detected one of the following Fatal Error conditions:

ERR1 WR DAT PE & WRITE REG	ERR2 CP IO PE
ERR2 TAG PE & WBIT	ERR6 CP BUFF ERR
ERR2 DMA PE	ERR6 CP NXM ERR
MCCJ CPR FTL ERR	

<14> **MBOX INTERUPT PENDING**
EBC2 MBOX INTR LVL3
The Mbox generates an interrupt request when it detects an error of any kind (excluding TB parity errors). This bit is set by the EBox on the third occurrence of T3 after it receives MBox Interrupt and is usually handled at IRD Time.



<13> **IBOX ERROR**
EBEA IBOX ERR LTH
Set by the EBox when the IBox reports one of the following error conditions.

ICBC IDRAM PE IDP6 IBMUX PE ICA7 ICS PE
EBD7 RSV MODE IDP6 IAMUX PE ICBC IBUF PE
ICB6 RLOG PE

When written by EBox microcode, it will clear IBox errors and <13>.

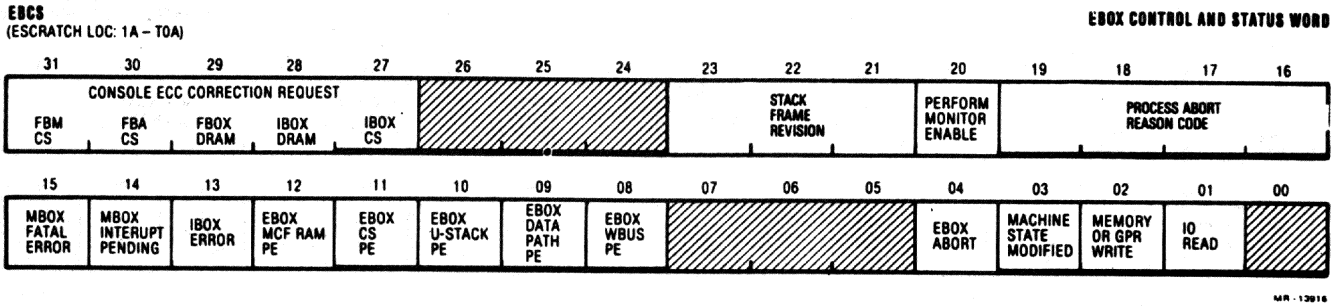
<12> **EBOX MEMORY CONTROL FIELD RAM PARITY ERROR**
EBD2 EMCR PE FLAG
Set when the EBox detected a parity error in the Memory Control Field (MCF) RAM.

<11> **EBOX CONTROL STORE PARITY ERROR**
EBD2 ECS PE FLAG
Set when the EBox detected a Control Store Parity Error. Setting this bit results in a immediate trap to the Console for ECC Correction. When the Console finishes correcting the error it will force the EBox to trap to EHM (Vector 8).

<10> **EBOX MICRO-STACK PARITY ERROR**
EBD2 USTK PE FLAG
Set when the Ebox detected a parity error when it popped the last entry off the micro-stack.

<09> **EBOX DATA PATH PARITY ERROR**
EBD2 EDP PE FLAG
Set when the EBox detected either an Operand parity error or a Result parity error.

MACHINE CHECK STACK FRAME BIT DESCRIPTIONS
EBCS



<08> EBOX WBUS PARITY ERROR
EBD2 WBUS PE FLAG
Set when the EBox detected a WBus Parity Error while it was driving the bus.

NOTE

1. Writing 1 to this bit clears bits <15>, <12:09> and <4>.

2. This bit has a different use in diagnostic mode.

<07:05> RESERVED

<04:01> VMS ABORT FLAGS
VMS uses these flags, in part, to determine whether or not the error is recoverable.

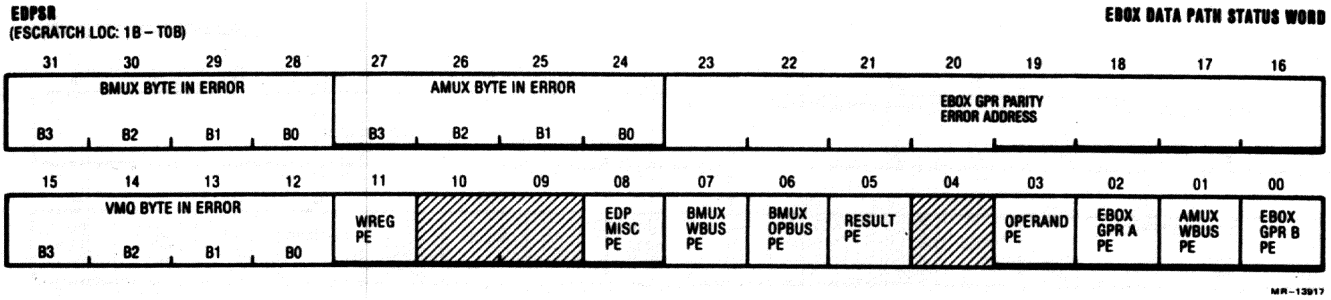
<04> EBOX ABORT
EBD3 EB ABORT FLAG
Set when the EBox detected an MCF RAM Parity Error or a Result Parity Error. Both cases are non-recoverable.

<03> MACHINE STATE MODIFIED
EBD3 STA MOD FLAG
Set by Microcode via the UMISC Field. Indicates that the state of the machine has been modified such that, should an error occur while this bit is set the operation currently executing in the EBox cannot be retried.

<02> MEMORY OR GPR WRITE
EBD3 MEM WRT FLAG
Set when a Memory or GPR write operation was in progress when the error occurred.

<01> IO READ
EBD3 IO RD FLAG
Set when the error involved an I/O register read. Since some I/O registers automatically clear after being read the contents of the I/O register may have been lost.

<00> RESERVED



This register is made up from the nibble registers in the PDP MCA.

<31:28> BMUX BYTE IN ERROR
PDP4 BMX ERR BYTE <3:0> (EDPH)
This field is only valid when either bit <07> or bit <00> is set. This field indicates the byte(s) that were associated with the BMux Parity Error.

<27:24> AMUX BYTE IN ERROR
PDP3 AMX ERR BYTE <3:0> (EDPH)
This field is only valid when either bit <01> is set, or when bit <02> is set, or when bit <03> is set and bits <00:02> and <06:07> are reset. This field indicates the byte(s) that were associated with the AMUX Parity Error.

<23:16> EBOX GPR PARITY ERROR ADDRESS
If EHMSTS <26> (EBox GPR B PE) or <25> (EBox GPR A PE) is set, these bits indicate the failing GPR address.

<15:12> VMQ BYTE IN ERROR
PDP5 VMQ ERR BYTE <3:0> (EDPH)
This field is only valid when bit <05> is set and bit <08> is reset. It indicates the byte(s) that were associated with the Result parity error.

<11> WREGISTER PARITY ERROR
PDP6 WREG ERR (EDPH)
Indicates that the parity at the input to the WReg Mux did not match the parity generated at the output of the WReg. The inputs to the WReg Mux are:

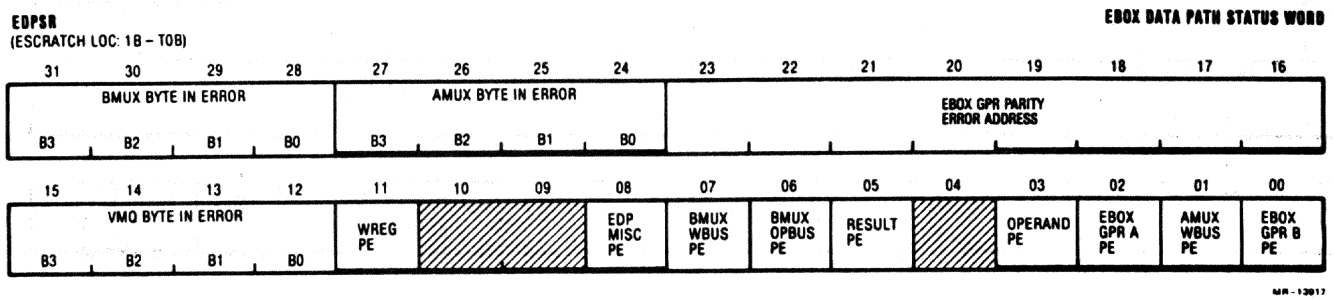
- o The AMux and BMux for WReg Shift Operations.
- o The BMux for WReg Format Operations.
- o The ALU for post ALU Wreg Shift Operations.

NOTE

This error will also cause a RESULT PE.

<10:09> RESERVED

MACHINE CHECK STACK FRAME BIT DESCRIPTIONS
EDPSR



<08> EDP MISCELLANEOUS PARITY ERROR
PDP8 EDP MISC (EDPH)
Indicates that the ALU was the input to the VMQMUX when a parity error was detected at the VMQMUX output. When this bit is set the contents of bits <12:15> are not valid. This error will also cause a RESULT PE.

<07> BMUX WBUS PARITY ERROR
PDP8 B WBUS (EDPH)
Indicates that the WBus was the input to the BMux when a parity error was detected at the BMux output. Bits <28:31> indicate the byte(s) in error.

<06> BMUX OPBUS PARITY ERROR
PDP8 B OPBUS (EDPH)
Indicates that the OPBus (which is protected by longword parity) was the input to the BMux when a BMux parity error was detected at the BMux output. When this bit is set the contents of bits <28:31> are not valid.

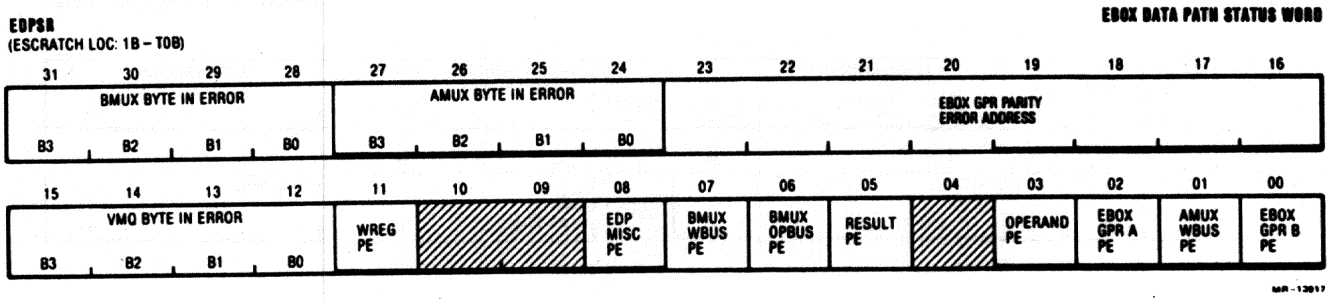
<05> RESULT PARITY ERROR
PDP8 RSLT CHK (EDPH)
If neither WREG PE <11> nor EDP MISC <08> are set, this bit indicates that the VMQSAV Register was the input to the VMQMUX when a VMQMUX parity error was detected. Otherwise, this bit is the "or" of all three of those conditions.

<04> RESERVED

<03> OPERAND PARITY ERROR
PDP8 OPER CHK (EDPH)
If bits <02:00> and <07:06> are reset, then this bit indicates that the VMQSAV Register was the input to the AMux when an AMux Parity error was detected. Otherwise, this bit indicates that one of the following errors were detected:

BMux GPR B PE	AMux GPR A PE
BMux WBus PE	AMux WBus PE
BMux OPBus PE	

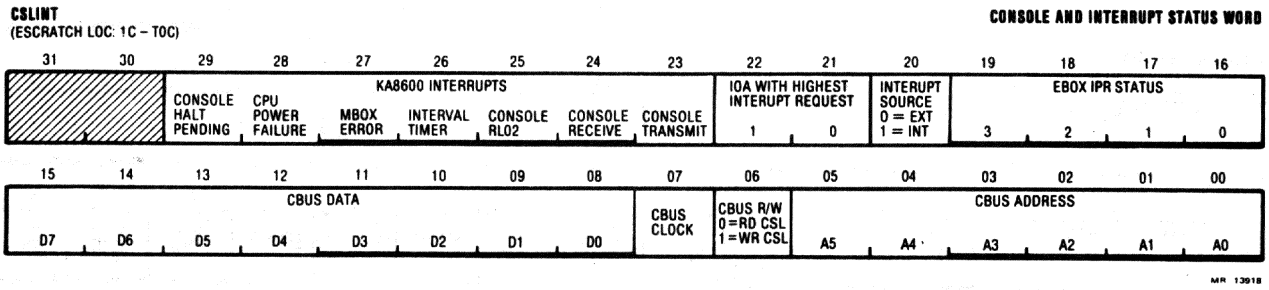
<02> EBOX GENERAL PURPOSE REGISTER A PARITY ERROR
PDP8 A RAM (EDPH)
Indicates that Scratch Pad-A was the input to the AMux when a parity error was detected at the AMux output.



<01> AMUX WBUS PARITY ERROR
PDP8 A WBUS (EDPH)
Indicates that the WBus was the input to the AMux when a parity error was detected at the AMux output.

<00> EBOX GENERAL PURPOSE REGISTER B PARITY ERROR
PDP8 B RAM (EDPH)
Indicates that Scratch Pad-B was the input to the BMux when a parity error was detected at the output of the BMux.

MACHINE CHECK STACK FRAME BIT DESCRIPTIONS
CSLINT



CSLINT (Console and Interrupt Register)

This is a two part register. Bits <31:16> reflect the System Interrupt Status and bits <15:00> reflect the CBus Status.

<31:30> RESERVED

<29:23> CPU INTERRUPT
Setting a bit in this field will result in a CPU interrupt.

<29> CONSOLE HALT PENDING
EBE3 CSL HP
Posted by the console. Indicates that the console wants to interrupt the EBox and force it to execute a Console Halt. The Ebox micro code will be forced to the CSM wait loop.

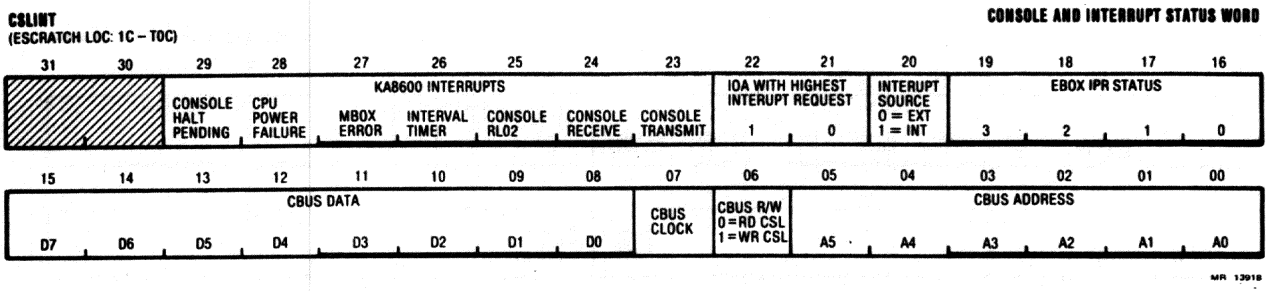
<28> CPU POWER FAIL
EBC2 CPU PF INTR LVL3
Posted by the console. Indicates that the EMM has notified the console about an impending power failure. VMS has approximately 300 ms to shut down in an orderly manner.

<27> MBOX ERROR
EBC2 MBOX INTR LVL3
Posted by the MBox. Indicates that the MBox detected an error of any type (excluding TB parity errors). The interrupt is handled by the EBox at IRD time. (Same as EBCS <14>).

<26> INTERVAL TIMER
EBE9 TIME INTR LVL3
Posted by the EBox. Indicates that the Interval Count Register has overflowed.

<25> CONSOLE RL02
EBC2 RL INTR LVL3
Posted by the console. Indicates that the console wants to interrupt the EBox to transfer a byte to or from the RL02.

During an RL02 read (SNAP files, etc.), the T11 assembles 512 bytes and then transfers them to the EBox a byte at a time via this interrupt mechanism.



During an RL02 write, the T11 interrupts the EBox for a byte of data until it has assembled 512 bytes. It then transfers the data to the RL02.

<24> **CONSOLE RECEIVE**
EBC2 TRX INTR LVL3
Posted by the console. Indicates that the console has acknowledged the last CPU transmission.

<23> **CONSOLE TRANSMIT**
EBC2 TTX INTR LVL3
Posted by the console. Indicates that the console wants to transfer a byte of information to the CPU via the CBus.

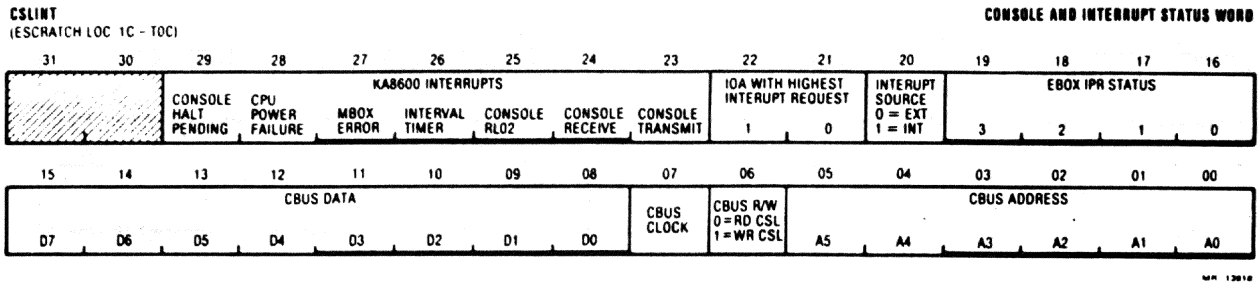
<22:21> **IOA WITH HIGHEST INTERRUPT REQUEST**
EBC1 EXT INTR SRC 1:0
The I/O Adapter (SBIA) with the highest IPR.

<20> **INTERRUPT SOURCE (0=EXT/1=INT)**
EBC3 INT INTR
Set by hardware. When set, indicates that the source of the pending interrupt is internal. When clear, indicates that the source of the pending interrupt is external.

<19:16> **EBOX IPR STATUS <03:00>**
EBC3 ACTIVE IPR 3:0
This field represents the least significant four bits of the highest active hardware (internal or external) interrupt request during the previous machine cycle. When this field is zero, there was no hardware interrupt.

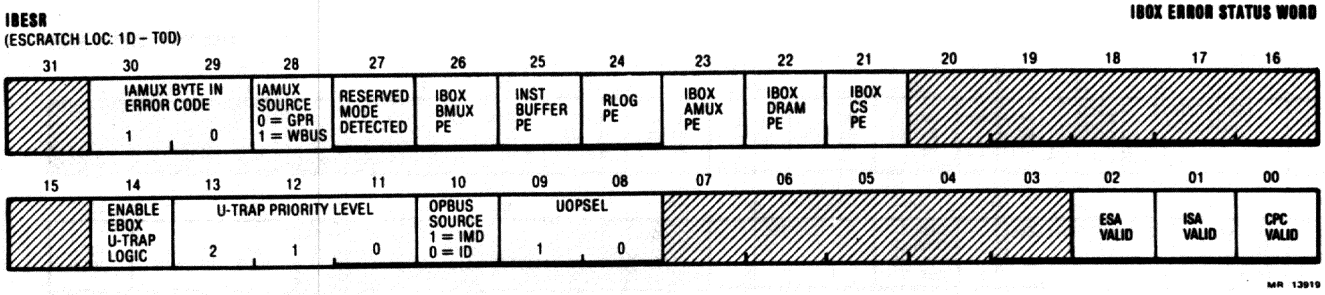
<15:08> **CBUS DATA <D7:D0>**
EBE2 CBUS IN REG D7:D0
Although this register is used primarily to receive data from the console, all data transfers between the console to the CPU are latched in this register.

<07> **CBUS CLOCK**
EBE2 CBUS CLOCK
This bit represents the state of the CBus Clock Line. The CBus Clock must be asserted by writing a one to this bit and negated (in a subsequent microinstruction) by writing a zero to this bit in order to complete a CBus read or write operation.



<06> CBUS R/W (0=RD CSL/1=WR CSL)
EBE2 CBUS WRITE
This bit determines whether a CPU read or write operation is to be performed over the CBus. The EBox is enabled to drive the CBus data lines when this bit is set. The console is enabled to drive the CBus data lines when this bit is reset.

<05:00> CBUS ADDRESS <A5:A0>
EBE2 CBUS A5:A0
The address of the console location to be read or written is loaded into this register.



This register is a combination of the IBox Error register (IBE) and the EBox Diagnostic Maintenance register (EDMS).

<31> RESERVED

<30:29> IBOX AMUX BYTE IN ERROR CODE
EBEA IAMUX EC <1:0> LTH
Indicates the most significant byte associated with the IBox AMux parity error.

Code	Byte
00	0
01	1
10	2
11	3

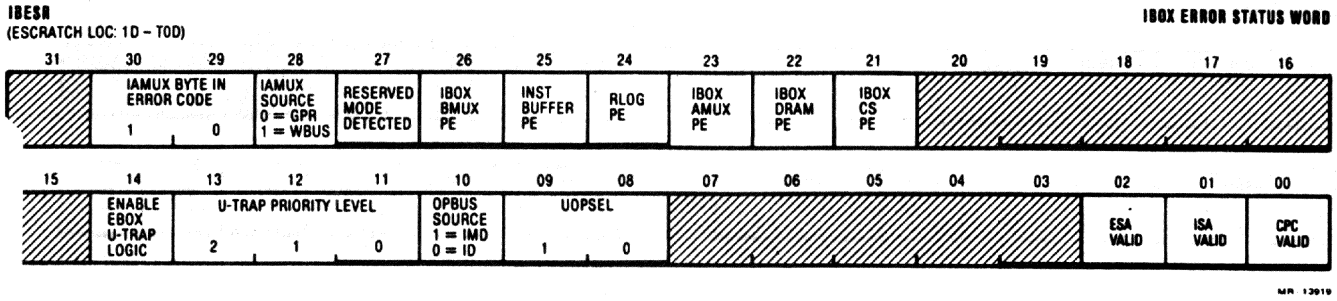
<28> IAMUX SOURCE (0=GPR/1=WBUS)
EBEA IWBUS DATA LTH
Indicates that the WBus was the input to the IBox AMux when an error was detected at the output of the IBox AMux.

<27> RESERVED MODE DETECTED
EBEA RSV MODE LTH
Indicates that an operand specifier attempted to use an addressing mode that is not allowed in the situation in which it occurred.

<26> IBOX BMUX PARITY ERROR
EBEA IBMUX PE LTH
Indicates that a parity error was detected at the output of the IBox BMux. The input to the BMux was either the IBuffer or the IMD Latch.

<25> INST BUFFER PARITY ERROR
EBEA IBUF PE LTH
Indicates that a parity error was detected on either the OPCode bytes (Byte 0 or Byte 1 in the IBuffer), or on the byte selected by the RMode Finder (IBGPR) during optimization.

<24> RLOG PARITY ERROR
EBEA RLOG PE LTH
Indicates that a parity error was detected while unwinding the RLog.



23> IBOX AMUX PARITY ERROR
EBEA IAMUX PE LTH
Indicates that a parity error was detected at the output of the IBox AMux. The input to the AMux was either the WBus or a GPR. See: IBESR <30:29> and <28>.

22> IBOX DRAM PARITY ERROR
EBEA IDRAM PE LTH
Indicates that a parity error was detected on the DRAM data.

21> IBOX CS PARITY ERROR
EBEA ICS PE LTH
Indicates that a parity error was detected on the IBox Control Store data.

20:16> RESERVED

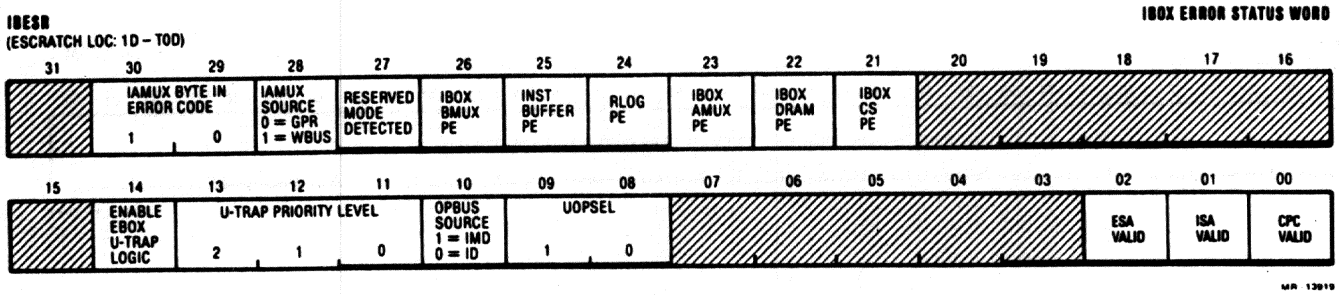
15:08> EDMS REGISTER BITS <D15:D08>
See EBEA

15> RESERVED

<14> ENABLE EBOX MICRO TRAP LOGIC
EBD3 EN ETRAP
Enables the EBox microtrap mechanism. This in turn allows CPU error reporting.

<13:11> MICRO TRAP PRIORITY LEVEL
EBDE UTRP <2:0>
This field indicates the priority of the last microtrap request.

Priority	Microtrap Type
0	EBox Read/Write Microtrap
1	OP Write Microtrap
2	IBox Error Microtrap
3	Misc Microtrap
4	Fork Microtrap
5	IMD Read Microtrap
6	ID Read Microtrap
7	STRING Read Microtrap



<10> **OPBUS SOURCE (0=ID/1=IMD)**
EBD5 SRC IMD LVL3
This bit is only valid when IBESR <09:08> = 3. When set, this bit indicates that the OPBus came from IMD register. When cleared, it indicates that the OPbus data came from ID register.

<09:08> **UOPSEL <1:0>**
EBD5 UOPSEL <1:0>
This field indicates the OPBus data source.

<1:0>	Data Source
0	IBox Register Select
1	Operand Source is EMD
2	Operand Source is IBUFFER
3	Operand Source is IMD or ID Registers (See Bit 10 above)

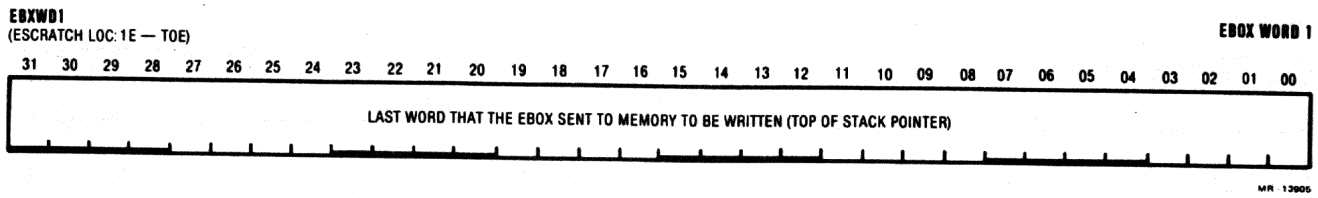
<07:03> **RESERVED**

<02> **ESA VALID**
This bit is set by the EHM if the IBox ESA VALID bit is set.

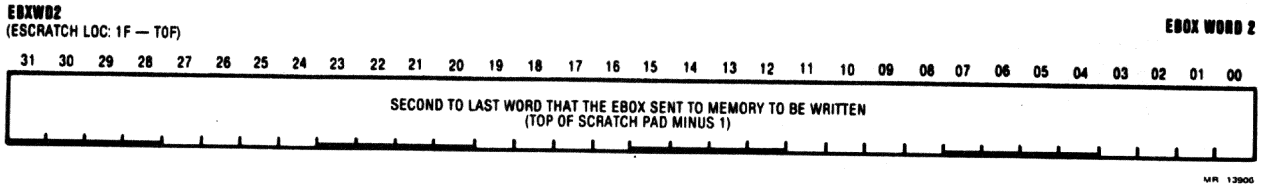
<01> **ISA VALID**
This bit is set by the EHM if the IBox ISA VALID bit is set.

<00> **CPC VALID**
This bit is set by the EHM if the IBox CPC valid bit is set.

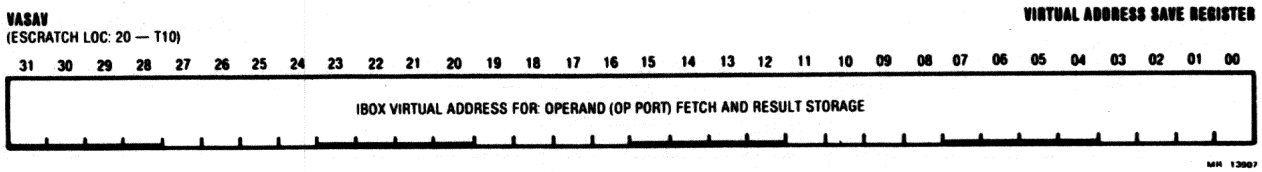
MACHINE CHECK STACK FRAME BIT DESCRIPTIONS
EBXWD1



<31:00> LAST EBOX WORD WRITTEN
This register contains a copy of the last word that the EBox wrote to the MBox. This word is obtained by popping the last word off the Scratch Pad Stack (EScratch Location F0).

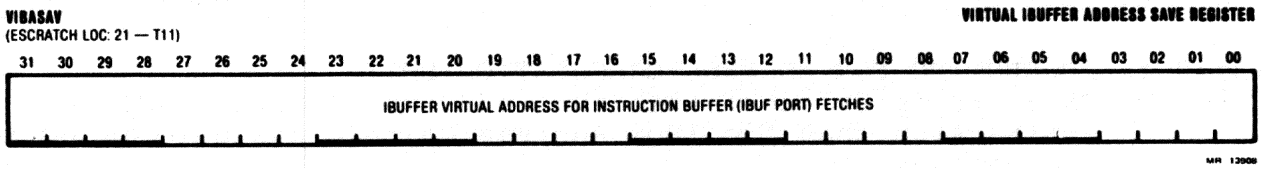


<31:00> SECOND TO LAST EBOX WORD WRITTEN
This register contains a copy of the second to last word that the EBox wrote to the MBox. This word is obtained by popping the second to last word off the Scratch Pad Stack (EScratch Location F1).

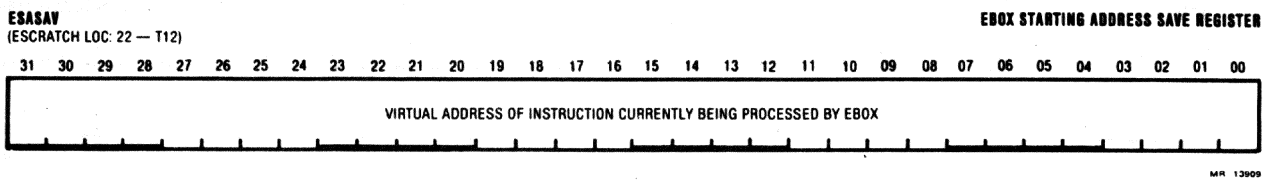


IVASAV (IBox Virtual Address Save) - This register contains the last virtual address that was calculated by the Address Calculation Unit and acknowledged by the MBox with PA ACK. Therefore, IVASAV will contain the Virtual Address associated with either the current operand fetch cycle, or the current result store cycle.

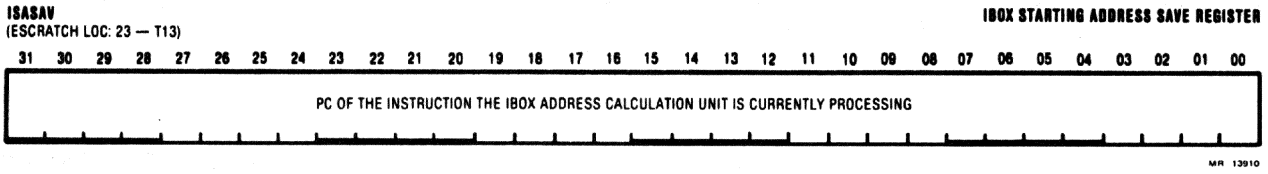
<31:00> Last Virtual Address used by the IBox for either an Operand Fetch or Result Store.



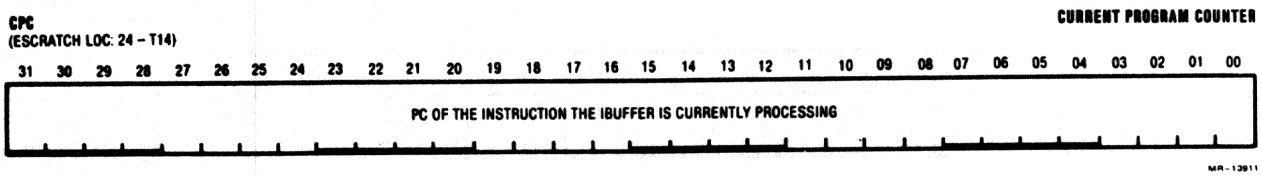
<31:00> IBOX IBUF PORT VIRTUAL ADDRESS
This register contains the last IBuffer virtual address that was acknowledged by the MBox (PA ACK).



<31:00> CURRENT PC FOR EXECUTION UNIT (EBOX)
This register contains the address (Macro PC) of the instruction that the EBox or FBox is currently processing.

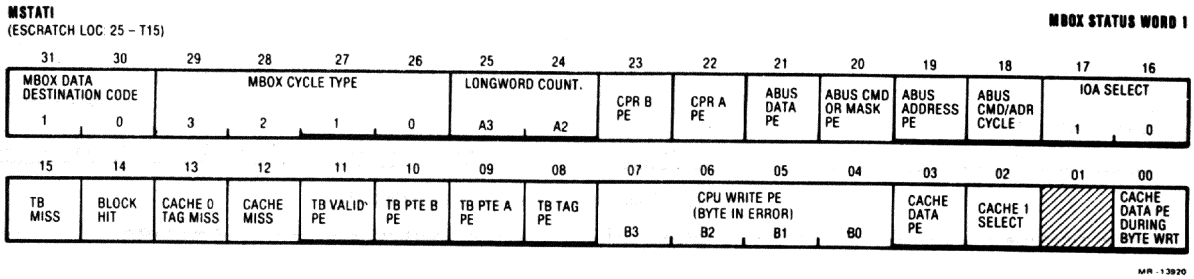


<31:00> CURRENT PC FOR ADDRESS CALCULATION UNIT
This register contains the address (Macro PC) of the instruction that the IBox Address Calculation Unit is currently processing.



<31:00> CURRENT PROGRAM COUNTER
This register contains the address (Macro PC) of the instruction that the IBox Address Calculation Unit will process next.

MACHINE CHECK STACK FRAME BIT DESCRIPTIONS
MSTAT1



This register is made up of the following MBOX REGISTERS: 2C (byte 3), 28 (byte 2), 24 (byte 1) , and 20 (byte 0).

<31:24> MBOX REGISTER 2C (MCC STATUS 1)
SOURCE: MCCJ (CRA) MBOX REG 2C (MCC STATUS 1)
ACCESS: Read Only
HELD AT: MCCJ TOT CYC ERR + T2

<31:30> MBOX DESTINATION CODE <1:0>
MCCC LAST DEST CP <1:0>
Indicates the destination code associated with the error.

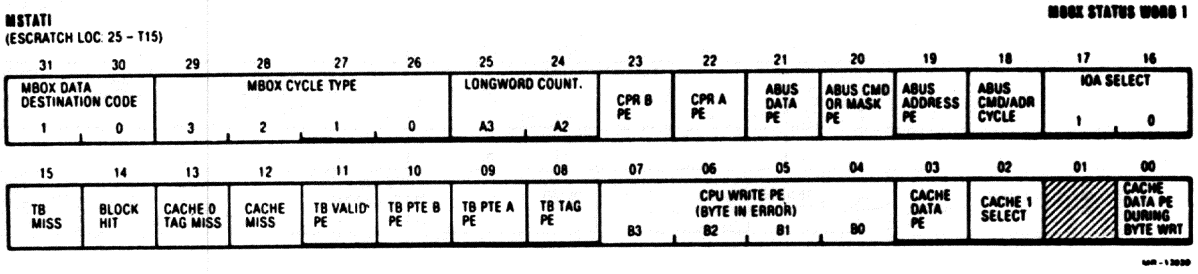
Code	Destination
00	IBUF (IBox - Don't Load Tail Pointer)
01	IMD (IBox - FETCH/STORE Operand)
10	EMD (EBox - FETCH/STORE)
11	IBUF (IBox - FETCH - Load Tail Pointer)

<29:26> MBOX CYCLE TYPE
MCCD U CYC TYP <3:0>
Indicates the microword cycle type associated with the error.

CODE	CYCLE	CODE	CYCLE
0000	NOP	1000	ABUS
0001	READ REG	1001	CP REFILL
0010	WRITE REG	1010	INVAL TB
0011	WRITEBACK	1011	TB CYC
0100	ABUS ARRAY WRT	1100	CP ARRAY WRT
0101	DATA CORRECTION	1101	CP WRITE
0110	CLEAR CACHE	1110	CP READ
0111	TB PROBE	1111	ABUS REFILL

<25:24> LONGWORD COUNT
MCCJ WD CNT <03:02>
Indicates the longword that was being processed when the error was detected.

<23:16> MBOX REGISTER 28 (MCC STATUS 3)
SOURCE: MBox Register 28 (MCA: CRB) MCCJ (CRB) MBOX REG 28 (MCC STATUS 3)
ACCESS: Read Write
HELD AT: MCCM CYC ERR SUM + T0



<23> CYCLE PARAMETER RAM B PARITY ERROR
MCCC CPR PERR B
Indicates that a Cycle Parameter RAM B parity error was detected. The MBox response is unpredictable.

<22> CYCLE PARAMETER RAM A PARITY ERROR
MCCC CPR PERR A
Indicates that a Cycle Parameter RAM A parity error was detected. The MBox response is unpredictable.

<21> ABUS DATA PARITY ERROR
MCD3 ABUS DAT PERR
Indicates that a longword parity error was detected on the ABus Data Field during either a CP I/O READ or DMA WRITE Cycle.

<20> ABUS COMMAND OR MASK PARITY ERROR
MCC4 ABUS CNTL PERR
If bit <18> is set, this bit indicates that a parity error was detected on the command/length field during an ABUS Command Address cycle. If bit <18> is reset then this bit indicates that a parity error was detected on the Mask/Status Field during an ABus Data Cycle.

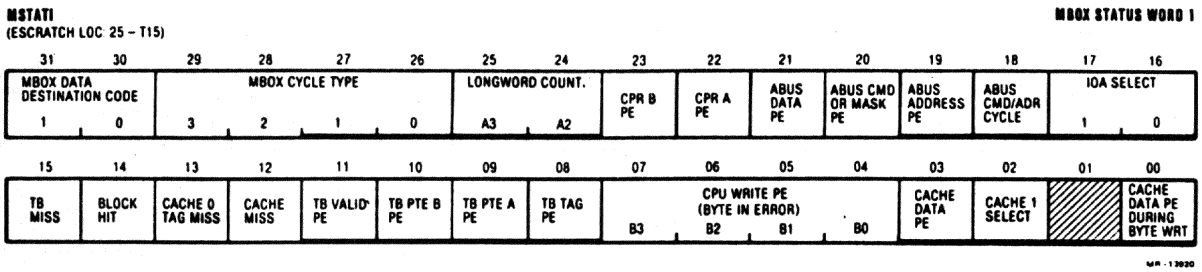
<19> ABUS ADDRESS PARITY ERROR
MAP2 ABUS ADR PERR
Indicates that a parity error was detected on the Address Field associated with an ABus Command Address Cycle.

<18> ABUS COMMAND/ADDRESS CYCLE
MCCJ ABUS LD CMD
Indicates that the MBox was executing a DMA Command Address cycle when an error was detected.

<17:16> IOA SELECT <01:00>
MCC4 ABUS SEL <01:00>
Indicates the ABus Adapter that was selected when the error was detected.

<15:08> MBOX REGISTER 24 (ADDRESS STATUS)
SOURCE: MAPP (REG) MBOX REG 24 (ADDR STATUS)
ACCESS: Read Only
HELD AT: Self holding at T3

MACHINE CHECK STACK FRAME BIT DESCRIPTIONS
MSTAT1



<15> TB MISS
MAP3 TB HIT
Indicates that the TB TAG (indexed by VA <16:09>) did not match VA <30:17> or the valid bit was not set. The MBox will send a Port Status of "A" (TB Miss) back to the EBox.

<14> BLOCK HIT
MAPL BUF BLOCK HIT
Indicates that either the Tag for Cache 0 or the Tag for Cache 1 matched PA <28:13>, at least one valid bit was set, and an I/O adapter was not selected.

<13> CACHE 0 TAG MISS
MAP3 C0 TAG MAT
Indicates that Tag in Cache 0 did not match PA <28:13>.

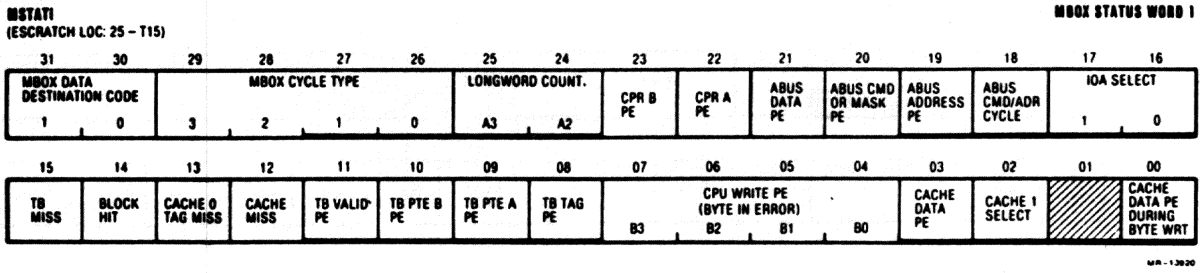
<12> CACHE MISS
MAPL BUF CACHE HIT
Indicates that either PA <28:13> failed to match both the Cache 0 Tag and the Cache 1 Tag or, if a match did occur then the valid bit for the target word(s) was not set.

<11> TB VALID PARITY ERROR
MAPP TB VAL ERR
Indicates that a parity error was detected on the valid bit when the TB was read. The MBox will send a Port Status of "8" (TB Error) back to the EBox.

<10> TB PTE B PARITY ERROR
MAP3 PTE B PAR ERR
Indicates that a parity error was detected when the TB PTE RAM containing PA <24:09> was read. The MBox will send a Port Status of "8" (TB Error) back to the EBox.

<09> TB PTE A PARITY ERROR
MAP4 PTE A PAR ERR
Indicates that a parity error was detected when the TB PTE RAM containing PA <29:25>, PROTECTION <D:A>, and MODIFY Bits were read. The MBox will send a Port Status of "8" (TB Error) back to the EBox.

<08> TB TAG PARITY ERROR
MAP3 TB TAG PAR ERR
Indicates that a parity error was detected when the TB TAG RAMs were read. The MBox will send a Port Status of "8" (TB Error) back to the EBox.



<07:00> **MCDU REGISTER 20**
SOURCE: MCDU REG 20 (DATA STATUS)
ACCESS: Read Only
HELD AT: MCCJ TOT CYC ERR + T3

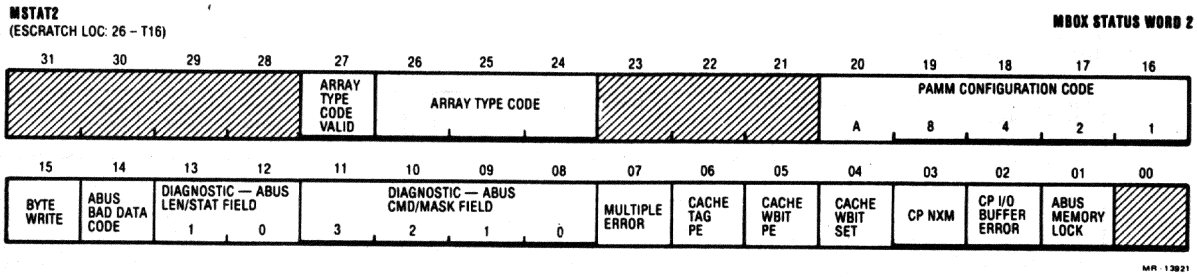
<07:04> **CPU WRITE (BYTE IN ERROR)**
UFO5 WR BYT <3:0> PERR (MCDU)
This field indicates that a parity error was detected on the data received from the CPU during a CPU write. Furthermore, this field indicates which byte(s) had the parity error.

<03> **CACHE DATA PARITY ERROR**
UFO5 CACHE BYT PERR (MCDU)
Indicates that a byte parity error was detected on data read from cache during any Cache operation other than a CPU Byte Write Hit. Includes: CPU Read, DMA Read, Writeback, and DMA Masked Write.

<02> **CACHE 1 SELECT**
MAPL CACHE 1 DAT
This bit indicates the Cache that was selected when the error was detected.

<01> **ARRAY READ**
MCCJ ANY REFILL
Indicates that a Cache Refill was in progress when an error was detected.

<00> **CACHE DATA PARITY ERROR DURING BYTE WRITE**
UFO5 CACHE BWRT PERR (MCDU)
Indicates that a Cache Data Parity Error was detected during a CP Byte-write Operation.



This register is made up from the following MBox Registers: 54 (byte 2), 5C (byte 1), and 58 (byte 0).

<31:24> **VMS SUPPLIED**
This field is supplied by VMS and describes the array type selected at the time that the error occurred.

<31:28> **RESERVED**

<27> **ARRAY TYPE CODE VALID**
Indicates that the Array Type Code in bits <26:24> is vaild.

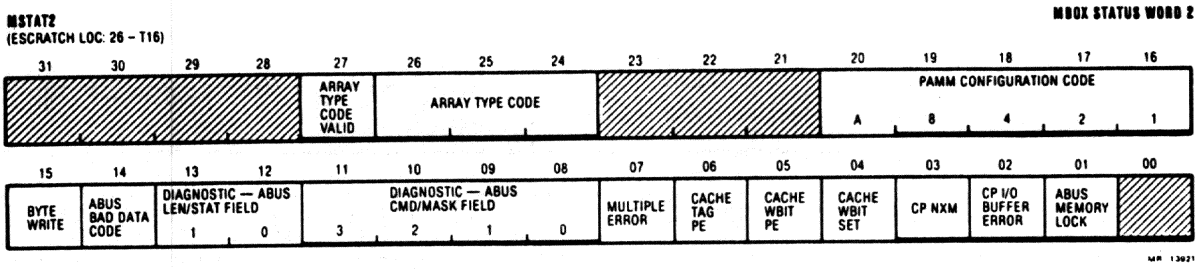
<26:24> **ARRAY TYPE CODE**
Indicates the Array type that was selected when the error occurred.

CODE	ARRAY TYPE	CODE	ARRAY TYPE
000	Reserved	100	Reserved
001	16 Mb Array	101	Reserved
010	04 Mb Array	110	Reserved
011	64 Mb Array	111	Reserved

<23:21> **RESERVED**

<20:16> **MBOX REGISTER 54**
SOURCE: MAP9 (RAM) MBOX REG 54 (PAMM)

<20:16> **PAMM CONFIGURATION CODE <A:1>**
MAP9 PAMM CONF <A:1>
The Error Handling Microcode uses MEAR <29:20> to address the PAMM and then loads the five-bit PAMM code into this field. Normally this field will indicate the Array Module or I/O Adapter selected when the error was detected. If, however, the error involved a CP to I/O transfer, then this field will not be valid.



CODE	SELECTS	CODE	SELECTS
00	Array Slot 0	18	I/O Adapter 0
01	Array Slot 1	19	I/O Adapter 1
02	Array Slot 2	1A	I/O Adapter 2 (Not used)
03	Array Slot 3	1B	I/O Adapter 3 (Not used)
04	Array Slot 4	1C	Reserved
05	Array Slot 5	1D	Reserved
06	Array Slot 6	1E	Reserved
07	Array Slot 7	1F	Non-Existent Address
08 - 17	Reserved		

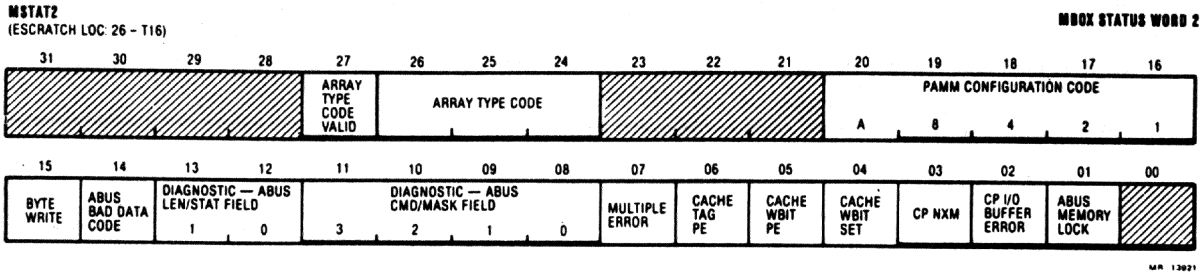
If a 16 MB Array was selected (MSTAT2 <26:24> = 001) use the syndrome (MDECC <14:09>) to determine which bit is in error. Use MEAR <23:22> and the Longword Count (MSTAT1 <25:24>) to determine the failing longword. Now, identify the failing SMU from the following chart. If check bit C0 is at fault, MEAR <22> determines which of the two rows is to be used.

	FAILING LONGWORD			
DATA BIT IN ERROR	Wd 0	Wd 1	Wd 2	Wd 3
0 - 15, C2, C4, C5 C0 (if MEAR <22> = 0)	SMU1	SMU2	SMU3	SMU5
16 - 31, C1, C3, C6 C0 (if MEAR <22> = 1)	SMU6	SMU7	SMU4	SMU8

If a 64 MB array was selected (MSTAT2 <26:24> = 011) use the longword count (MSTAT1 <25:24>) to determine the failing SMU according to the following chart.

MSTAT1 <25:24>	FAILING DAUGHTERBOARD
00	SMU1
01	SMU2
10	SMU3
11	SMU4

MACHINE CHECK STACK FRAME BIT DESCRIPTIONS
MSTAT2



<15:08> MBOX REGISTER 5C
SOURCE: MCCJ (CRA) MBOX REG 5C (MCC STATUS 2)

<15> BYTE WRITE
MCC7 CP BYTE WRITE
Indicates that the MBox was servicing a byte write request.

<14> ABUS BAD DATA CODE
MCC4 AB BAD DAT ERR
Indicates that the CP IO read data received from the ABus Adapter was marked as bad data via the ABus Status Field.

<13:12> DIAGNOSTIC ABUS LENGTH/STATUS FIELD
MCC4 LABUS STAT <1:0>
Indicates the state of the ABus Length and Status lines during an I/O diagnostic operation.

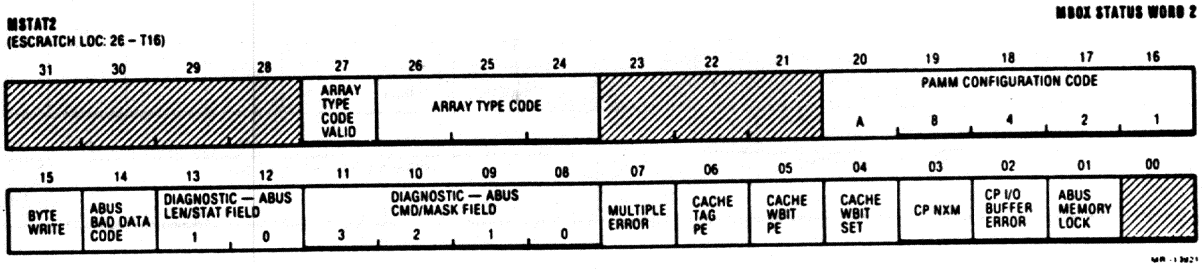
<11:08> DIAGNOSTIC ABUS COMMAND/MASK FIELD
MCC4 LABUS MSK <3:0>
Indicates the state of the ABus Command and Mask lines during an I/O diagnostic operation.

<07:00> MBOX REGISTER 58
SOURCE: MCCJ (CRB) MBOX REG 58 (MCC STATUS 4)

<07> MULTIPLE ERROR
CRB5 MULT ERR (MCCJ)
Indicates that a second MBox error was detected before the EBox could read and clear the first error. Most or all of the information associated with the second error will be lost.

<06> CACHE TAG PARITY ERROR
CRB5 CACHE TAG PERR (MCCJ)
Indicates that a parity error was detected in the address and valid bit portion of the Cache tag.

<05> CACHE WRITTEN BIT PARITY ERROR
CRB5 TAG WRT PERR (MCCJ)
Indicates that a parity error was detected in the Cache tag Written Bit Field. The MBox handles the request as though the written bit was a one.



<04> CACHE WRITTEN BIT SET
CRB5 WRITTEN (MCCJ)
Indicates that the cache written bit was set when an error was detected.

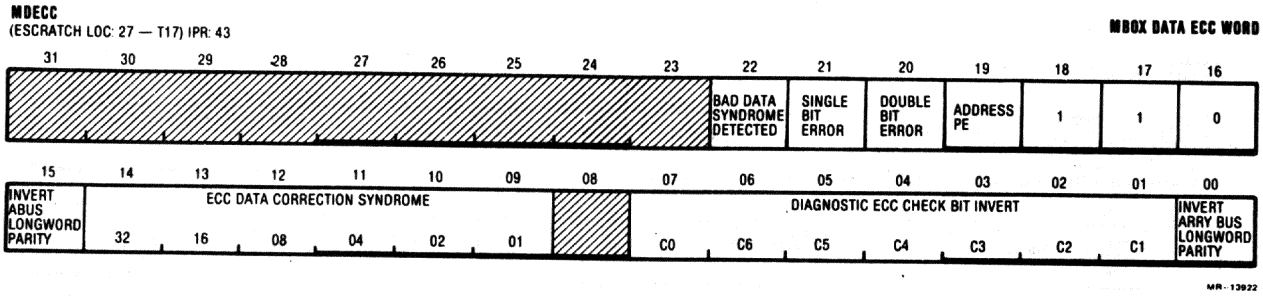
<03> NON EXISTENT MEMORY
CRB5 NXM (MCCJ)
Indicates that either the memory request was to Non-Existent Memory or an I/O adapter attempted to address another I/O adapter. An MBox Fatal Error status code is sent to the EBox for CP initiated requests, and the DMA ERROR line is asserted to the I/O Adapter for I/O initiated requests. All writes are cancelled.

<02> CP I/O BUFFER ERROR
CRB5 CP IO BUF ERR (MCCJ)
Indicates that the selected ABus Adapter detected an error while processing a CPU request.

<01> ABUS MEMORY LOCK
CRB5 LOCK LINE (MCCJ)
When set, indicates that the Memory Lock Line was being driven on the ABus. This line may be driven by either the MBox or an ABus Adapter.

<00> RESERVED

MACHINE CHECK STACK FRAME BIT DESCRIPTIONS
MDECC



MDECC (MBox Data ECC Status Register) - This register is made up of three MBox registers; 70 (byte 2), 60 (byte 1), and 50 (byte 0).

<31:24> **Reserved**

<23:16> Source: MCDM (ECC) MBox Reg 70 (DATA ECC ERROR)
Held at: ECC4 ERR HLD and T2D

<23> **Reserved**

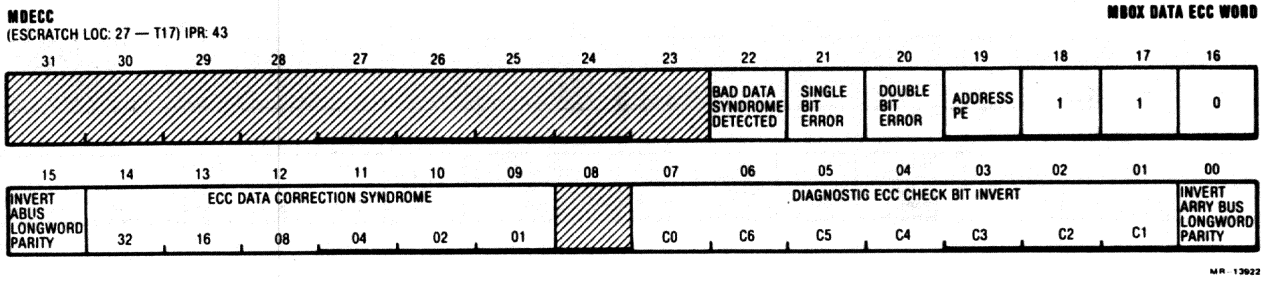
<22> **BAD DATA SYNDROME DETECTED**
ECC4 BD ERR (MCDM)
The bad data bit is XORed into the ECC check bit generation whenever "known" bad data is written into either the cache or the array. A read to that location will result in the detection of a Bad Data code and cause this bit to be set. Note that to read check bits from cache a cache byte parity error must have been found to invoke the check bit read.

<21> **SINGLE BIT ERROR**
ECC4 SB ERR (MCDM)
The data word read from cache or main memory had a single bit (ECC correctable) error.

<20> **DOUBLE BIT ERROR**
ECC4 DB ERR (MCDM)
The data word read from cache or main memory had a double bit error or a detectable multiple bit error.

<19> **ADDRESS PE**
ECC4 AP ERR (MCDM)
The word fetched from memory was either written or read from the wrong location. The ECC code indicates that the parity of the address written and the parity of the address read from are different. ECC is generated over the data bits and a parity bit computed over the physical address bits <29:04>.

<18:16> **110**
ECC2 DATA <33:32> (MCDM)
These bits should always return as a binary 110. Therefore, this field position can be used as a key to confirm the location of MDECC in a Stack Frame.



<15:08> Source: MCDM (ECC) MBox Reg 60 (DATA SYNDROME)
Held at: ECC4 ERR HLD

<15> **INVERT ABUS LONGWORD PARITY**
ECC3 LWP INVERT REG (MCDM)
When this bit is set, and bit <01> in Register 10 (Data Control 2) is set the ECC MCA will generate bad ABus longword parity.

<14:09> **ECC DATA CORRECTION SYNDROME**
ECC3 SYN REG <32:1> (MCDM)
ECC3 SYN REG <32:1> (MCDM) - This field corresponds to the syndrome generated by the ECC chip and indicates the failing bit number.

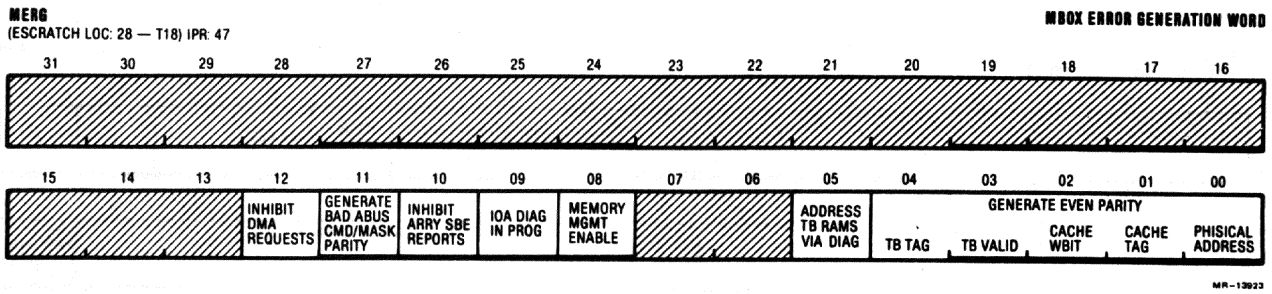
SYNDROME	(MSB)	70	0421000	66666655555444443333322222111111
IN OCTAL	(LSB)	07	0000421	65432132654654326543265432654321
DATA BIT	(MSB)	BA	CCCCCCC	33222222222211111111110000000000
	(LSB)	DP	0654321	10987654321098765432109876543210

<08> **Reserved**

<07:00> Source: MCDM (ECC) MBox Reg 50 (DATA CHECK INVERT)

<07:01> **DIAGNOSTIC CHECK BIT INVERT <CP:C1>**
ECC3 <CP:C1> INVERT REG (MCDM)
When set the corresponding ECC check bits will be inverted.

<00> **INVERT ARRY BUS LONGWORD PARITY**
ECC3 LWP INVERT REG (MCDM)
When this bit is set, and bit <01> in Register 10 (Data Control 2) is set, then the longword parity generated on ARRAY BUS data will be inverted when the ECC MCA is the check mode. This bit is write only and is read via bit <15> in MDECC (MBox Register 60).



MERG (MBox Error Generation Register) - This register is made up of MBox registers 18 (byte 1) and 14 (byte 0).

<31:16> **Reserved**

<15:08> Source: MCCJ (CRB) MBox Reg 18 (MCC CONTROL 3)
Note: This field is set by loading the Control Register on CRB4.

<15:13> **Reserved**

<12> **INHIBIT DMA REQUESTS**
MCCJ INH DMA REQ
When set, the Mbox will not honor any DMA requests.

<11> **GENERATE BAD ABUS CMD/MASK PARITY**
MCCJ CMD BAD PAR
When set, ABus Command/Length and ABus Mask/Status will be transferred with even parity.

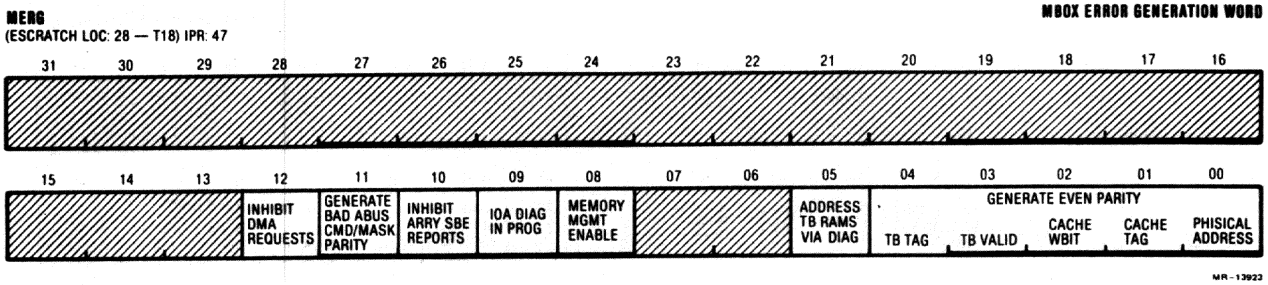
<10> **INHIBIT ARRY SBE REPORTS**
MCCJ INH ARY CORR REP
When set, this bit prevents the MBox from reporting ECC correctable errors. The errors will still be corrected as usual.

<09> **IOA DIAG IN PROG**
MCCJ IOA DIAG IN PROG
When set, the Abus command/mask field and length/status lines are driven from MCC CONTROL REGISTER 1C <03:00>.

<08> **MEMORY MGMT ENABLE**
MCCJ MEM MAN EN
When set, memory management is enabled. All virtual references are then translated by the TB. When this bit is cleared, all references will be treated as physical and all TB parity error checking will be disabled.

<07:00> Source: MAPP (REG) MBox Reg 14 (ADDR CONTROL 2)

<07:06> **Reserved**



<05> **ADDRESS TB RAM VIA DIAG**
MAPP ADR RAM DIAG
The effect of this bit depends on the cycle type:

Write TB - Causes the TB RAMs containing physical address bits <12:09> and selected by the IVA lines to be written instead of those selected by the EVA lines. Used for diagnostic testing of TB RAMs.

Read TB - Causes the contents of the TB RAMs containing physical address bits <12:09> and selected by the IVA to be read instead of those selected by the EVA lines. Used for diagnostic testing of TB RAMs.

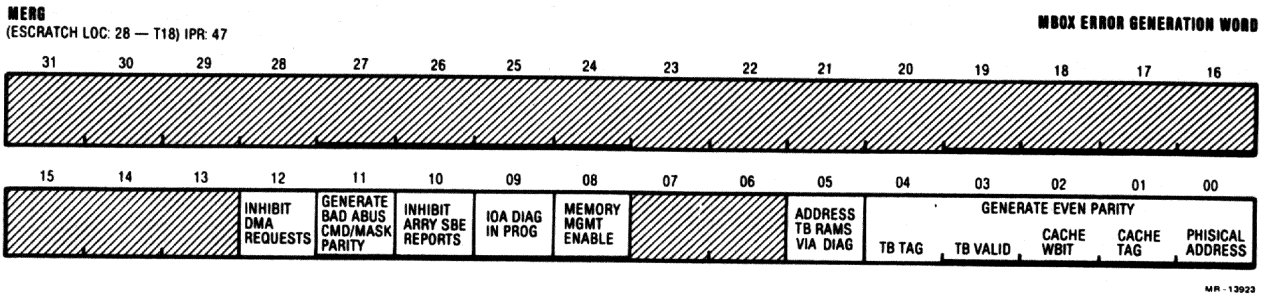
Diagnostic Read Cache Tag Address - Causes the written bit, written parity valid bits and cache tag parity to be read in place of the cache tag address bits. Used for diagnostic testing of the cache tag RAMs.

<04:00> **GENERATE EVEN PARITY**
When set, the bits in this field will cause the MBox to generate even (bad) parity for the corresponding function.

<04> **GENERATE EVEN PARITY TB TAG**
MCCJ EVN TB TAG PAR
When set, a write to the TB will result in the TB tag being written with even parity. A read to this TB location will result in a TB TAG parity error.

<03> **GENERATE EVEN PARITY TB VALID**
MAPP GEN TB VAL ERR
When set, a write to the TB will result in the TB Valid bit field being written with even parity. A read to this TB location will result in a TB Valid parity error.

<02> **GENERATE EVEN PARITY CACHE WBIT**
MAPP GEN EVN WBIT PAR
When set, parity for the cache written bit will be complemented before being written in the cache. A read to this Cache location will result in a Cache WBit parity error.



<01> **GENERATE EVEN PARITY CACHE TAG**
MAPP GEN EVN TAG PAR
When set, the cache data address tag is stored with even parity during a cache write. A read to this Cache location will result in a Cache Tag parity error.

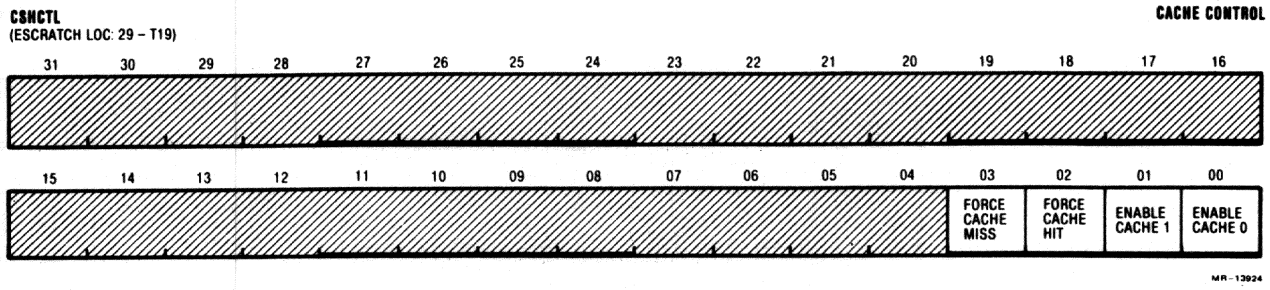
<00> **GENERATE EVEN PARITY PHYSICAL ADDRESS**
GEN EVN PA PAR
The effect of this bit depends on the type of operation the MBox is performing.

Cache or Array Writes - When set, the address parity bit generated as part of the ECC character generation will be complemented. If the write was to the array this condition will be detected when that array location is next read. If the write was to the Cache a Cache Byte parity error will have to be forced by some other means in order to detect this condition.

DMA - The MBox will detect an ABus Address Parity Error.

CP to I/O Transfers - The ABus Adapter will detect bad address parity and set CP I/O Buffer Error.

TB Writes - Bad parity will be forced in both PTE A and PTE B.



This register is made up of MBOX Register 04 (byte 0).

<31:08> **RESERVED**

<07:00> **SOURCE: MAPP (REG) - MBOX REG 04 (ADDR CONTROL 1)**

<07:04> **RESERVED**

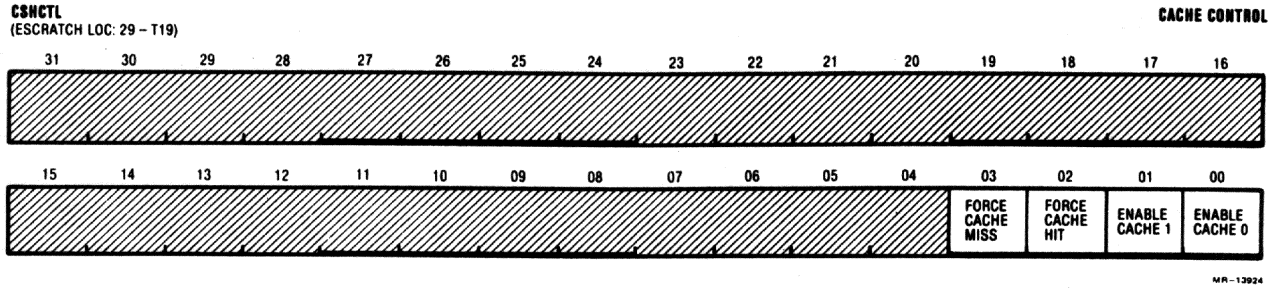
<03> **FORCE CACHE MISS**
REG4 FORCE CACHE MISS (MAPP)
When set, forces a cache miss (overrides CACHE 1 ENABLE and CACHE 0 ENABLE). This bit is used for forcing a cache refill during diagnostics.

<02> **FORCE CACHE HIT**
REG4 FORCE CYC (MAPP)
This bit works in conjunction with CACHE 1 ENABLE and CACHE 0 ENABLE as shown in the following table:

Force Hit	C1 En	C0 En	Function
0	0	0	Cache Miss
0	0	1	Cache 0 determines Hit or Miss
0	1	0	Cache 1 determines Hit or Miss
0	1	1	Both Caches determine Hit or Miss
1	0	0	Cache Miss
1	0	1	Force Hit in Cache 0
1	1	0	Force Hit in Cache 1
1	1	1	Illegal, gives tag parity error with Written Bit = 1

The three bits <02:00> reflect the following:

Code 000 - Cache 0 and 1 Disabled
Code 001 - Cache 1 Disabled
Code 010 - Cache 0 Disabled
Code 011 - Normal Running Code.
Code 100 - Cache 0 and 1 Disabled
Code 101 - Used to force a sweep of Cache 0 if Written Bit is set. Also used for diagnostic purposes to force a hit in a given cache.

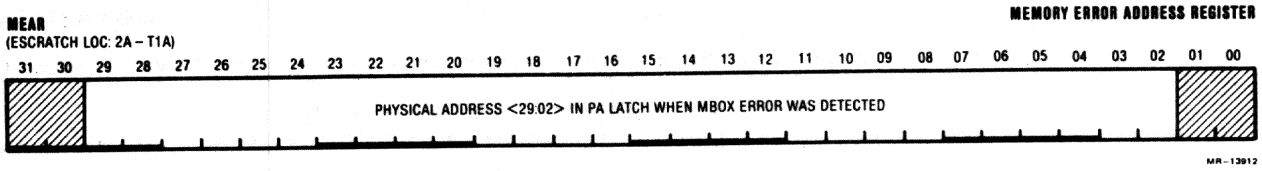


Code 110 - Used to force a sweep of Cache 1 if Written Bit is set. Also used for diagnostic purposes to force a hit in a given cache.

Code 111 - This is an illegal code that covers the case of a hit in both caches. A tag parity error with W=1 is also forced if, during normal operation, both caches give indications of a hit. This is considered an MBOX FATAL ERROR.

<01> **ENABLE CACHE 1**
REG4 CACH 1 ON (MAPP)
When set, enables Cache 1 (allows Cache 1 to be read and written).

<00> **ENABLE CACHE 0**
REG4 CACH 0 ON (MAPP)
When set enables Cache 0 (allows Cache 0 to be read and written).



This is a copy of MAP1/3 (ADA/ADB) MBOX REG 7C (ERROR ADDR). It contains the physical address present at the output of the PA Mux when the MBox detected an error. Interpreting this address depends on the MBox Data Destination Code (MSTAT1 <31:30>) and the MBox Cycle Type (MSTAT1 <29:26>).

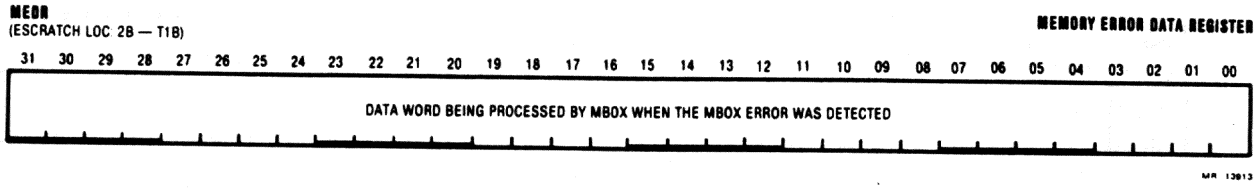
1. CP initiated (Non I/O) Read/Write Operations - MEAR will contain the address of the longword that was associated with the error unless:

   CP REFILL or WRITEBACK - (The CP request caused either a Cache Refill Cycle, or a Cache Writeback Cycle). Then MSTAT1 <25:24> (Longword Count <A3:A2>) must be used to determine the address of the longword that was associated with the error. If MSTAT1 <25:24> equal MEAR <03:02> then MEAR contains the address of the longword associated with the error. Otherwise, you must substitute MEAR <03:02> with MSTAT <25:24> to determine the address.

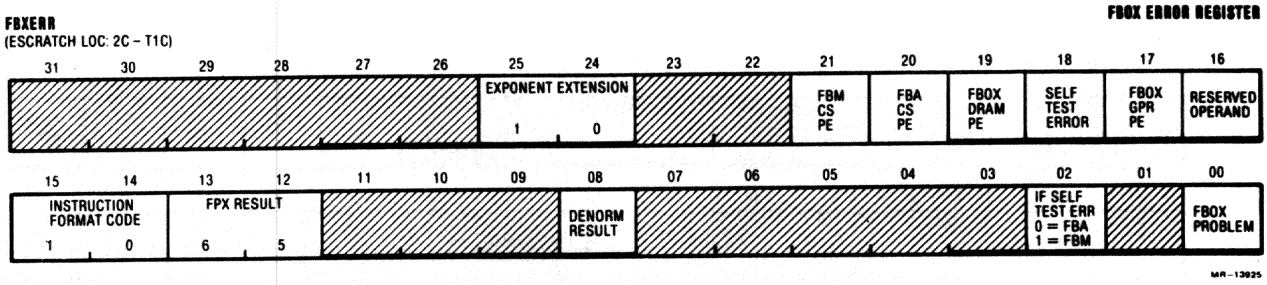
2. CP Initiated I/O Read/Write Cycles - MEAR is not valid for any CP initiated I/O Cycles.

3. ABus Initiated Read/Write Cycles - MEAR will contain the starting address of the data to be processed (i.e., longword, quadword, or octaword). MSTAT1 <25:24> (Longword Count <A3:A2>) must be used to determine the address of the longword that was associated with the error. If MSTAT1 <25:24> equal MEAR <03:02> then MEAR contains the address of the longword associated with the error. Otherwise, you must substitute MEAR <03:02> with MSTAT <25:24> to determine the address.

MACHINE CHECK STACK FRAME BIT DESCRIPTIONS
MEDR



This is a copy of MCD1/3 (MDP) MBOX REG 78 (DATA ERROR). It is only valid (held) for ABus Parity Errors (DMA Data PEs and CPU Read PEs only), ABus BDC Errors, and CPU Write PEs. It contains the data word latched in the MCD MCAs when the error occurred.



The EHM builds this register in the EBox Scratch Pads by reading three FBox registers: 03, 02, and 01.

<31:26> RESERVED

<25:24> EXPONENT EXTENSION <01:00>
FBRD EXT <1:0> (FA01)
This field, which is an extension of the exponent field, indicates overflow and underflow conditions as follows:

FIELD	CONDITION
00	Normal
01	Overflow
10	Underflow
11	Underflow

<23:22> RESERVED

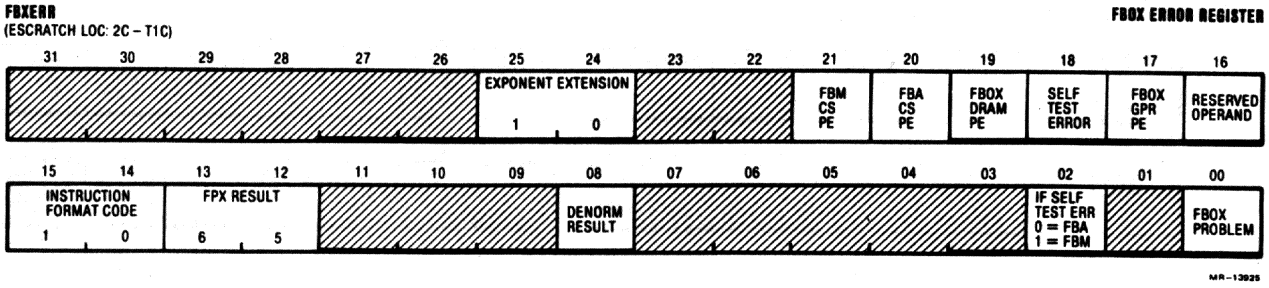
<21> FBM CONTROL STORE PARITY ERROR
MPZ5 CS PAR ERR (FM07)
Set when the FBM module detected an FBM Control Store Parity Error. When set, the CS Address is latched in the FBM MSQ MCA.

<20> FBA CONTROL STORE PARITY ERROR
ACC4 RAM PERR (FA13)
Set when the FBA module detected an FBA Control Store Parity Error. When set, the Control Store Address is latched in the FBA MSQ MCA.

<19> FBOX DRAM PARITY ERROR
MCB3 FDRAM PAR ERR (FM11)
Set when the FBM module detected a Dispatch RAM Parity Error. Neither the DRAM address nor the DRAM data are latched.

<18> SELF TEST ERROR
FBR3 SELF TEST ERROR (FA01)
Set when the FBox detected an error while running the Self Test. Bit <02> will indicate whether the error was associated with the FBA or FBM Module.

<17> FBOX GENERAL PURPOSE REGISTER PARITY ERROR
FBR4 GPR ERROR (FA01)
Set when the FBA Module detected a GPR Parity Error. The specific byte in error cannot be identified.



<16> **RESERVED OPERAND**
FBR6 RESERVED OP (FA01)
This bit generally indicates a software problem and has no significance in the context of a hardware error. It indicates that one of the operands received by the FBox had a negative sign and an exponent of zero.

<15:14> **INSTRUCTION FORMAT CODE <01:00>**
FBR9 FORM <01:00> (FA01)
This field indicates the format of the instruction that the FBox was executing when the FBox status was saved in this register.

CODE	FORMAT
00	F
01	G
10	D
11	I or H

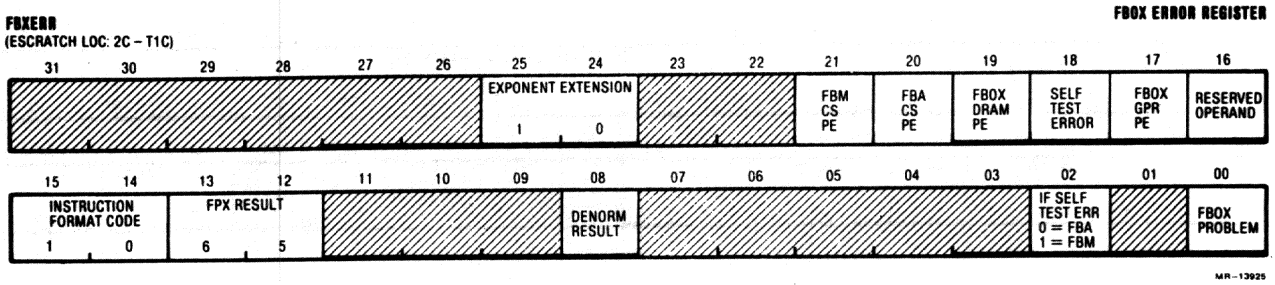
<13:12> **FPX RESULT <06:05>**
FXP6 FXRES <5:6> (FA01)
This field has no significance in the context of an error. It represents exponent result bits <13:12>.

<08> **DENORM RESULT**
FBR1 DENORM (FA01)
This bit has no significance in the context of errors. It indicates that a rounding operation resulted in a "carry out".

<07:03> **RESERVED**

<02> **IF SELF TEST ERROR (0=FA/FM=1)**
FBR4 WBUS D00 (FA01)
This bit has a double meaning. During normal operation this bit indicates that the divisor was equal to zero. If SELF TEST ERROR is set, however, this bit indicates which FBox module detected the Self Test error (0=FA/FM=1).

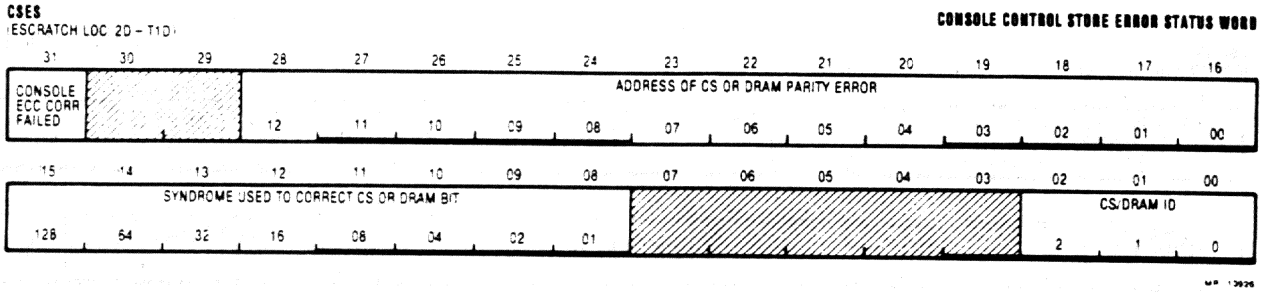
<01> **RESERVED**



<00> **FBOX PROBLEM**
FBR1 FBOX PROBLEM (FA01)
Set when the FBox detected one of the following conditions:

Exponent Extension Problem ..... Bits <25:24>
GPR Parity Error ..... Bit <17>
FBM Control Store Parity Error ... Bit <21>
FBA Control Store Parity Error ... Bit <20>
FDRAM Parity Error ..... Bit <19>
Self Test Error ..... Bit <18>
Divide by Zero ..... Bit <02>
Denormalize Result ..... Bit <08>
Reserved Operand ..... Bit <16>

MACHINE CHECK STACK FRAME BIT DESCRIPTIONS
CSES



The contents of this register are generated by the console during Control Store and Dispatch RAM ECC correction. The console sends this register to the CPU as part of the ECC correction process.

<31> CORRECTION FAILURE
Indicates that the console was unable to correct the Control Store or Dispatch RAM identified in the CS/DRAM ID field <02:00>.

<30:29> RESERVED

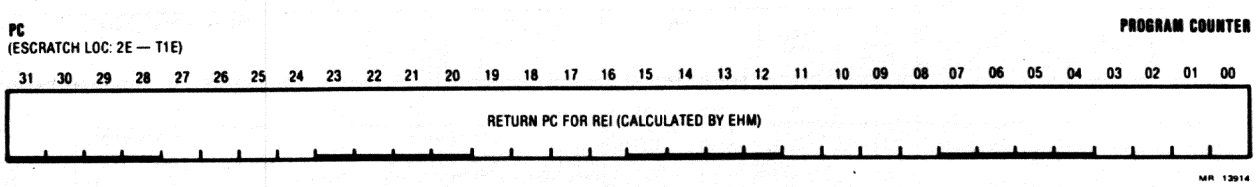
<28:16> CONTROL STORE ADDRESS
Address the console used when correcting the Control Store or DRAM Parity Error.

<15:08> CONTROL STORE SYNDROME
This is the syndrome the console used to correct the bit in error.

<07:03> RESERVED

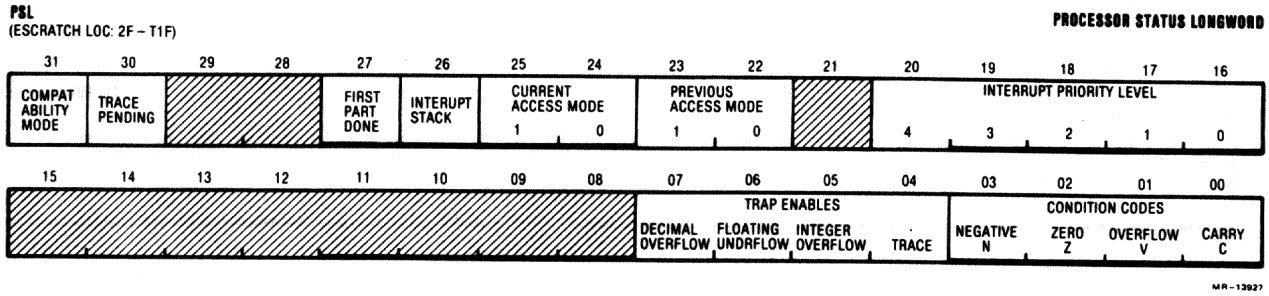
<02:00> CONTROL STORE/DRAM IDENTIFICATION CODE
This is the code passed to the console from the EBox to determine what Control Store to apply correction to.

CODE	MEANING
0	No Error
1	EBox Control Store Parity Error
2	IBox Control Store Parity Error
3	IBox DRAM Parity Error
4	FBox DRAM Parity Error
5	FBox Adder Module Control Store Parity Error
6	FBox Multiplier Module Control Store Parity Error
7	MBox Control Store Errors (Data, Address, or Micro-stack)



This register contains the PC to be used by VMS if it determines that an REI is possible. The contents of this register are determined by the Error Handling Microcode. Depending on the instruction in the pipeline that was associated with the error, this register will reflect the CPC, the ISA, or the ESA.

MACHINE CHECK STACK FRAME BIT DESCRIPTIONS
PSL



<31> COMPATIBILITY MODE
When set, indicates that the processor is in PDP-11 compatibility mode. When clear, indicates that the processor is in native (VAX) mode.

<30> TRACE PENDING
This bit works in conjunction with Enable Trace bit <04>. If Enable Trace is set at the beginning of an instruction then the processor will automatically set this bit and thus cause a trap to the Trace Handler Routine.

The Trace Handler Routine will gather the desired information about the state of the system and REI (leaving the Enable Trace and Trace Pending bit alone). This will allow the next instruction to be traced. When the Trace Handler Routine wants to discontinue tracing it will clear both bits (Enable Trace and Trace Pending).

<29:28> RESERVED

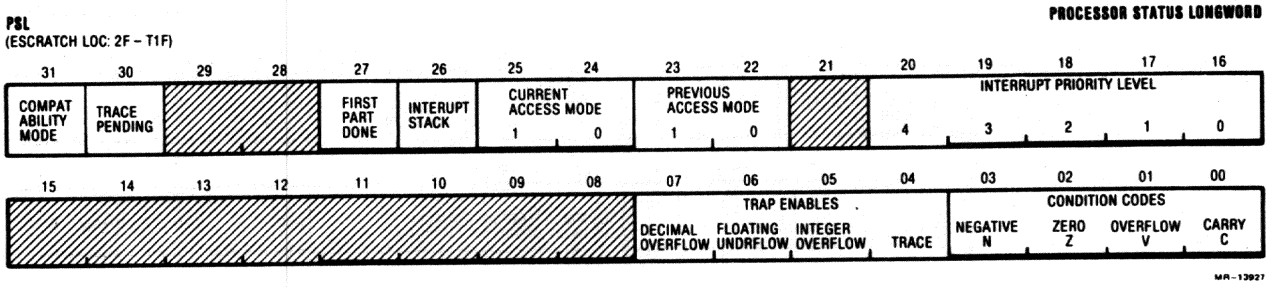
<27> FIRST PART DONE
This bit is set by long instructions that can be interrupted during execution (e.g., Move String). The REI following the interrupt will continue the interrupted instruction.

<26> INTERRUPT STACK
When set, the processor is executing on the interrupt stack. Any mechanism that sets this bit also clears current mode and raises the IPL above 0. If an REI attempts to restore a PSL with IS=1 and non-zero current mode or zero IPL, a reserved operand fault is taken. When this bit is clear, the processor is executing on the stack specified by current mode. At bootstrap time, IS is set.

<25:24> CURRENT ACCESS MODE
This field indicates the access mode of the currently executing process, as follows: 0 - KERNEL, 1 - EXECUTIVE, 2 - SUPERVISOR, 3 - USER

<23:22> PREVIOUS ACCESS MODE
This field is loaded from the Current Access Mode by exceptions and CHMx instructions. It is cleared by interrupts, and restored by REI's.

<21> RESERVED



<20:16> INTERRUPT PRIORITY LEVEL
Indicates the current processor priority, in the range 0 to 1F (Hex). The processor will accept interrupts only on levels greater than the current level. At bootstrap time, IPL is initialized to 1F (Hex).

<15:08> RESERVED

<07> DECIMAL OVERFLOW TRAP ENABLE
When set, it forces a decimal overflow trap after execution of an instruction that had a conversion error, or produced a result with a decimal overflow. When this bit is clear, no trap will occur, however, the condition code V bit will still set.

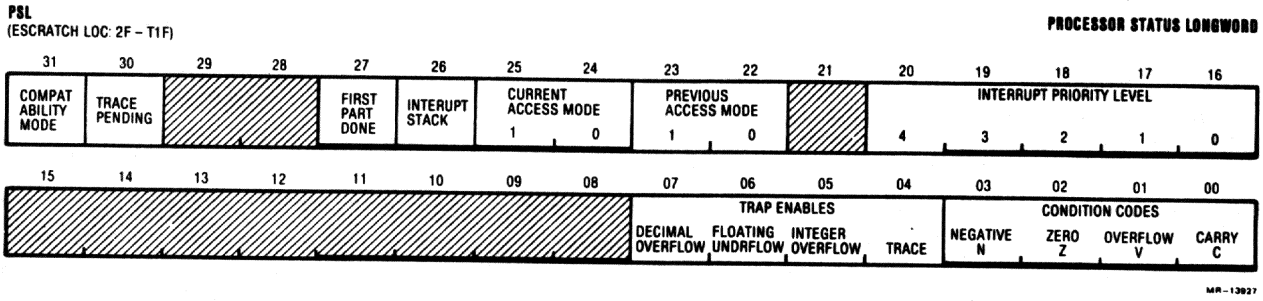
<06> FLOATING UNDERFLOW EXCEPTION ENABLE
When this bit is set, it forces a floating underflow exception after execution of the instruction that produced a result with an underflow (e.g., a result exponent, after normalization and rounding, less than the smallest representable exponent for the data type). When this bit is clear, no exception occurs.

<05> INTEGER OVERFLOW TRAP ENABLE
When this bit is set, it forces an integer overflow trap after execution of an instruction that produced an integer result that overflowed or had a conversion error. When this bit is clear, no integer overflow trap will occur, however, the condition code V bit will still set.

<04> TRACE ENABLE
When this bit is set at the beginning of an instruction, it will cause Trace Pending <30> to set. When Trace Pending is set at the end of an instruction, a trace fault is taken before the execution of the next instruction. When TP is clear, no trace exception occurs.

<03:00> CONDITION CODES: N, Z, V, C

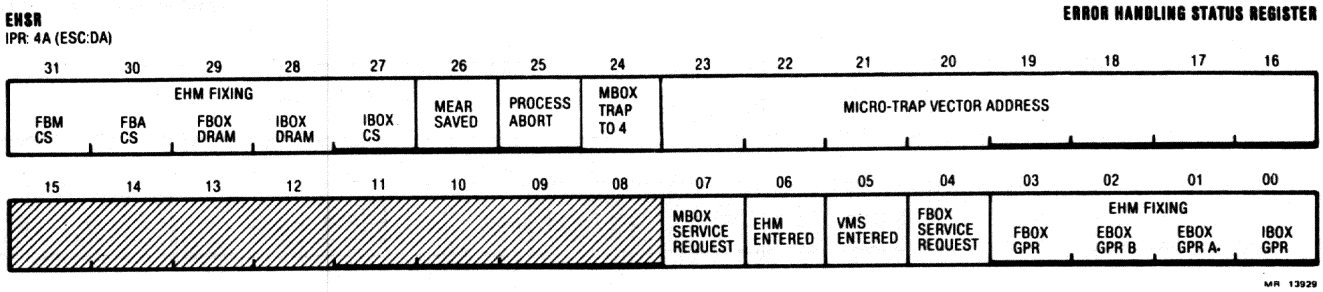
N BIT - When set, the N (negative) condition code bit indicates that the last instruction that affected this bit produced a negative result. If this bit is clear, the result was positive or zero.



Z BIT - When set, the Z (zero) condition code indicates that the last instruction which affected this bit produced a result which was zero. When this bit is clear, the result was non-zero.

V BIT - When set, the V (overflow) condition code bit indicates that the last instruction which affected this bit either had a conversion error or produced a result whose magnitude was too large to be properly represented in the operand which received the result. When this bit is clear, there was no conversion error or overflow.

C BIT - When set, the C (carry) condition code bit indicates that the last instruction which affected this bit either had a carry out of the most significant bit of the result or a borrow into the most significant bit. When this bit is clear, there was no carry or borrow.



<31:27> EHM IS FIXING
The EHM sets the appropriate bit in this field just before it sets the corresponding bit in EBCS <31:27>. Setting a bit in EBCS <31:27> causes the EBox to interrupt the Console for Control Store or Dispatch RAM correction.

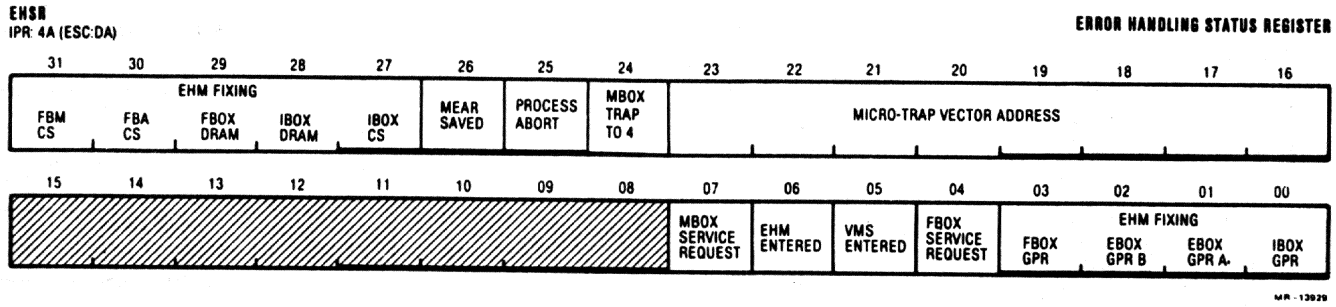
<26> MEAR SAVED
When the EHM is called to handle an MBox Error Register Full (ERF) micro trap, it saves MEAR in ESC: DB and sets this bit. Later, when the EHM is servicing the MBox interrupt associated with the ERF micro trap, it checks this bit to determine whether it should get MEAR from the MBox or ESC: DB.

<25> PROCESS ABORT
The EHM sets this bit if it determines that: 1) An MBox CPR Parity Error failed to result in an MBox Fatal Error, or 2) The EBox failed to detect bad data on the OPBus. If this bit is set, VMS will either Bugcheck the user or the system.

<24> MBOX TRAP TO 4
This bit indicates the occurrence of an MBox trap to 4. When running with the the IPL above 1D this may be the only sign of an SBE. It is used by VMB to check for single bit errors.

<23:16> MICRO-TRAP VECTOR ADDRESS
The EHM saves the entry level trap vector address in this field. The vector addresses are:

VECTOR	REASON
2	FBox error (Called by FBox interrupt handler)
4	EHM detected a process abort condition during a MBox ERF micro-trap
6	MBox error (Called by MBox interrupt handler)
8	EBox error
8	MBox fatal error
8	TB parity error (Ebox port request only)
10	EBox Op-Port-Write and IBox error
10	IBox Op-Port-Write and TB parity error
10	EBox IMD read and IBox error
10	EBox IMD read and TB parity error



VECTOR REASON (Cont)

18 EBox fork and IBox error
18 EBox fork and TB parity error
18 EBox ID read and IBox error
18 EBox string read and IBox error
18 EBox string read and TB parity error
1E EBox sync failure
1F RLog unwind parity error

<15:08> **RESERVED**

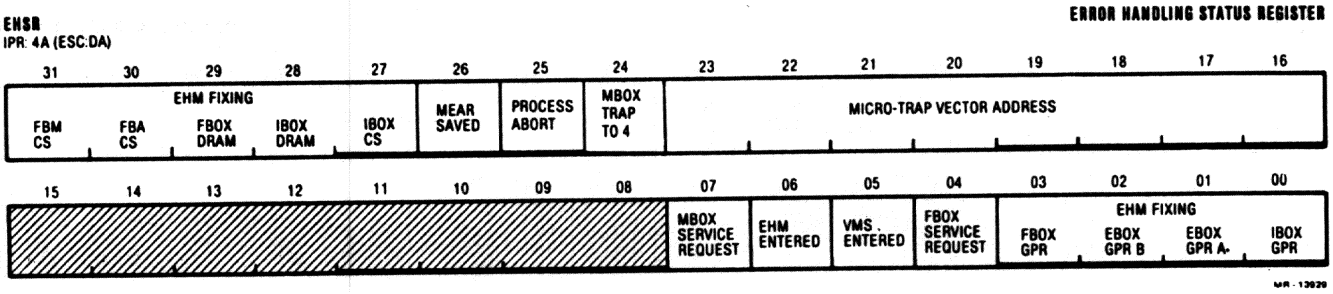
<07> **MBOX SERVICE REQUEST**
If the interrupt handling microcode determines that it was called to handle an MBox error interrupt, it will set this flag and call the EHM. The EHM will test this flag and, if set will process the MBox error.

<06> **EHM ENTERED**
This flag is used by the EHM to detect a double error trap condition. That is, those cases when a second error trap occurs before the EHM routine is able to finish processing the first error and pass control to VMS.

This flag is checked each time EHM is entered. If the flag is clear (which is the expected case), then EHM will set this flag and process the error in the normal manner. If the flag is set, however, indicating that EHM was in the process of handling an error when it was called to handle a second error, then one of two things will happen:

1. If the second error was detected by either the EBox or the MBox fatal error detection circuitry, then EHM will loop at UPC 21. This in turn will cause a Keep Alive Fail condition and the Console will print the following message and capture (Snap Shot) the state of the system:

" Attempting to save machine state due to"
"MACHINE DOUBLE ERROR"



2. If the second error was detected by the IBox then EHM will put a code of 5 in CSM.STATUS (ESC: C0) and call the CSM.ENTRY.DE routine. This will result in a Keep Alive Fail Condition and the Console will print the following message and Snap Shot the state of the system:

" Attempting to save machine state due to"
"CPU ERROR HALT"

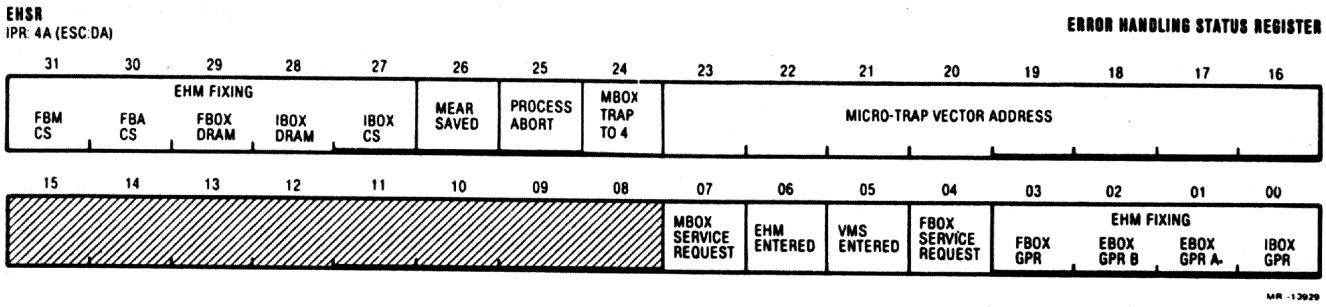
NOTE

When EHM finishes processing the error, it will set the VMS ENTERED flag, clear this flag and call the VMS Machine Check Handler.

<05> **VMS ENTERED**
This flag is similar to the EHM ENTERED flag. It is used to detect the case where the VMS Machine Check Handler is in the process of handling an error when a second error is detected.

EHM sets this flag just before it calls the Machine Check Handler. The Machine Check Handler processes the errors and clears this flag just before it executes an REI to continue the operation (or a BUGCHECK to halt the operation).

If a second error trap occurs while the Machine Check Handler is still processing the first error, then the EHM routine will process the error in the normal manner. That is, build a Stack Frame, clear the error condition, roll back the PC's and determine if it should call VMS.



However, since the VMS ENTERED flag is set (indicating that VMS was processing an error when a second error was detected), EHM will not call VMS. Instead, it will put a code of 5 in CSM.STATUS (ESC: C0) and call the CSM.ENTRY.DE routine. This in turn will result in a Keep Alive Fail condition and the Console will print the following message and capture (Snap Shot) the state of the system:

" Attempting to save machine state due to"
"CPU ERROR HALT"

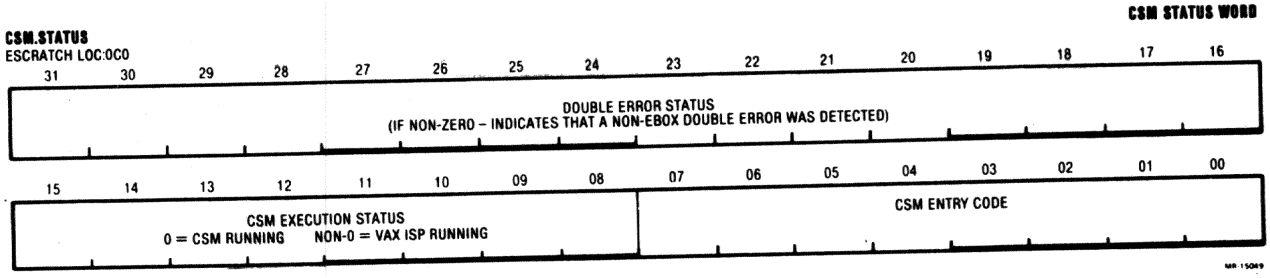
<04> **FBOX SERVICE REQUEST**
The micro-routine that handles FBox problems will set this flag if it determines that the problem was caused by an FBox hadrware error. The routine will then call the EHM routine to process the error. The EHM will test this flag to determine if it was called to handle an FBox error.

<03> **FIX FBOX GPR PE**
Set by the EHM when it attempts to correct an FBox GPR parity error.

<02> **FIX EBOX GPRB PE**
Set by the EHM when it attempts to correct an EBox GPR B parity error.

<01> **FIX EBOX GPRA PE**
Set by the EHM when it attempts to correct an EBox GPR A parity error.

<00> **FIX IBOX GPR PE**
Set by the EHM when it attempts to correct an IBox GPR parity error.



CSM.STATUS (CSM Status Register)

This is the status word used to control the Console Support Microcode (CSM). It is located in ESC C0.

<31:16> **DOUBLE ERROR STATUS**
This field is normally equal to zero. If this field is non-zero then it indicates that a non-EBox double error condition has occurred.

<15:08> **EXECUTION STATUS**

If this field is equal to zero it indicates that CSM is executing in the EBox. If this field is not equal to zero (i.e., non-zero) then it indicates that the VAX ISP microcode is executing in the EBox.

<07:00> **ENTRY CODE**
This field contains a code that indicates reason why CSM was started.

Code Reason

00 CSM is not executing.
04 Interrupt Stack not Valid (Software Error).
05 A non-EBox double error was detected.
06 Halt Instruction was decoded in Kernel Mode.
07 SCB Vector Code <1:0> = 3 (Software Error).
08 SCB Vector Code <1:0> = 2 (no WCS Microcode).
09 Pending Error on Halt.
0A CHMx Instruction decoded and Interrupt Stack = 1.
0B CHMx Instruction decoded and Vector <1:0> ≠ 0.
10 Micro-break was encounterd which caused the console to start CSM at CSM.ENTRY.MICRO:.
11 Console Halt Pending was set which caused CSM to start running on AFork at A.CSM.ENTRY.MICRO:.
15 The CPU was powered up and the Console started CSM at CSM.ENTRY.PO:.
16 During the power up sequence this code is used by the FIND 64KB and FIND RPB procedures to interface with CSM.ERROR: routine. The Console should never see this code. If it does then there something very wrong in the CPU.

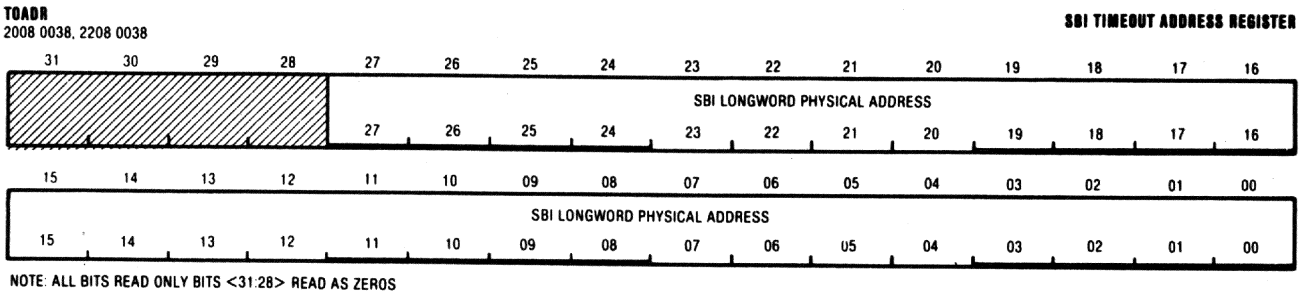
APPENDIX L

SBIA STACK FRAME BIT DESCRIPTIONS

The VMS machine check handler will build an I/O adapter (IOA) stack frame (entry code 13) for SBI ALERT, SBI FAULT, and SBI ERROR interrupts. In addition, the VMS machine check handler will build an IOA stack frame if it was called to handle a machine check (entry code 4) and it determines that the machine check is related to an I/O error (i.e., CP IO BUF ERROR was set during a CP read operation).

This appendix contains bit and field descriptions for the major control and status registers (listed below) that make up an IOA stack frame.

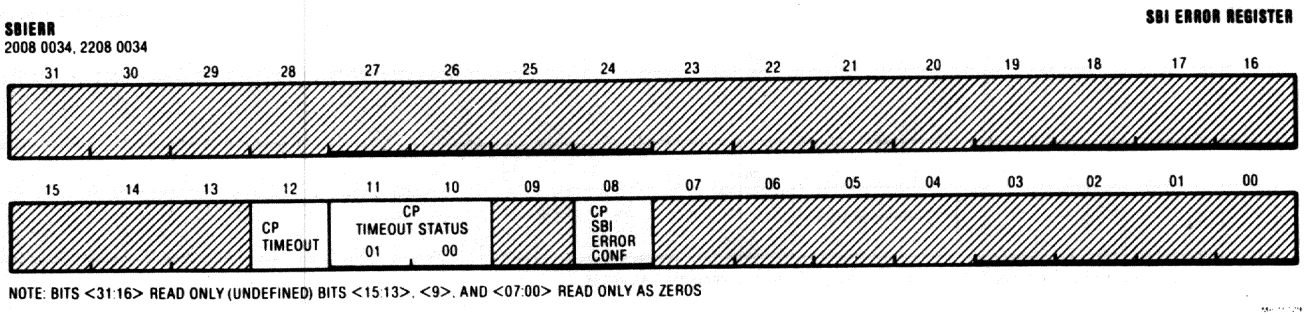
SBIA STACK FRAME BIT DESCRIPTIONS
SBIA SBI TIMEOUT ADDRESS



The SBI timeout address register holds the address for all CPU transactions in the SBIA. When an error occurs on a CPU transaction, the timeout address register is locked.

<31:28> MBZ (SS36)
These bits are forced to zero by the zero fill logic.

<27:00> SBI LONGWORD PHYSICAL ADDRESS - BUS REG D <27:00> (SS27)
The Timeout Address Register is loaded with the physical address, for the CPU command, each time a command/address is transferred from the file data latch to the command/address latch. It will be locked if Error Summary Register bit <23> is set.



NOTE: BITS <31:16> READ ONLY (UNDEFINED) BITS <15:13>, <9>, AND <07:00> READ ONLY AS ZEROS

The SBI error register stores information about CPU transactions which failed on the SBI due to timeout or error confirmation.

<31:16> ZERO (SS36)
These bits are read as zeros provided by the zero fill logic.

<15:13> ZERO (SS32)
These read only bits are forced to zero by hardware ground potential.

<12> CP TIMEOUT
BUS REG D<12> (SS32)
This bit will be set when there is an SBI timeout on a CPU reference for one of the following reasons:

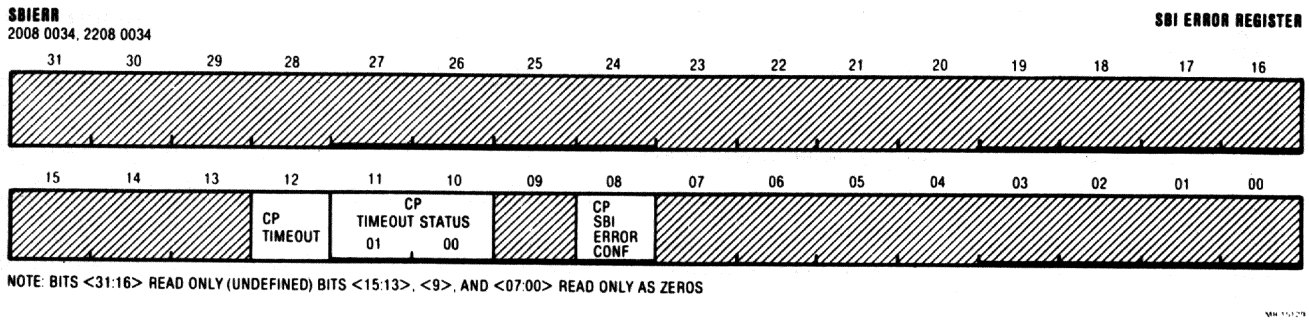
1. Unsuccessful access: When the SBIA does not receive an acknowledge confirmation for a CPU command/address or write data within 512 SBI cycles (102.4 microseconds) from the time the SBIA first requests the SBI. Unsuccessful access can be caused by the following:

   a. SBIA is unable to win the SBI through bus arbitration
   b. Target NEXUS is always busy when accessed
   c. The address is for a non-existent device or address
   d. Combinations of the first two

2. If the SBIA does not receive the read data within 512 SBI cycles of the acknowledge for the command/address, it is a read data timeout.

   When this bit sets, Error Summary Register <23> will be set, locking Error Summary Register <31:26> (type of reference), SBI Error Register <11:10, 08>, and the Timeout Address Register (referenced address). This bit is reset when the CPU writes it to a one. This will also reset <11:10, 08>.

SBIA STACK FRAME BIT DESCRIPTIONS
SBIA SBI ERROR REGISTER



<11:10> CP TIMEOUT STATUS <01:00>
BUS REG D <11:10> (SS32)
The timeout status bits are made up of two signals as follows:

1. Bit <01>: State machine is in the read pending state.
2. Bit <00>: SBI confirmation equals 01, busy.

Together these two signals indicate the type of SBI timeout.

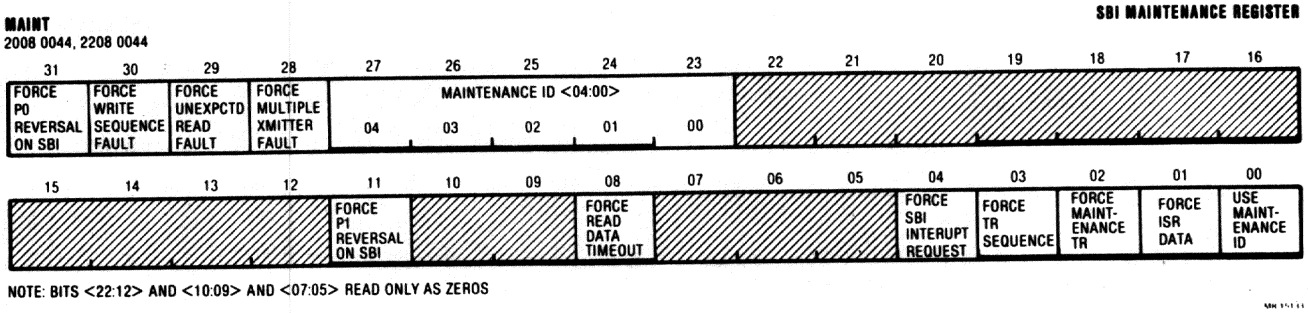
a. 00: SBI NEXUS did not respond (No Response)
b. 01: Device was busy (Busy)
c. 10: Waiting for read data
d. 11: Cannot happen

These bits are locked by Error Summary Register <23>, and reset when the CPU writes SBI Error Register <12>.

<09> ZERO (SS32)
These read only bits are forced to zero by hardware ground potential.

<08> CPU SBI ERROR CONFIRMATION
BUS REG D <08> (SS32)
This bit is set when the SBI state machine enters the error abort state if the SBI NEXUS has returned an error confirmation on a CPU read/write command/address cycle. If this bit is set, Error Summary Register <23> will be set, locking the Timeout Address Register, Error Summary Register <31:26>, and bits <12:10> of this register. This bit is reset when the CPU writes bit <12>.

<07:00> ZERO (SS32)
These read only bits are forced to zero by hardware ground potential.



NOTE: BITS <22:12> AND <10:09> AND <07:05> READ ONLY AS ZEROS

This register is used as a diagnostic and maintenance tool. Operational software does not use this register.

<31> FORCE PO REVERSAL
SS24 FRC PO REV ON SBI
When set, this bit will cause the SBIA to transmit bad PO parity on the SBI for all SBIA to SBI transactions. This includes CPU read/write and DMA read data.

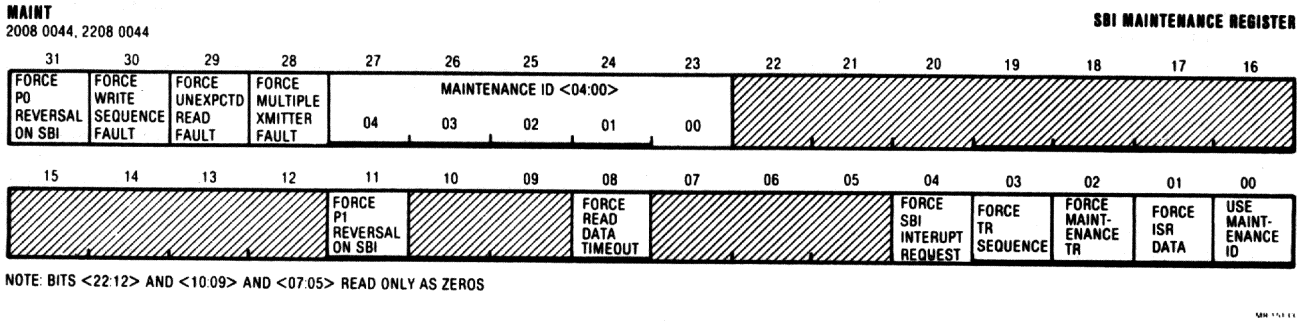
<30> FORCE WRITE SEQUENCE FAULT
SS24 FORCE WSQ FAULT
When set, this bit will force SBI TAG <01> to a logic 1. When used with a CPU write to an SBI nexus register, it will force the write data tag to 111, the diagnostic tag. This will cause a write sequence fault because SBI devices are looking for a tag of 101, write data.

<29> FORCE UNEXPECTED READ FAULT
SS24 FORCE UNEXP READ
When this bit is set, the maintenance ID, bits <27:23>, with a TAG of zero, will be repeatedly transmitted on the SBI (the data is undefined). When the nexus, as selected by the maintenance ID, receives read data (TAG = 0), it should assert BUS SBIT FAULT because of the unexpected read data.

<28> FORCE MULTIPLE TRANSMITTER FAULT
SS24 FORCE MULTI
Setting this bit forces a multiple transmitter fault in any selected nexus. The CPU will load the maintenance ID with the ID of the selected nexus, then read that nexus configuration register. On the cycle after the command/address is transmitted on the SBI, the SBIA will enable the SBI to continually transmit a TAG = 111 with the maintenance ID (data is undefined).

When the nexus transmits the read data, the ID transmitted by the nexus is the SBIA's ID. It will be ORed with the maintenance ID, and as long as the bits are not masked, will cause the nexus to detect a multiple transmitter fault.

SBIA STACK FRAME BIT DESCRIPTIONS
SBIA SBI MAINT REG



NOTE: BITS <22:12> AND <10:09> AND <07:05> READ ONLY AS ZEROS

<27:23> MAINTENANCE ID <04:00>
SS24 MAINT ID <04:00>
These bits are used to generate the maintenance ID in the following instances:

1. Generation of unexpected read fault
2. Generation of multiple transmitter fault
3. Used by the silo as the compare ID
4. Used to check ID logic

<22:12> MBZ (SS33, SS36, SS32)
Must be zero.

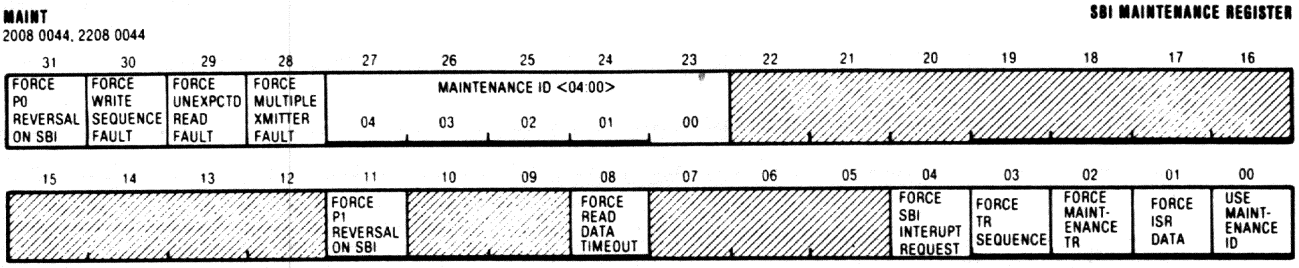
<11> FORCE P1 REVERSAL ON SBI
SS24 FRC P1 REV ON SBI
When set, this bit will cause the SBIA to transmit bad P1 parity on the SBI for all SBIA to SBI transactions. This includes CPU read/write and DMA read data.

<10:09> MBZ (SS32)
Must be zero.

<08> FORCE READ DATA TIMEOUT
SS24 FORCE TIMEOUT
This bit will preset the state machine timeout counter to all ones when the state machine enters the read wait start state. The timer will expire on the first count, generating a timeout condition while waiting for CPU read data.

<07:05> MBZ (SS32)
Must be zero.

<04> FORCE SBI INTERRUPT REQUEST
SS24 MAINT REQ ENA
When set, this bit will enable SBI Silo Comparator Register <07:04> and <03) to force interrupt requests and ALERT on the SBI.



<03> **FORCE TR SEQUENCE**
SS24 FORCE TR SEQ
This bit will enable SBI Silo Comparator Register <15:00> to assert TR <15:00> on the SBI when a CPU Command/Address is transmitted. It is used in conjunction with a loop back read to test the SBIA and SBI nexus arbitration logic.

<02> **FORCE MAINTENANCE TR**
SS24 FORCE MAINT TR
This bit will unconditionally assert the TR corresponding to SBI Silo Comparator Register <15:00>.

<01> **FORCE INTERRUPT SUMMARY READ DATA**
SS24 FORCE ISR DATA
This bit is used to enable the SBIA to respond to an interrupt summary read to check out the circuitry that prioritizes the ISR data and generates the vectors.

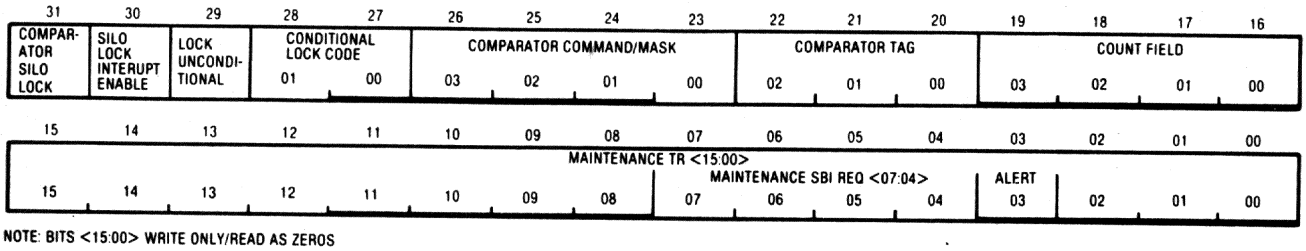
During the response cycle of an interrupt summary read, the SBIA will enable the write data latch to be transmitted on the SBI along with a TAG, MASK, and ID of zero.

<00> **USE MAINTENANCE ID**
SS24 USE MAINT ID
This bit will enable the use of the maintenance ID, bits <27:23> for diagnostic purposes.

SBIA STACK FRAME BIT DESCRIPTIONS
SBIA SBI SILO COMPARE

SILCOMP
2008 0040, 2208 0040

SBI SILO COMPARE REGISTER



NOTE: BITS <15:00> WRITE ONLY/READ AS ZEROS

<31> **COMPARATOR SILO LOCK**
SS21 CMP SILO LOCK
The Comparator Silo Lock bit is set if the count in the silo counter has reached F. When this bit sets, the CPU will be interrupted if bit <30> is set. This bit is cleared when the CPU loads the silo count field with a count other than F.

<30> **SILO LOCK INTERRUPT ENABLE**
SS25 SILO LOCK INTR EN
The CPU sets this bit to enable an interrupt when bit <31>, CMP SILO LOCK, sets.

<29> **LOCK UNCONDITIONAL**
SS25 LOCK UNCOND
When this bit is set, the silo counter will count on each SBI cycle. It will cause a silo lock within 16 SBI cycles, depending upon the count that had been previously loaded into the silo count field.

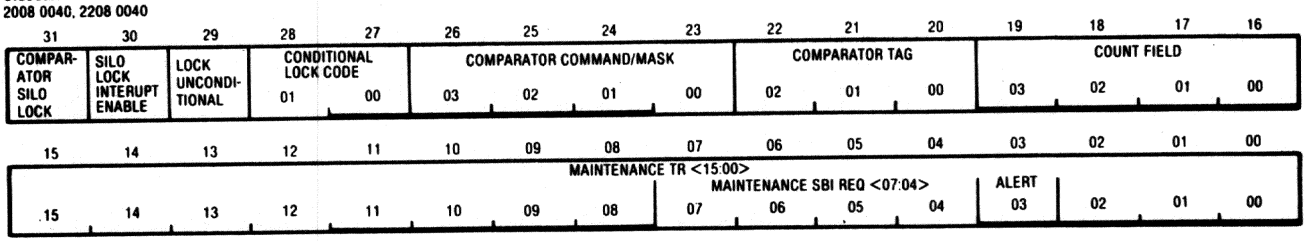
<28:27> **CONDITIONAL LOCK CODE <01:00>**
SS25 COND LOCK CODE <01:00>
These two bits determine the comparisons that will enable counting the silo counter to achieve a silo lock. If the SBI data matches the silo comparator bits, for the enabled comparison, the counter is incremented. The conditions are as follows:

1. 00: No compare (no comparison is made)
2. 01: SBI ID
3. 10: SBI ID and SBI TAG
4. 11: SBI ID and SBI TAG and SBI COMMAND/MASK

<26:23> **COMPARATOR COMMAND/MASK**
SS25 COMP CMD/MSK <03:00>
These bits provide the base for the silo comparison, when it is enabled to compare the command/mask. If the SBI tag is 011, command/address, then this field is compared with SBI B <31:28>, the SBI function. If the SBI tag is other than 011, this field is compared to the SBI mask bits.

SILCOMP
2008 0040, 2208 0040

SBI SILO COMPARE REGISTER



NOTE: BITS <15:00> WRITE ONLY/READ AS ZEROS

<22:20> COMPARATOR TAG
SS25 COMP TAG <02:00>
These bits provide the base for the silo comparison, when it is enabled to compare the tag. This field is compared with SBI TAG <02:00>.

<19:16> COUNT FIELD
SS19 COUNT FIELD
The CPU loads the silo counter with the two's complement of the number of SBI cycles to be loaded into the silo after a comparison is made. When the count reaches F, the silo is locked. The counter is also enabled by the Lock Unconditional bit <29>.

<15:00> MAINTENANCE TRANSFER REQUEST <15:00>
SS25 MAINT TR <15:00>
These bits provide the means to simulate SBI transfer requests, SBI interrupt requests, and SBI alert for diagnosing the interrupt logic and SBI priority arbitration logic. They also provide a means of testing the lower 16 bits of the silo. They are controlled by SBI Maintenance Register bits <04:02> as follows:

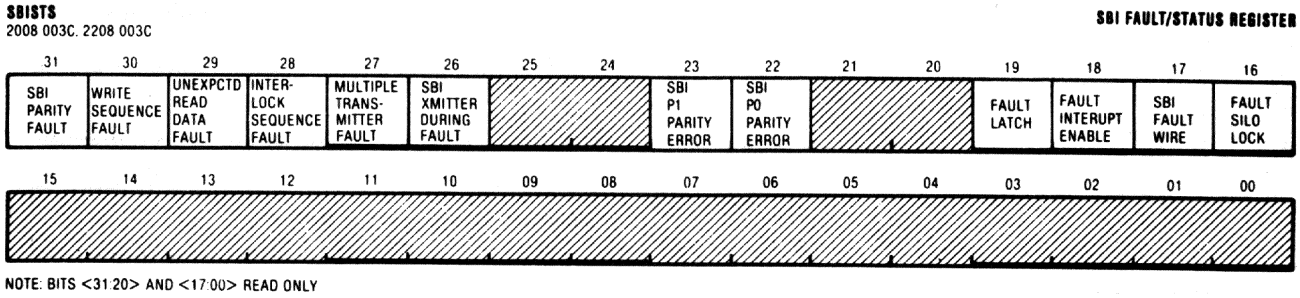
1. The asserted MAINT TR <07:04> bit will cause the corresponding SBI REQ <07:04> bit to be asserted if SBI Maintenance Register <04>, MAINT REQ ENA, is set.

2. If MAINT TR <03> and SBI Maintenance Register <04> are both set, SBI ALERT will be asserted.

3. If SBI Maintenance Register <03> is set, the asserted MAINT TR <15:00> will cause the corresponding SBI TR <15:00> to be asserted when a CPU command/address is transmitted on the SBI. See SBI Maintenance Register bit <03>.

4. MAINT TR <15:00> will cause the corresponding SBI TR <15:00> to be asserted if SBI Maintenance Register bit <02>, FORCE MAINT TR, is set.

SBIA STACK FRAME BIT DESCRIPTIONS
SBIA SBI FAULT STATUS REGISTER



NOTE: BITS <31:20> AND <17:00> READ ONLY

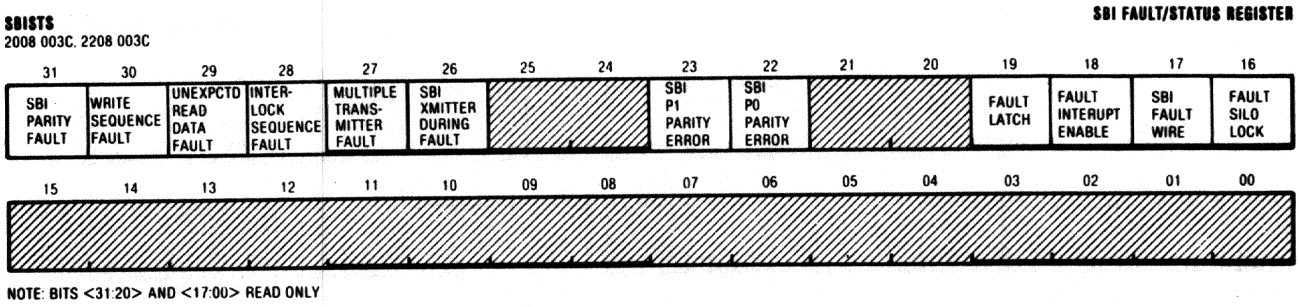
The SBI fault/status register saves information on SBI faults on transactions in which the SBIA may or may not have been involved. All SBI devices monitor the SBI every cycle and check for errors. If an error is detected, the Fault wire is asserted and a fault/status register (part of the configuration register for non-CPU devices) is locked.

<31> **SBI PARITY FAULT**
SS11 FAULT REG B <31>
This bit will set if the SBIA detects an SBI parity error on received SBI information. Bits <23:22> will indicate whether it was an address/data or control parity error. The register is written every SBI cycle, and locked when SBI FAULT is asserted. This bit will be cleared when SBI FAULT is deasserted. This bit is only valid if bit <19> is set.

<30> **WRITE SEQUENCE FAULT**
SS11 FAULT REG B <30>
This bit is set if the SBIA is expecting write data, and receives SBI information, with no parity error, but the tag does not indicate write data (101). This bit is also locked when SBI FAULT is asserted, and clears when SBI FAULT clears. This bit is only valid if bit <19> is set.

<29> **UNEXPECTED READ DATA**
SS11 FAULT REG B <29>
This bit sets if the SBIA receives information with the SBIA ID (10000) with a read data tag (000), but the SBIA is not expecting read data (no read pending). This bit is locked when SBI FAULT is asserted, and clears when SBI FAULT clears. This bit is only valid if bit <19> is set.

<28> **INTERLOCK SEQUENCE FAULT**
SS11 FAULT REG B <28>
This bit is set if the SBIA receives a valid command/address for an interlock write masked but the interlock flip-flop is not set (an interlock read has not occurred). This bit is also locked when SBI FAULT is asserted, and clears when SBI FAULT clears. This bit is only valid if bit <19> is set.



NOTE: BITS <31:20> AND <17:00> READ ONLY

<27> **MULTIPLE TRANSMITTER FAULT**
SS11 FAULT REG B <27>
This bit will set if the SBIA detects an ID that is not the same as the ID it transmitted on the SBI. This bit is also locked when SBI FAULT is asserted, and clears when SBI FAULT clears. This bit is only valid if bit <19> is set.

<26> **SBI TRANSMITTER DURING FAULT**
SS11 FAULT REG B <26>
This bit sets when the SBIA was the nexus transmitting on the SBI. This bit is locked when SBI FAULT is asserted, and clears when SBI FAULT clears. This bit is only valid if bit <19> is set.

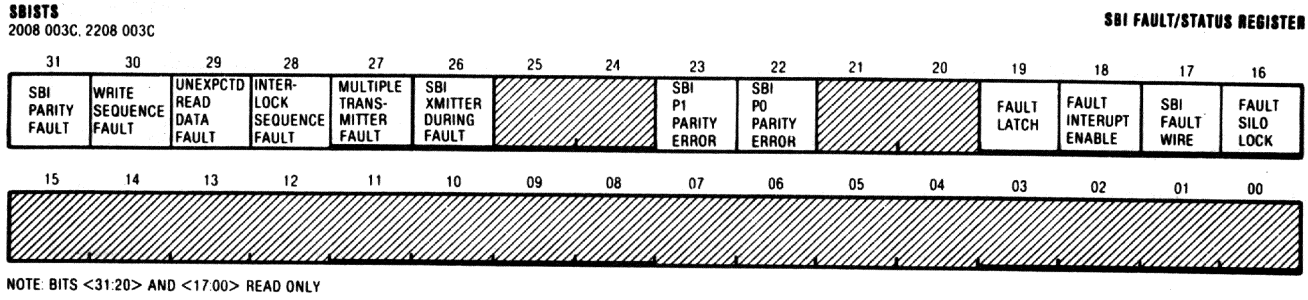
<25:24> **MBZ (SS33)**
Must be zero.

<23> **SBI P1 PARITY ERROR**
SS11 FAULT REG B <23>
This bit indicates that an SBI parity error was over SBI B <31:00>. It is only valid if bit <19> is set. It is locked when SBI FAULT is asserted, and will clear when SBI FAULT clears.

<22> **SBI P0 PARITY ERROR**
SS11 FAULT REG B <22>
This bit indicates that an SBI parity error was over SBI TAG <02:00>, SBI ID <04:00>, and SBI MASK <03:00>. It too, is only valid if bit <19> is set, and is locked when SBI FAULT is set. It will clear when SBI FAULT clears.

<21:20> **MBZ (SS33)**
Must be zero.

SBIA STACK FRAME BIT DESCRIPTIONS
SBIA SBI FAULT STATUS REGISTER



NOTE: BITS <31:20> AND <17:00> READ ONLY

<19> **FAULT LATCH**
BUS REG D <19> (SS33)
If an SBI nexus, including the SBIA, detects an SBI fault, the nexus will assert SBI FAULT. The SBIA, upon reception of SBI FAULT will set the fault latch, which will keep SBI FAULT asserted. It will remain asserted until the CPU clears the fault latch by writing a one to bit <19>. SBI FAULT will be asserted for the following SBI error conditions:

1. Interlock sequence fault
2. Unexpected read fault
3. Write sequence fault
4. Multiple transmitter fault
5. Parity fault

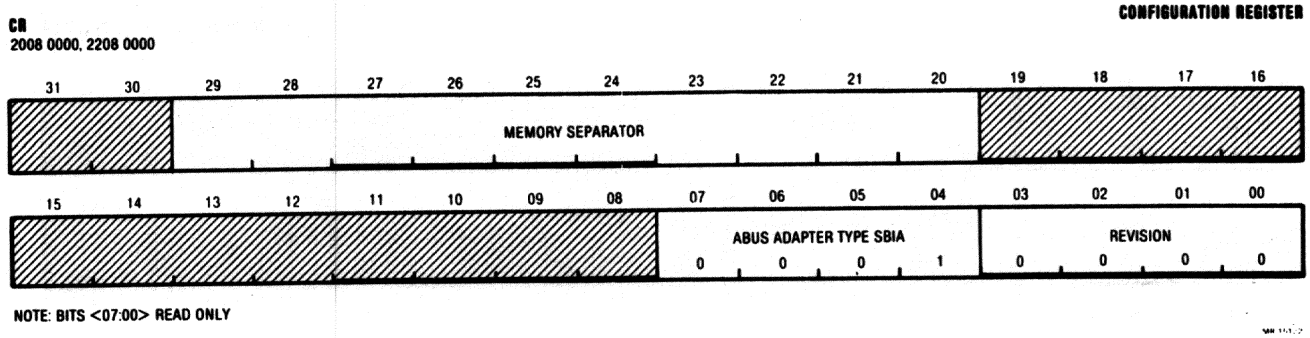
When this bit sets, Fault/Status Register <31:26> and <23:22> will be locked.

<18> **FAULT INTERRUPT ENABLE**
SS21 FAULT INTR ENA
The CPU will set this bit by writing bit <17> to enable an SBI fault to generate an interrupt. The interrupt will be asserted if the fault latch, bit <19>, is set.

<17> **SBI FAULT WIRE**
SS33 BUS REG D <17>
This bit indicates the state of the SBI FAULT signal.

<16> **FAULT SILO LOCK**
SS33 BUS REG D <16>
This bit will be set when the silo locks due to an SBI fault. It will be reset when the CPU resets the fault latch, bit <19>.

<15:00> **MBZ (SS36)**
Must be zero.



<31:30> **MBZ (SS28)**
Must be zero.

<29:20> **MEMORY SEPARATOR**
SS28 MSR <27:18>
This field defines the memory address boundry and is equal to the number of megabytes of memory addressable over the ABus. If bit 29 is asserted, there are 512 MBytes of memory and bits <28:20> are disregarded when the hardware checks the DMA address. These bits are bits <29:20> in the Memory Separator register, but within the SBIA they are shifted right by two bits to match the physical address.

<19:08> **MBZ (SS36)**
Must be zero.

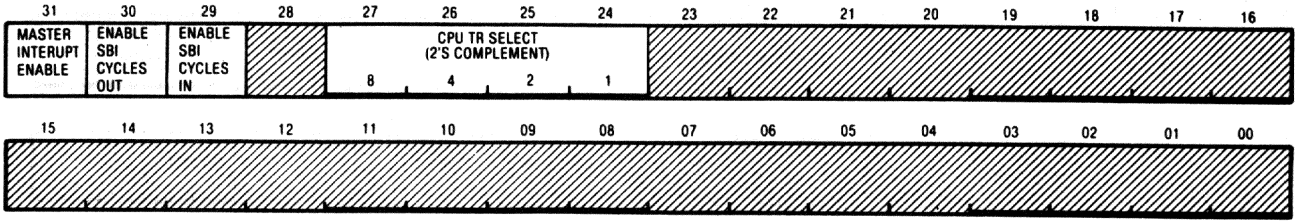
<07:04> **ABUS ADAPTER TYPE**
BUS REG D<07:04> (SS28)
These bits identify the type of ABus adapter, 0001 for the SBIA.

<03:00> **ABUS ADAPTER REVISION**
BUS REG D<03:00> (SS28)
These bits identify the revision of the ABus adapter; these bits are hardwired.

SBIA STACK FRAME BIT DESCRIPTIONS
SBIA CONTROL STATUS REGISTER

CSR
2008 0004, 2208 0004

CONTROL AND STATUS REGISTER



NOTE: BITS <28:00> READ ONLY AS ZEROS

<31> **MASTER INTERRUPT ENABLE**
SS29 MSTR INTR ENA
When set, this bit will enable the SBA module to prioritize the interrupts and generate the appropriate interrupt priority level for CPU polling.

<30> **ENA SBI CYCLES OUT**
SS29 ENA SBI OUT
This bit must be set for normal operation. It enables the CPU to access SBI NEXUS registers. If the CPU attempts to access an SBI NEXUS register with this bit reset, it is an error condition, and ERROR SUMMARY register <20> and <19> will be set. See description of ERROR SUMMARY register.

<29> **ENA SBI CYCLES IN**
SS29 ENA SBI IN
This bit must also be set for normal operation. It enables all DMA activity through the SBIA. If this bit is not set, the SBIA will not recognize SBI function codes and will not respond to SBI commands (SBI confirmation is 00, no response).

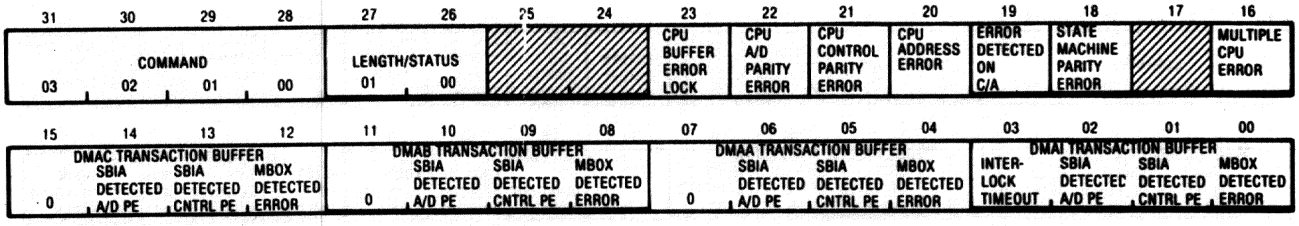
<28> **ZERO (SS33)**
This read only bit will always be zero.

<27:24> **CPU TR SELECT <08:04>**
CPU TR SEL <08:04> (SS07)
These bits provide backplane visibility of the jumpers used to select the SBI TR for CPU transactions. The contents of this field is the two's complement of the TR level.

<23:00> **ZERO (SS36)**
These bits are always read as zero provided by the zero fill logic.

ERRSUM
2008 0008, 2208 0008

ERROR SUMMARY REGISTER



NOTE: BITS <31:24> AND <22:19> AND <17:16> READ ONLY AS ZEROS

MR-15174

The error summary register contains most of the information about errors which have been detected by the SBIA on transactions involving the SBIA.

<31:28> COMMAND <03:00>
BUS REG D<31:28> (SS26)
These bits represent the ABus command bits for a CPU I/O register read/write. They are loaded each time the command/address latch is loaded, and are latched by CPU ERROR LOCK, bit <23>.

<27:26> LENGTH/STATUS <01:00>
BUS REG <27:26> (SS26)
These bits represent the ABus data length for a CPU I/O register read/write. They are also loaded each time the command/address latch is loaded, and are latched by CPU ERROR LOCK, bit <23>.

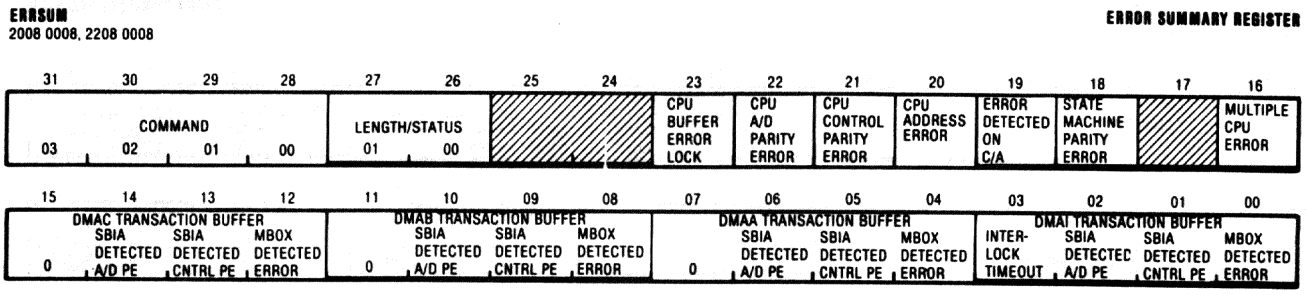
<25:24> MBZ
Must be zero.

<23> CPU BUFFER ERROR LOCK
SS37 CPU ERROR LOCK
This bit will be asserted for any of the following errors on a CPU I/O read/write.

1. A/D Parity Error (bit <22>)
2. Control Parity Error (bit <21>)
3. Address Error (bit <20>)
4. CPU read/write timeout on SBI (SBI Error Register bit <12>)
5. SBI Error (SBI Error Register <08>)

If this bit is set, Error Summary Register <31:26> and the Timeout Address Register are latched. If clear, these bits will represent the most recent transaction. Writing this bit will clear Error Summary Register <22:19,16>.

SBIA STACK FRAME BIT DESCRIPTIONS
SBIA ERROR SUMMARY



NOTE: BITS <31:24> AND <22:19> AND <17:16> READ ONLY AS ZEROS

MR-15124

<22> CPU A/D PARITY ERROR
SBAN A/D PTY BAD
This bit is set if a parity error is detected on the address/data bits of the command/address or write data for a CPU I/O read/write. If the error is detected on the command/address, bit <19> will also be set. Parity is checked on the output of the file data latch. If this bit sets, bit <23> is set. This bit is cleared when the CPU writes bit <23>.

<21> CPU CONTROL PARITY ERROR
SBAN CNTRL PTY BAD
This bit is set if a parity error is detected on the control field of the command/address or write data for a CPU I/O read/write. If the error is detected on the command/address cycle, bit <19> will also be set. Parity is checked on the output of the file data latch. If this bit sets, bit <23> is also set. This bit is also cleared when the CPU writes bit <23>.

<20> CPU ADDRESS ERROR
SS38 LOCAL ADR ERR
This bit is set if the CPU accesses a nonexistent SBIA register or when an SBI NEXUS register is accessed when Control/ Status Register <30> is clear (CPU access to the SBI is disabled). When it sets, it will set bit <23>, and it is cleared when the CPU writes bit <23>. This error will be detected when the command/address word is available, so bit <19> should also be set.

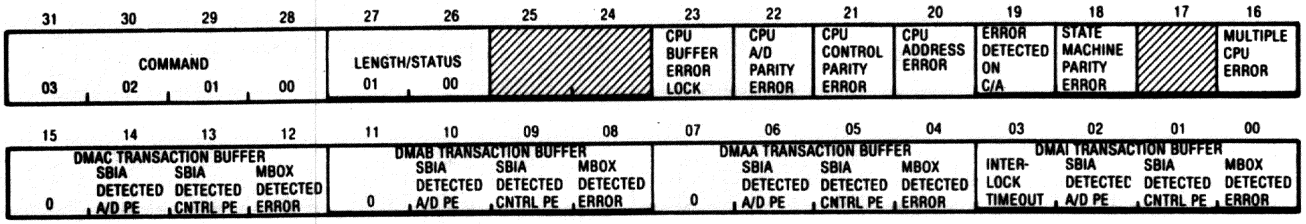
<19> ERROR DETECTED ON COMMAND/ADDRESS CYCLE
SBAN ERR ON C/A
This read only bit is set if a address/data, control parity, or address error is detected on the command/address cycle. The setting of this bit will set bit <23>. This bit will be reset when the CPU writes a one to bit <23>.

<18> STATE MACHINE PARITY ERROR
SBAO FORCE PARITY TRAP
This error bit will be set if the state machine microword does not contain even parity. The occurrence of this error will cause a CPU transaction to be aborted, if one is in progress, and generate an interrupt. A state machine parity error can occur if no CPU transaction is in progress so it will not set bit <23>.

<17> MBZ (SS33)
Must be zero.

ERRSUM
2008 0008, 2208 0008

ERROR SUMMARY REGISTER



NOTE: BITS <31:24> AND <22:19> AND <17:16> READ ONLY AS ZEROS

MR-15124

<16> **MULTIPLE CPU ERROR**
SBAN MULT CPU ERR
This bit can only be set if bit <23> is already set and a CPU addressing error is detected on the command/address cycle or there is an address/data or control parity error on the command/address or write data. This bit will not be set for a write data parity error for the transaction that sets bit <23>, but for a subsequent transaction. Bit <16> is reset when the CPU writes bit <23>.

<15> **MBZ (SS32)**
Must be zero.

<14> **DMAC TRANSACTION BUFFER SBIA DETECTED A/D PARITY ERROR**
SS30 DMAC A/D ERROR
This bit will be set for a data parity error when the read data is being transferred from transaction buffer C to the SBI during a DMA read. This bit cannot be set if bits <13> or <12> have been previously set. This bit is cleared by the CPU writing it. The DMAC Command/Address Register and DMAC ID Register will be locked if this bit sets. The setting of this bit will generate a local interrupt.

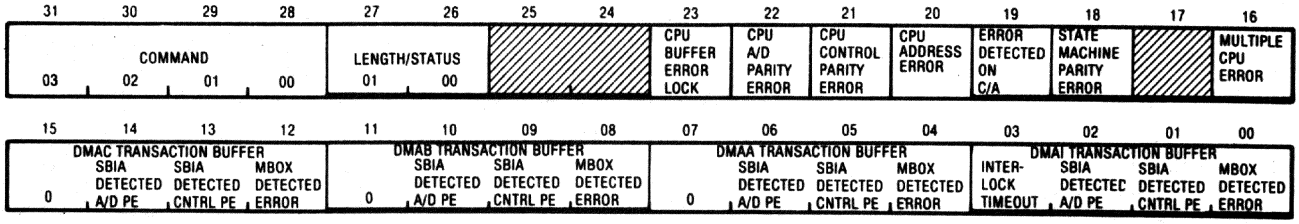
<13> **DMAC TRANSACTION BUFFER SBIA DETECTED CONTROL PARITY ERROR**
SS30 DMAC CONTROL ERROR
This bit will be set for a control parity error when the read data is being transferred from transaction buffer C to the SBI during a DMA read. This bit cannot be set if bits <14> or <12> have been previously set. This bit is cleared by the CPU writing it. The DMAC Command/Address Register and DMAC ID Register will be locked if this bit sets. The setting of this bit will generate a local interrupt.

<12> **DMAC TRANSACTION BUFFER MBOX DETECTED ERROR**
SS30 DMAC MBOX ERR
This bit will be set if the MBox detects a parity error or NXM on the transfer of command/address from the DMAC transaction buffer. This bit cannot be set if bits <14> or <13> have been previously set. This bit is cleared by the CPU writing it. The DMAC Command/Address Register and DMAC ID Register will be locked if this bit sets. The setting of this bit will generate a local interrupt.

<11> **MBZ**
Must be zero.

SBIA STACK FRAME BIT DESCRIPTIONS
SBIA ERROR SUMMARY

ERRSUM — **ERROR SUMMARY REGISTER**
2008 0008, 2208 0008



NOTE: BITS <31:24> AND <22:19> AND <17:16> READ ONLY AS ZEROS

MR-15124

<10> **DMAB TRANSACTION BUFFER SBIA DETECTED A/D PARITY ERROR**
SS30 DMAB A/D ERROR
This bit will be set for a data parity error when the read data is being transferred from transaction buffer B to the SBI during a DMA read. This bit cannot be set if bits <09> or <08> have been previously set. This bit is cleared by the CPU writing it. The DMAB Command/Address Register and DMAB ID Register will be locked if this bit sets. The setting of this bit will generate a local interrupt.

<09> **DMAB TRANSACTION BUFFER SBIA DETECTED CONTROL PARITY ERROR**
SS30 DMAB CONTROL ERROR
This bit is set for a control parity error when the read data is being transferred from transaction buffer B to the SBI during a DMA read. This bit cannot be set if bits <10> or <08> have been previously set. This bit is cleared by the CPU writing it. The DMAB Command/Address Register and DMAB ID Register will be locked if this bit sets. The setting of this bit will generate a local interrupt.

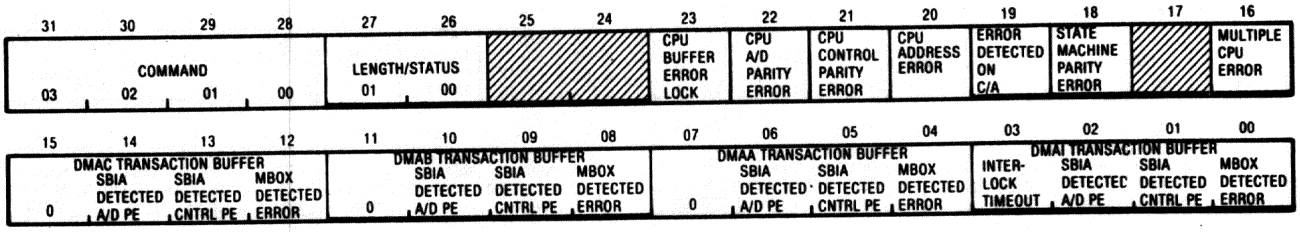
<08> **DMAB TRANSACTION BUFFER MBOX DETECTED ERROR**
SS30 DMAB MBOX ERR
This bit is set if the MBox detects a parity error or NXM on the transfer of command/address from the DMAB transaction buffer. This bit cannot be set if bits <10> or <09> have been previously set. This bit is cleared by the CPU writing it. The DMAB Command/Address Register and DMAB ID Register will be locked if this bit sets. The setting of this bit will generate a local interrupt.

<07> **MBZ**
Must be zero.

<06> **DMAA TRANSACTION BUFFER SBIA DETECTED A/D PARITY ERROR**
SS30 DMAA A/D ERROR
This bit is set for a data parity error when the read data is being transferred from transaction buffer A to the SBI during a DMA read. This bit cannot be set if bits <05> or <04> have been previously set. This bit is cleared by the CPU writing it. The DMAA Command/Address Register and DMAA ID Register will be locked if this bit sets. The setting of this bit will generate a local interrupt.

ERROR SUMMARY REGISTER

ERRSUM
2008 0008, 2208 0008



NOTE: BITS <31:24> AND <22:19> AND <17:16> READ ONLY AS ZEROS

MR-1512A

<05> **DMAA TRANSACTION BUFFER SBIA DETECTED CONTROL PARITY ERROR**
SS30 DMAA CCNTRL ERR
This bit is set for a control parity error when the read data is being transferred from transaction buffer A to the SBI during a DMA read. This bit cannot be set if bits <06> or <04> have been previously set. This bit is cleared by the CPU writing it. The DMAA Command/Address Register and DMAA ID Register will be locked if this bit sets. The setting of this bit will generate a local interrupt.

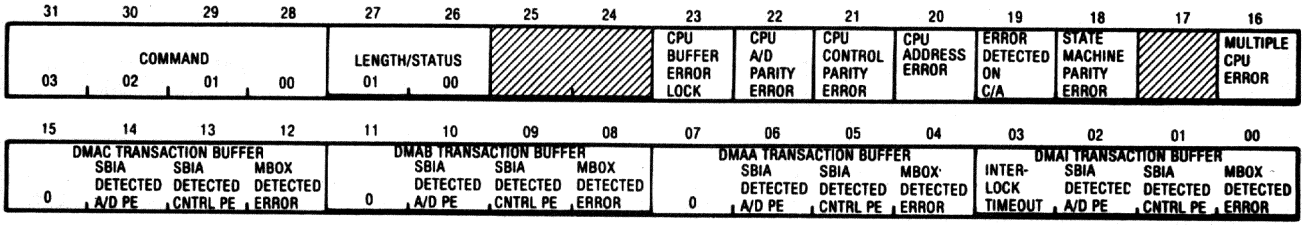
<04> **DMAA TRANSACTION BUFFER MBOX DETECTED ERROR**
SS30 DMAA MBOX ERR
This bit is set if the MBox detects a parity error or NXM on the transfer of command/address from the DMAA transaction buffer. This bit cannot be set if bits <06> or <05> have been previously set. This bit is cleared by the CPU writing it. The DMAA Command/Address Register and DMAA ID Register will be locked if this bit sets. The setting of this bit will generate a local interrupt.

<03> **DMAI TRANSACTION BUFFER INTERLOCK TIMEOUT**
SS30 DMAI TIMEOUT
This bit is set if an interlock write masked does not occur within 512 SBI cycles (102.4 microseconds) after an interlock read masked. This bit cannot be set if bits <02>, <01>, or <00> have been previously set. The setting of this bit will generate a local interrupt. This bit is cleared by the CPU writing it. The DMAI Command/Address and DMAI ID Register will be locked if this bit sets.

<02> **DMAI TRANSACTION BUFFER SBIA DETECTED A/D PARITY ERROR**
SS30 DMAI A/D ERROR
This bit is set for a data parity error when the read data is being transferred from transaction buffer I to the SBI during a DMA interlock read. This bit cannot be set if bits <03>, <01>, or <00> have been previously set. This bit is cleared by the CPU writing it. The DMAI Command/Address and DMAI ID Register will be locked if this bit sets. The setting of this bit will generate a local interrupt.

ERRSUM
2008 0008, 2208 0008

ERROR SUMMARY REGISTER

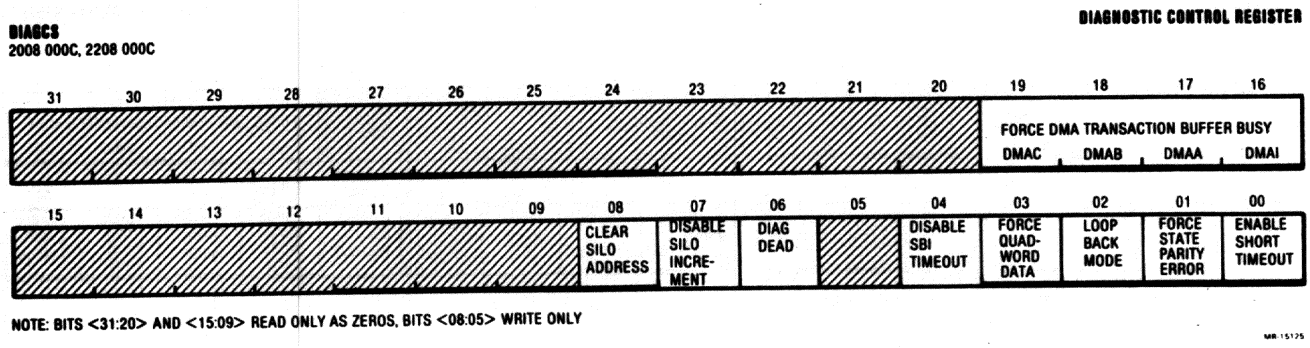


NOTE: BITS <31:24> AND <22:19> AND <17:16> READ ONLY AS ZEROS

MR-15124

<01> **DMAI TRANSACTION BUFFER SBIA DETECTED CONTROL PARITY ERROR**
SS30 DMAI CNTRL ERROR
This bit is set for a control parity error when the read data is being transferred from transaction buffer I to the SBI during a DMA interlock read. This bit cannot be set if bits <03:02> or <00> have been previously set. This bit is cleared by the CPU writing it. The DMAI Command/Address Register and DMAI ID Register will be locked if this bit sets. The setting of this bit will generate a local interrupt.

<00> **DMAI TRANSACTION BUFFER MBOX DETECTED ERROR**
SS30 DMAI MBOX ERR
This bit is set if the MBox detects a parity error or NXM on the transfer of command/address from the DMAI transaction buffer. This bit cannot be set if bits <03>, <02>, or <01> have been previously set. This bit is cleared by the CPU writing it. The DMAI Command/Address Register and DMAI ID Register will be locked if this bit sets. The setting of this bit will generate a local interrupt.



<31:20> **MBZ (SS36)**
Must be zero.

<19:16> **FORCE DMA TRANSACTION BUFFER BUSY**
SS29 FORCE DMAC (DMAB, DMAA, DMAI) BUSY
These bits are used to direct DMA traffic into specific DMA transaction buffers by forcing other buffers to be busy. The state of these bits has no effect on a DMA transaction already in progress.

<15:09> **MBZ (SS32)**
Must be zero.

<08> **CLEAR SILO ADDRESS**
SS19 CLR SILO ADR
This bit will clear the silo address upon setting. When this register is read, this bit will always be zero (hardwired).

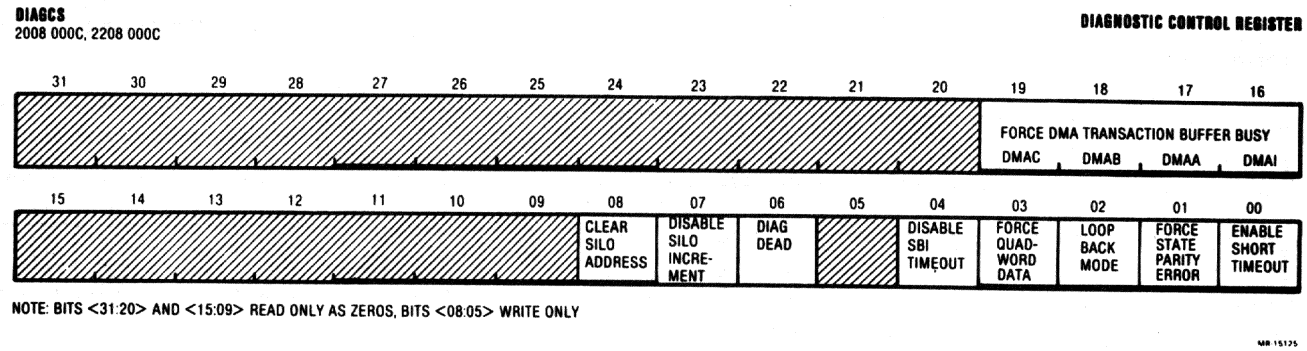
<07> **DISABLE SILO INCREMENT**
SS29 DISABLE SILO INC
When set, this bit will prevent the silo address from incrementing. This bit will be reset during normal operations to allow the silo address to increment. This bit is also read as zero.

<06> **DIAGNOSTIC DEAD**
SS29 DIAG DEAD
When this bit sets, it will simulate ABUS DEAD, interrupting the console and causing a reboot. ABUS DEAD is normally asserted by SBI FAIL. This bit is also read as zero.

<05> **MBZ (SS30)**
Must be zero.

<04> **DISABLE SBI TIMEOUT**
SS29 DISABLE SBI TMO
When set for diagnostics, this bit will prevent a timeout condition while waiting for the SBIA to gain control of the SBI, while waiting an acknowledgment from a NEXUS or while waiting for CPU read data.

SBIA STACK FRAME BIT DESCRIPTIONS
SBIA DIAGNOSTIC CONTROL REGISTER



NOTE: BITS <31:20> AND <15:09> READ ONLY AS ZEROS, BITS <08:05> WRITE ONLY

MR-15125

<03> FORCE QUADWORD DATA
SS29 FORCE QUAD DATA
This bit is used by microdiagnostics with bit <02> (Loop Back Mode), to provide a way to use a quadclear to loop data back on the SBI. FORCE QUAD DATA is set, then the CPU will execute a quadclear, but for microdiagnostics, the address will be a memory address instead of an SBI address (bit 27 is clear).

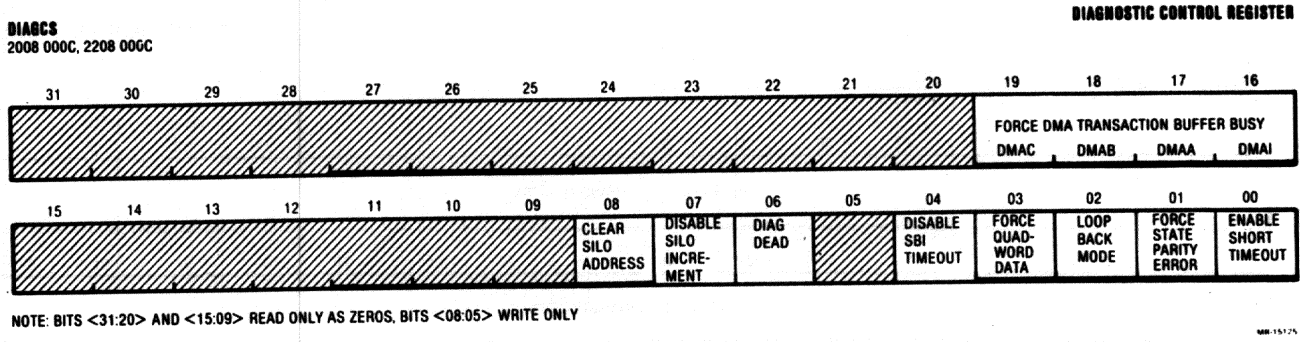
The ABus command/address is the same; it specifies a CPU write to the quadclear register. The ABus write data is the same except for the address, which will be for a memory (cache) address. When the command/address is transmitted on the SBI it will be received by the SBIA, as it always is, but in this case, the address will be less than the configuration register. To the SBIA it looks like a DMA extended write mask to memory and it will be handled as such.

For a normal quadclear, the write data is forced to all zeros. In this case, FORCE QUAD DATA will enable the A-Data Assembly Multiplexers to transfer the contents of the Write Data Latch (the ABus Write Data, 1011 and the Quadword Boundary Address) to the SBI. When the second write data longword is transferred to the SBI transceivers, bits 30 and 27 will be toggled (set). This will allow the setting of all data bits on the SBI.

<02> LOOP BACK MODE
SS29 LOOP BACK MODE
This bit is used by microdiagnostics to allow a CPU read or write to be looped back in the SBIA. The PAMM has to be configured such that a memory (cache) address is mapped to an I/O adapter, and the same PAMM address, but with bits 27 and 28 inverted, is mapped to a memory address.

In the SBIA, LOOP BACK MODE will invert address bits 25 and 26, if bit 27 is reset, which will be the case if the CPU write is to a memory address.

When the CPU writes a memory location that is mapped to an I/O adapter, the MBox will write the command/address and write data longword into the register file. The SBIA will carry out the command like a normal CPU write. When the command/address is transferred from the command/address latch to the SBI, because LOOP BACK MODE is set and address bit 27 is reset, address bits 25 and 26 will be inverted.



The command/address, followed by the write data will be transmitted on the SBI. When the SBIA clocks the SBI receivers and looks at the received data, the address is less than the memory separator (in the configuration register), it will transfer the command/address and write data to the register file and request MBox service.

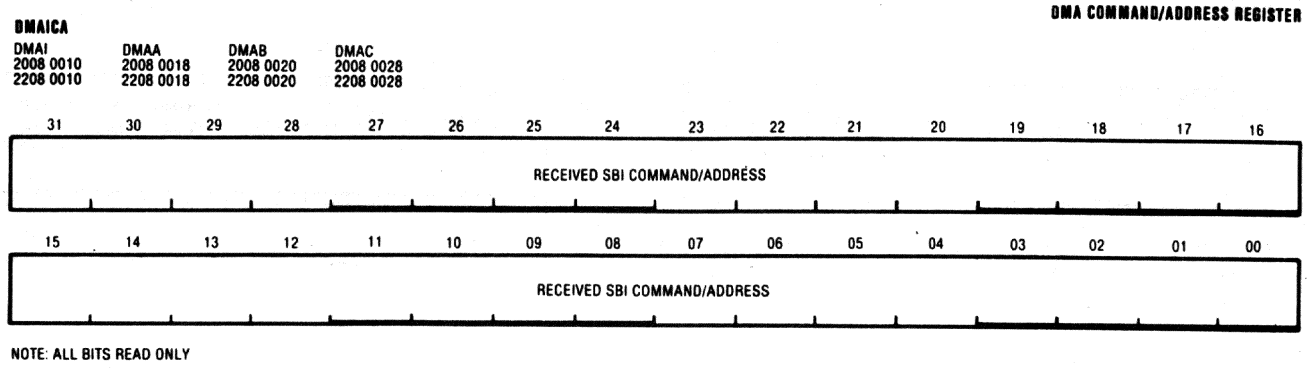
The MBox will write the data into memory because the address, with bits 27 and 28 inverted (bits 25 and 26 in the SBIA), addresses a different PAMM location. This location is mapped to memory.

This diagnostic bit can also be used with a CPU read in a similar manner to further check out the SBIA logic.

<01> **FORCE STATE PARITY ERROR**
SS29 FORCE STATE PTY
If this bit is set, a state machine parity error will be forced during the CPU ARB WAIT state.

<00> **ENABLE SHORT TIMEOUT**
SS29 ENA SHORT TIMEOUT
When set, this bit will enable an SBI timeout in 8 SBI cycles instead of the normal 512 cycles. Read as zero.

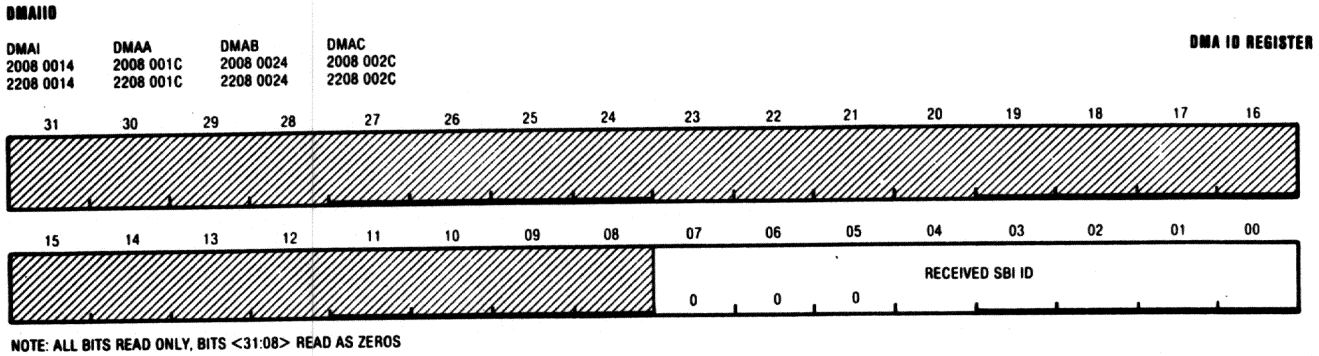
SBIA STACK FRAME BIT DESCRIPTIONS
SBIA DMA COMMAND/ADDRESS REGISTER



NOTE: ALL BITS READ ONLY

These four registers (DMAACA, DMABCA, DMACCA, and DMAICA) are used to save information about transactions in the DMA buffers. Each time a command/address is loaded into a DMA buffer, a copy of the command/address and SBI ID of the commander is saved in the two registers corresponding to the DMA buffer. If an error is dectected by the MBox or the SBIA after the transaction is confirmed, the two registers are locked.

<31:00> RECEIVED SBI COMMAND/ADDRESS
BUS REG <31:00> (SS32, SS33)
Each time a command/address is loaded into a DMA transaction buffer in the DC022, that command/address is also loaded into the corresponding DMA Command/Address error register. These error registers are actually TTL register files that are addressed by the upper two bits of the DC022 write address. SBI B <31:00> are written in these registers, with bits <31:28> being the SBI command codes and bits <27:00> the longword address. These error registers are locked if the SBIA or MBox detects a DMA error.



These four registers (DMAAID, DMABID, DMACID, and DMAIID) are used to save information about transactions in the DMA buffers. Each time a command/address is loaded into a DMA buffer, a copy of the command/address and SBI ID of the commander is saved in the two registers corresponding to the DMA buffer. If an error is dectected by the MBox or the SBIA after the transaction is confirmed, the two registers are locked.

<31:08> **MBZ (SS36)**
Must be zero.

<07:00> **RECEIVED SBI IDENTIFICATION**
BUS REG <07:00> <SS23)
Each time a command/address is loaded into a DMA transaction buffer in the DC022, the SBI ID is also loaded into the corresponding DMA ID error register, an extension of the DMA command/address error registers. These error registers are TTL register files like the command/address error registers, and addressed the same way, by the upper two bits of the DC022 write address. Bits <07:05> have the inputs at ground potential so they will always be read as zero. Bits <04:00> are loaded with REC SBI ID <04:00>. Like the DMA command/address error registers, these registers are locked if the SBIA or MBox detects a DMA error.